Task Recommendation for Flow Applications

Daire Ó Broin

A thesis submitted to the
Department of Computer Science,
University of Dublin, Trinity College
in fulfillment of the requirements for the degree of
Doctor of Philosophy

September 2009
Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other university, and that unless otherwise stated, it is entirely my own work.

I agree that Trinity College Library may lend or copy this thesis upon request.

______________________________
Daire Ó Broin

Dated: September 28, 2009
Acknowledgements

Firstly, I would like to thank my supervisor Dr Siobhán Clarke for her guidance, keen judgement, and support. I have learnt much from her over the years.

I would also like to thank Prof. Vinny Cahill for the part he played in my joining the DSG in the first place, and his direction in the CARMEN project.

I am very grateful to have been part of the DSG. It is, of course, the people who made the DSG a unique research environment. I would like to thank all my colleagues and friends in the DSG, and in particular, I would like to single out Dr Raymond Cunningham, Ivanna Dusparic, Alan Gray, Anthony Harrington, Andrew Jackson, Dr Andronikos Nedos, Sean Reilly, Vinny Reynolds, Mike Spence, and Ioanna Stamouli for making it such a great environment in which to work.

A special thank you to Dr Cormac Driver for illuminating the way to the end.

Thank you to my mother, brothers and sister for their support over the years. Thanks to my dad, whose encouragement so many years ago led me to where I am today.

The biggest thanks of all goes to my wonderful girlfriend Dúlra for her continual love and unwavering support, without which I would undoubtedly have gone down to the river to end it all.
Daire Ó Broin

University of Dublin, Trinity College

September 2009
Abstract

Flow is an immensely enjoyable mental state that is characterised by a “complete immersion in what one is doing” [61]. A model of flow, which has evolved over the last three decades, asserts that three key conditions must be present for a person to experience flow: a person must engage in a challenging task that requires skills and he must believe his skills match the challenges of the task; the task must have clear goals; and the task must provide immediate feedback. Flow applications, that is, applications that aim to assist their users to experience flow can be built for almost any activity, provided the activity supplies a set of challenges that require skills. Central to flow applications is the recommendation of tasks likely to produce the key conditions of flow. This observation led to the research question addressed by the thesis: how can tasks likely to produce the key conditions of flow be recommended?

Task recommendation for flow has a number of challenges. First, finding tasks whose challenges match a user’s skills; this entails taking a user’s perception of his current skills into account and using them to estimate the challenges of tasks, which involves a degree of uncertainty. As a user’s ratings are dependent on the skills he had when he rated the task, some recommendation algorithms are unsuitable, and this gives rise to a second challenge: accurately determining from tasks the information required by the recommendation algorithm. Poor recommendations will result if values of the attributes describing that item are misguided, and this cannot be prevented even with a domain expert. It is important for recommendation approaches to improve over time.
to minimize the absence of flow; only one existing approach does this but it has an underlying assumption that renders it unviable for task recommendation.

This thesis describes three approaches that use and adapt existing recommendation strategies and apply them to the problem of recommending tasks to support the creation of flow. It also describes an approach for improving recommendations continuously over time and as a result subsequent users of a flow application using this approach are likely experience more flow. In order to evaluate these approaches, the general method used was to build and deploy a flow application containing the recommendation strategy being tested. This involved the iterative design and development of a flow application for introductory programming. The effectiveness of the strategies was assessed by measuring relevant response variables as the flow applications were being used. A secondary research question was also considered in this thesis: how can applications that assist its users to experience flow be developed for any activity? To this end, the thesis describes a framework for developing flow applications.
Publications Related to this Ph.D.

List of Figures

2.1 Remaining in flow necessitates learning. 21

4.1 A simple example of a Bayesian network, taken from 79

4.2 The recommendation process used in the Stereotype approach. 86

4.3 Task confidence level: (i) “definitely won’t succeed/more than likely won’t succeed”; (ii) “definitely will succeed”; (iii) “stand a chance/probably will succeed”. 91

5.1 A class diagram depicting the key classes used in Inka 110

5.2 The session plan shows the name and progress (percentage complete) of each task. The main buttons are: go (show current task), edit (edit session), and snapshot (create context snapshot). 111

5.3 The student’s current task, showing the goal, progress, link to available materials (task resources), and a camera icon for creating a context snapshot. 111

5.4 An example of the form for updating progress. 112

5.5 The total time in the red is then the total length of red line segments; in this example, the total time is $a + b + c$. 118

5.6 An overview of a session using Inka 119

5.7 An example of a task resource. Task resources make a task easier, or make the goal of the task clearer. 121
5.8 The form for a creating a context snapshot. .................................................. 121
5.9 The percentage of time per subject with the conditions of flow absent. .......... 124
5.10 A class diagram showing the key classes in Musika .................................. 126
5.11 Task screen of Musika showing: comparison between goal notes (grey) and notes played (in green), % task complete (time elapsed), and score. 129

6.1 Inka 2 has additional visual feedback. .............................................................. 139
6.2 The frequencies of the task scores. ................................................................. 143
6.3 Switching tasks in Inka 2. ............................................................................. 150
6.4 The show task screen of Inka 2. .................................................................... 150
6.5 The frequencies of the task scores. ................................................................. 155
6.6 The responses to the questionnaire items. ..................................................... 157
6.7 The % accuracy demonstrated in the simulation. The x-axis of the graph represents the least number of ratings required before the strategy is permitted to change the current index. ............................................................... 162

7.1 A class diagram depicting a high-level overview of the application framework .............................................................................................................. 170
7.2 A high level view of the recommendation process ......................................... 175
7.3 A sequence diagram showing the key classes and methods involved in the Stereotype recommendation strategy ................................................. 177
7.4 A sequence diagram showing the key classes and methods involved in the MCR recommendation strategy ................................................................. 180
7.5 A class diagram showing the framework classes involved in supplying feedback. ........................................................................................................ 184
7.6 A class diagram showing the framework classes involved in measuring flow. ................................................................................................................ 187
7.7 A class diagram showing the relationships between the activity elements
    and the key behaviour and key attributes of each activity element. . . . 190
7.8 The context model, showing some context classes available in the frame-
    work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 192
7.9 Any context can be extended so that it can be updated manually . . . 193
7.10 The extensions to the framework for Relaks. . . . . . . . . . . . . . . . 200
7.11 The ThoughtStream\textsuperscript{TM}– a GSR device. The band with the metal plates
    on the left-hand side of the picture is placed on to the hand. . . . . . 201
7.12 The extensions to the framework for Joga. . . . . . . . . . . . . . . . . 205
List of Tables

3.1 Summary of the support of flow application requirements 68

5.1 Time with the conditions of flow absent broken down by session and subject 123

6.1 A summary of the responses to the questionnaire items (in percentages) 145

6.2 The precision of the MCR recommendation strategy 154

6.3 The average scores of the questionnaire items 157

7.1 The options for the form to measure flow 188

7.2 The complete list of properties of the application framework relating to task recommendation that can be set, along with a description of their purposes, the values they can have, and an example of each 195

7.3 The complete list of properties of the application framework relating to improving recommendations that can be set, along with a description of their purposes, the values they can have, and an example of each 196

7.4 The list of properties relating to feedback, along with a description of their purposes, the values they can have, and an example of each 197
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>iii</td>
</tr>
<tr>
<td>Abstract</td>
<td>v</td>
</tr>
<tr>
<td>List of Figures</td>
<td>viii</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xi</td>
</tr>
<tr>
<td>Chapter 1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Flow</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Task Recommendation Challenges</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Task Recommendation for Flow Applications</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Context-aware Computing</td>
<td>7</td>
</tr>
<tr>
<td>1.5 Introductory Programming</td>
<td>8</td>
</tr>
<tr>
<td>1.6 Frameworks</td>
<td>11</td>
</tr>
<tr>
<td>1.7 Contributions</td>
<td>12</td>
</tr>
<tr>
<td>1.8 Where Does This Thesis Fit?</td>
<td>14</td>
</tr>
<tr>
<td>1.9 Thesis Outline</td>
<td>15</td>
</tr>
<tr>
<td>Chapter 2 Flow</td>
<td>17</td>
</tr>
<tr>
<td>2.1 The Flow Model</td>
<td>17</td>
</tr>
<tr>
<td>2.1.1 The Characteristics of Flow</td>
<td>18</td>
</tr>
</tbody>
</table>
Chapter 2 Flow in Computer Mediated Environments

2.1 The Conditions of Flow

2.1.2 The Conditions of Flow ........................................... 19
2.1.3 Flow and Learning ................................................ 20
2.1.4 Measuring Flow .................................................... 21
2.1.5 Limitations of the Flow Model ................................. 30

2.2 Flow in Computer Mediated Environments ....................... 30

2.2.1 Hyperlead .......................................................... 31
2.2.2 IT-Emperor ......................................................... 32
2.2.3 Super Tangrams ................................................... 33
2.2.4 Pearce et al. ....................................................... 34
2.2.5 SingStar ............................................................ 37
2.2.6 Burleson ............................................................ 37

2.3 Conclusion ............................................................. 40

Chapter 3 Literature Review on Task Recommendation ............ 44

3.1 Approaches to Task Recommendation ............................. 44

3.1.1 Content-based Methods .......................................... 46
3.1.2 Collaborative Methods ......................................... 48
3.1.3 Hybrid Methods .................................................. 51
3.1.4 Recommending Tasks ........................................... 54

3.2 Task Recommendation in Education .............................. 56

3.2.1 KBS-Hyperbook .................................................. 57
3.2.2 ELM-ART .......................................................... 59
3.2.3 ADAPTS ........................................................... 61
3.2.4 Personal Recommender System for Learning Networks .... 63

3.3 Recommendation Improvement .................................... 64
3.4 Conclusion ............................................................. 67
Chapter 4  Design of a Task Recommendation System for Flow

4.1  Selection of Task Recommendation Strategies
4.1.1  Collaborative Approaches
4.1.2  Content-based Approaches
4.1.3  Hybrid Approaches
4.1.4  Multi-criteria Decision Making (MCDM)

4.2  Task Recommendation
4.2.1  Stereotype
4.2.2  Multi-Attribute Utility Theory (MAUT) Approach
4.2.3  Multi-Criteria Recommendation (MCR) Approach

4.3  Improving the Recommendations
4.3.1  Clear Goal and Feedback
4.3.2  Confidence Level

4.4  Conclusion

Chapter 5  Pilot Studies

5.1  Pilot Study 1
5.1.1  Setting
5.1.2  Sample
5.1.3  Materials
5.1.4  Variables/Measures
5.1.5  Procedure
5.1.6  Findings

5.2  Pilot Study 2
5.2.1  Materials
5.2.2  Variables/Measures
5.2.3  Procedure
5.2.4  Findings
Chapter 1

Introduction

*Flow* is a highly desirable state, characterised by a “complete immersion in what one is doing” [61]. This thesis is concerned with assisting people to experience flow.

This chapter introduces *flow* and describes the key conditions necessary for flow. Central to *flow applications*, that is, applications that aim to is the recommendation of tasks likely to produce the key conditions of flow. The challenges of task recommendation are outlined, followed by the requirements of a flow application, and how tasks can be recommended in such an application. This is followed by a brief introduction to context-aware computing, an area that’s important for two of the requirements of flow applications.

The flow applications studied in this thesis are primarily for the activity of introductory programming, and the next section describes the motivation for a working on this topic and some related research in introductory programming. This is followed by an overview of frameworks, pertinent because an output of the thesis is a framework for the development of flow applications. Next, a summary of the contributions of the thesis is given, followed by a reflection on where this thesis fits. The chapter closes with an outline of the thesis.
1.1 Flow

Research into optimal experience or flow, as it became known\(^1\), began more than thirty years ago. Flow is an immensely enjoyable mental state that is characterised by a “complete immersion in what one is doing” [61]. Indeed, it is so enjoyable that people invest considerable amounts of time and money “for the sheer sake of doing it” [59]. That is, people do an activity for itself “and not for the usual rewards of everyday life, like a paycheck or a promotion” [61]. Also, the results of studies have suggested that flow is a universal state, that it is the same for people regardless of their culture, gender and age [61].

By studying a great many different activities, such as chess, rock climbing, and surgery, it was discovered that the experience of flow has essentially the same elements, regardless of the activity [61]. The most universal of these elements of flow is that one is completely immersed in an activity. This immersion means that one’s attention is completely consumed by the task at hand, and as a result one tends to “forget the worries and concerns that take up our attention in ordinary life” [61]. Another element of flow is that one tends to have a sense of control, or, more precisely, one does not consider the possibility of losing control. The final element of flow is one’s sense of time is distorted, usually so that time seems to pass much faster than usual.

A model of flow has been developed over the last thirty years, which, in addition to describing the elements of flow, also supplies an answer to an important question: what conditions must be present for a person to have a flow experience? The model asserts that there are three key conditions [57]. First, a person must engage in a challenging task that requires skills and he must believe his skills match the challenges of the task. Second, the task must have clear goals. A goal is a desired state which a person aims to reach; if the goal is clear he will know, at any moment, whether or not he has reached

\(^{1}\)Csikszentmihalyi used the term flow because in the early studies of optimal experience, people often used it to describe the experience [54]. This experience, though frequently involving doing something challenging, seemed effortless at the time – like “being carried by the flow of a river” [61].
it. Some tasks supply a full set of goals (as with a musical score which specifies the next notes to play). Other tasks supply some goals and the person can set the intermediary goals himself (as with a rock climber who chooses the next hold to go for). And third, the task must provide immediate feedback – that is, information letting the person know how well he’s doing. For example, a tennis player can see where he hit the ball; a musician can hear if he played the right notes.

To go from ordinary experience to flow experience, we must ensure the key conditions of flow are present. Unfortunately, this is not easy [59]. Applications that aim to assist their users in having a flow experience can be built for almost any activity, provided the activity supplies a set of challenges that require skills. In this thesis, we refer to such applications as flow applications. Central to flow applications is the recommendation of tasks likely to produce the key conditions of flow.

1.2 Task Recommendation Challenges

Recommendation systems, most commonly by estimation of ratings for items that have not been seen by a user, is the central problem of the area of recommender systems [8]. A well-known example of a recommender system is Amazon.com [134], which recommends books, CDs, and other items. Recommendation approaches fall into three categories: content-based, collaborative, and hybrid [8]. In content-based approaches, items that score highest against a user profile are recommended. Collaborative approaches recommend items to a user based on whether the items were liked by similar users. Hybrid approaches mix content-based and collaborative approaches.

A crucial difference between recommending tasks to produce flow and recommending other items is the challenge of finding tasks whose challenges match user skills. This means that a user’s perception of his current skills must be taken into account and used to estimate the challenges of tasks, and this involves a degree of uncertainty.
Furthermore, as a user learns, the level of his skills changes and, essentially, the user becomes a different person. This has been called the “stability versus plasticity” problem [41]. While with other domains, changes to the user can occur over time, for example, as the user’s interests drift, nowhere are user changes more sharply evident than with knowledge and skills, where large changes can occur even within a single session.

This means that a user’s ratings are dependent on certain characteristics of the user when he rated the item. In the case of recommending tasks for flow, it means that a user’s ratings are dependent on the skills he had when he rated the task. An implication of this problem is that it makes some recommendation algorithms unsuitable for recommending tasks to produce flow. In particular, collaborative approaches are unsuitable for this reason, and this means a content-based approach is required, giving rise to a second challenge of task recommendation: accurately determining from tasks the information required by the recommendation algorithm.

The necessary information is much more difficult to obtain for tasks than for the usual items recommender systems are built for. For example, the necessary attributes of movies can be easily extracted from an internet movie database. With tasks, some attributes of particular importance to flow are the skills a task requires. Skills can be selected either manually or automatically. In a manual approach, the skills are manually associated with a task. In some cases, such as in the KBS-Hyperbook [97], weights are used to specify the importance of each of the skills in completing the task. Errors cannot be prevented even when attribute values are selected by a domain expert.

In an automatic approach, skills can be automatically associated with a task, but this is suitable for very few domains [192]. Moreover, existing approaches, such as [192] for the domain of programming, determine only whether a skill is required by a task, and not the importance of the skill. This is no small matter. Suppose a user has little confidence in a certain skill; whether that skill features very heavily or hardly at all in the task will greatly affect how difficult the user finds the task.
Thus, in a content-based approach, poor recommendations will result if values of the attributes describing that item are misguided, and this cannot be prevented even with a domain expert. Recommendation approaches in which recommendations do not improve over time can have a most undesirable consequence. Systems using such approaches will respond in the same way to users who find themselves in similar situations to previous users. Certainly, no problems result when the previous users received good recommendations. However, when the previous users received poor recommendations, the current users will receive the same poor recommendations, and this leads directly to the absence of the conditions of flow. This undesirable consequence provides considerable motivation to identify and improve upon the sources of poor recommendations, so that recommendations continuously improve over time, and the key conditions of flow are increasingly present.

The main limitation of existing approaches to improving recommendations is that, while each of these approaches has the potential to give better recommendations than a system that doesn’t use the approach, only the collaborative approaches continuously improve the recommendations over time. However, as already mentioned, collaborative approaches are unsuitable to recommending tasks to produce the conditions of flow because of the dynamic user characteristics.

To improve a recommendation, it is necessary firstly to determine when a recommendation is poor, and secondly for each item that was poorly recommended, to find a way to alter the values of its attributes so that subsequent recommendations of the item will be better received. A considerable difficulty with this is that there is no way to produce an evaluation function which could take an item and determine its quality. The item must be recommended to a number of users and their response will determine its quality.
1.3 Task Recommendation for Flow Applications

In order to fulfil its goal of assisting users in having flow experiences, a flow application must satisfy a number of requirements. These were identified using an application-led approach, by designing flow applications for a number of diverse activities, and refining the requirements with each application. The following requirements for flow application were identified:

R1 The most important requirement is that the application must influence (that is, support the creation of) the three key conditions of flow. This requirement is composed of three sub requirements:

R1.1 The application must recommend tasks since to go into flow one needs, at any moment, to be engaged in a task. Moreover, these tasks must have clear goals (as described in Section 1.1).

R1.2 The recommended tasks must be chosen so that the challenges of the task (as the user perceives them) are balanced with the user’s skills (as he perceives them); this is one of the key conditions of flow.

R1.3 The application must supply or enhance feedback from the recommended tasks, since receiving feedback is another of the key conditions of flow.

R2 The quality of the recommendations should improve as the application is used. That is, situations in which the recommendations led to the conditions of flow being absent will recur with subsequent users, and when they do, it should be more likely that this time, the conditions of flow will be present.

R3 The application must measure conditions of flow so that it can become aware of their absence and subsequently take action to aid their return.
Task recommendation is central to a flow application. How exactly a recommended task is represented depends on the activity and on the environment in which the activity takes place. If the activity takes place entirely on a computer, the tasks might be represented as videos, as they are in the video game Grand Theft Auto [87]. An activity that involves interacting at intervals with a computer, such as a flow application for cookery, might represent tasks as a mixture of text and pictures, and keep track of completed subtasks, like in the Electronic Performance Support System, ADAPTS [34]. Activities in which users might not be able to or wish to look at a screen, such as yoga, could be represented using audio.

An activity that is tied to a particular environment might represent tasks by physically embedding them in the environment. For example, for the activity of climbing, a sequence of grips on a climbing wall could light up, specifying the task (the climber must use only the grips that are lit up). Some activities may benefit by representing tasks in augmented reality, such as superimposing graphics on reality, for example, for service and maintenance on cars and aircraft [178]. Still other activities may benefit by representing tasks in virtual reality, for example, if it is dangerous for the user to perform the tasks in reality until he has developed sufficient skills, such as in surgery [11].

1.4 Context-aware Computing

Context is important for two of the requirements of flow applications. Probably the most frequently cited definition of context is supplied by Dey, who defines context as “any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves” [66]. Context-aware applications can adapt their behaviour by taking into account changing context. Since
the term context-aware first appeared (in [186]), many context-aware applications have been developed, which, as a group, have utilised a diverse range of context, including location, identity, time, temperature, noise levels, user’s goals, and user’s emotional state.

Some examples of context-aware systems include: a fieldwork assistant application that assists an archaeologist by automatically recording the time and the location of artefacts as they are unearthed [183], a context-aware pill bottle that issues a patient with reminders to take his pills and alerts the patient’s family if the pills aren’t taken [10], and a movie recommender system that takes context into account, such as place (movie theatre/at home) and companion (none/boyfriend or girlfriend/friends/family) [7].

The two requirements of flow applications for which context is important are measuring flow and providing feedback. Measuring whether the user is currently in flow can be updated either manually by the user or potentially automatically (see Section 2.1.4). As for providing feedback, it is possible in many situations to use some context available to the system to supply users with feedback they couldn’t otherwise obtain. Some examples include: a system that gives a user visual and audio feedback letting him know how close he came to reaching the goal of performing a certain kind of karate punch [127], SingStar, which gives the user visual feedback letting her know whether she hit the note or if not how close she was [5], and Nike + iPod which uses a small accelerometer attached to, or embedded in, a shoe to supply a user with visual and audio feedback about goals relating to time, distance, and calories burned [104].

1.5 Introductory Programming

The flow applications studied in this thesis are primarily for the activity of introductory programming. This activity was chosen for two main reasons. First, it has the potential
to provide almost anyone who engages in it with flow experiences, yet rather that experiencing flow, many people experience frustration. Second, it has a moderately sized set of well defined skills, making it more manageable, a desirable characteristic for the first activity studied, that is, before attempting to generalise to other activities.

Beginner programmers face many challenges such as forming structured solutions to problems, understanding how programs are executed, and learning a rigid and perhaps confusing syntax, and these challenges can be overwhelming and discouraging [113]. Many approaches have been taken to make programming more accessible, that is, to reduce the challenges. One approach is to use a simplified language that familiarises users with fundamental concepts of programming, and enables a smooth transition to a more general purpose language such as Java or C++; examples include BASIC [126] and Blue [120].

Another approach is to avoid syntax problems entirely by means of graphical programming languages, in which it is impossible to make a syntax error. An example of this is LogoBlocks [22] in which graphical blocks that represent Logo [165] code can be dragged around the screen and connected together to form a program. A palette of blocks on the side of the screen shows the user the available commands, and commands and conditionals that require parameters have shapes with cut-outs making it clear which types of blocks can be connected to a given block.

Another means of reducing the challenges of programming is rather than using an Integrated Development Environment (IDE) designed for professional programmers, a more accessible IDE can be used. An example of this is BlueJ [119], which is an IDE for Java, specifically designed for beginner programmers. The main window in BlueJ depicts a UML class diagram of the application structure with which users can interact. Users can, for example, create a new instance of a selected class or inspect a selected object. Users can also invoke any public method of a selected object, enabling the introduction of the usual text interfaces or GUIs to be delayed until users have developed
a deeper understanding of classes and objects. BlueJ also makes more advanced skills such as unit testing accessible to the beginner.

The abstract nature of programming also provides challenges. Most introductory programs involve performing arithmetic on numbers and storing the results in invisible registers; this causes difficulties for students in understanding their programs and correcting any problems they contain [113]. Micro-worlds, containing characters and objects that can be controlled by a user, aim to make programming more concrete. The characters in micro-worlds are usually capable of few actions, leading to small, more easily mastered languages; moreover, they typically include simulators enabling students to watch their programs in action [113].

An example of a micro-world based system is Scratch [149], which uses a similar approach to program construction as LogoBlocks; the most notable difference is that code is associated with different sprites that can move independently. Scratch enables its users (mainly children) to easily create games and videos, which can motivate them to learn more. A similar project is Alice [51], aimed at older children and college students. It is more advanced than Scratch, which trades intuitive ease-of-use for power. In Alice users can use graphical programming and text-based programming, and Alice programs, written entirely in Java, can even be imported into the NetBeans IDE, once a user’s skills have developed sufficiently.

In addition to reducing the challenges of programming, tools can also provide feedback that let the user know if he has reached his goal, and more importantly, if he has not reached his goal, to give him information about what happened that might indicate what changes he should make to the program. For example, Alice animates all its commands giving the user a visual representation of the effect of each. In BlueJ, objects can be inspected, showing the value of each variable, and by inspecting before and after a method is invoked, useful feedback can be obtained. In this way, BlueJ shows static visualization of objects, but this can be extended using Jeditor [151] which
enables dynamic visualization of objects, giving the user a visual representation of the effect of each line of code.

Tools such as those described above provide environments in which tasks can be done, but suitable tasks still need to be recommended to produce the key conditions of flow. However, some of these environments offer the advantage of additional feedback (one of the key conditions flow). In addition, setting some tasks in such environments can make them more desirable to students. Typical introductory programming assignments such as “sort a list of numbers” or “generate the sum of the first 1,700 integers” often fail to engage students [114]. On the other hand, environments like Robocode [156] (in which students use Java to design tanks that battle to the death against other students’ tanks), or Alice 3 which uses 3D graphics from the best selling videogame Sims 2 [12], can be considerably more motivating for students.

1.6 Frameworks

A software framework is defined as “the skeleton of an application that can be customised by an application developer” [107]. It is also commonly defined as “a set of cooperating classes that make up a reusable design for a specific class of software” [88]. Rather than reusing single components, a framework enables “whole software (sub-)systems including their design” to be reused [176]. Frameworks increase modularity using stable interfaces that encapsulate mutable implementation details [79].

This modularity can improve the quality of software by containing the impact of changes in design or implementation, which in turn makes software easier to understand and maintain [79]. Frameworks increase reusability by defining generic components that can be reused to create new applications, making it unnecessary to recreate and revalidate “common solutions to recurring application requirements and software design challenges”, which can yield considerable improvements in programmer productivity,
and increase the quality, performance, and reliability of software [79]. Frameworks also reuse analysis; they describe a vocabulary for discussing a problem [107].

A notable characteristic of frameworks is inversion of control [107], which means that it is the framework, and not the particular application that determines the application-specific methods to call [79]. As the creation of new applications seems endless, new components will inevitably be required [107]. To this end, frameworks increase extensibility by enabling applications to extend its stable interfaces, and by allowing new components to be plugged into the framework using object composition [79].

Although developing complex software is difficult, it is even more difficult to develop high quality, extensible, and reusable frameworks for complex application domains [79]. However, this seems to be outweighed by the incentives offered by frameworks: enhanced reusability and extensibility, improved quality of software, and the reduction in the cost of developing applications from a specific class of software by “an order of magnitude” [181]. In order to enable developers of flow applications to readily use the results of this thesis, a framework for flow applications was developed (see Chapter 7).

1.7 Contributions

Flow is a valuable state, but “[i]t is not easy to transform ordinary experience into flow experience” [39]. The research question addressed by the thesis is: how can tasks likely to produce the key conditions of flow be recommended? In providing an answer this question, this thesis has addressed a number of shortcomings of the state of the art, described below, and has made the following contributions:

- Task recommendation to support the creation of flow

Existing approaches to task recommendation are generally based on automatically updating a user skills model, and there is no guarantee that this will coincide with a user’s perception of his mastery of the skills, or they are limited by low
precision, which is a concern for maintaining the “delicate balance” [54] between challenges and skills required for flow. This thesis describes three approaches that use and adapt existing recommendations strategies and apply them to the problem of recommending tasks to support the creation of flow. In addition, it is possible that, for certain domains, other factors (besides the three key conditions of flow) could make it more likely still that a person would experience flow. The three approaches to task recommendation described in this thesis can be readily extended to incorporate such factors.

- **Improving recommendations in flow applications**
  In a flow application, the quality of task recommendations greatly affects its primary goal (to assist the user in experiencing flow), and consequently improving poor recommendations is vital. The main limitation of existing approaches to improving task recommendations is that only the collaborative approaches continuously improve the recommendations over time, and collaborative approaches are unviable for task recommendation because they have an underlying assumption that users’ characteristics remain static. This thesis describes an approach for improving recommendations by analysing context snapshots to identify the source of poor recommendations and then making changes to the tasks, or suggesting changes that need to be made to the tasks, in order that better recommendations for subsequent users will result.

- **Iterative design and development of a flow application**
  This thesis describes the iterative design and development of a flow application (Inka) for introductory computer programming. Inka was deployed in an authentic environment and built on the limitations of existing flow applications, including the methods of measuring flow, task recommendation, and improving recommendations.
• **A framework for developing flow applications**

No framework for supporting the development of flow applications currently exists. Frameworks, such as this one “allow people with superior creative skills to build innovative applications without having to be expert programmers.” [67]. They can reduce the cost of developing applications from a specific class of software by “an order of magnitude” by enabling both design and code to be reused [181], and they can improve the quality of software [79]. The framework is generic – it can be used for almost any activity that has challenges requiring skills, and it is extensible so that it can deal with the differences introduced by different activities. A simple example of this is that different kinds of context can be introduced to supply feedback to suit a given activity.

1.8 Where Does This Thesis Fit?

Interdisciplinarity refers to the amalgamation of knowledge from a number of different areas, especially to work towards a solution of an actual problem [27]. Many problems have required the insights of other disciplines [173]. It is the problem itself that “points out, perhaps demands which disciplines and what methods might best be [employed]” [27]. This thesis is concerned with the problem of assisting people to experience flow. Many different areas are relevant to this problem. Among them are: flow theory (for the flow model and approaches to measuring flow), recommender systems (for approaches to recommending tasks), context-aware computing (for measuring flow and providing feedback), augmented reality (for enhancing feedback), and technology enhanced learning (for approaches to recommending tasks and providing feedback).

The challenges of interdisciplinarity include: “different methods and operational objectives within and between disciplines” [27]; “different ‘languages’ within the disciplines and between the disciplines” [27]; that “it takes a great deal of time and effort to
fully engage another discipline, to sufficiently understand its language, concepts, and methods” [136]; and that disciplines can have different “sets of values” [21]. Sperber maintains that the only way in which attention is paid to interdisciplinary work is if a different version of the work is produced for each discipline involved, and each version is submitted to a journal concerned with that single discipline [194].

This thesis focuses on one aspect of the larger problem of assisting people to experience flow: task recommendation. As a result, the work presented in this thesis fits mainly into the area of recommender systems and the area of technology-enhanced learning (TEL). In recommender systems, most research to date has focused on more abstract tasks, such as “find new items”; few specific tasks (like “find a movie for a first date”) have been explored [185]. This thesis describes one such specific task: recommending tasks for flow. As for technology-enhanced learning, this work has added a tool for enhancing learner experience, in particular for introductory programming, and also framework to support the development of similar tools for other activities.

1.9 Thesis Outline

This thesis is organised as follows. Chapter 2 describes the flow model and methods of measuring flow, and reviews systems that aim to influence the conditions of flow while the user is engaged in an activity in a computer mediated environment. Chapter 3 presents a literature review on task recommendation. It examines approaches to task recommendation, reviews some representative systems, examines approaches to improving recommendations, and identifies limitations of existing approaches in recommending tasks to produce the key conditions of flow. Chapter 4 details the design of an extensible task recommendation system for recommending tasks to produce the key conditions of flow.
Chapter 5 describes two pilot studies, aimed at measuring the effectiveness of two of the task recommendation strategies described in this thesis for producing the conditions of flow: the Stereotype strategy and the MAUT strategy. To this end, two flow applications were built for this Inka (for the activity of introductory programming) and Musika (for the activity of playing or practising music). Chapter 6 describes three studies, which address a number of shortcomings of the pilot studies, and evaluate the effectiveness of the MCR task recommendation strategy described in this thesis as well as the strategy for improving recommendations. To this end, another flow application, Inka 2, was developed (also for the activity of introductory programming.)

Chapter 7 describes an output of the thesis: a framework for flow applications, the aim of which is to facilitate the development of flow applications. Two flow applications developed using the framework (Joga, for the activity of yoga, and Relaks for the activity of relaxation) were used to describe how the framework can be reused and extended. Finally, Chapter 8 concludes the thesis by detailing its achievements and discussing possible future work.
Chapter 2

Flow

This chapter describes the flow model and methods of measuring flow, and reviews systems that aim to influence the conditions of flow while the user is engaged in an activity in a computer mediated environment.

2.1 The Flow Model

The flow model was developed by studying people who spent a great deal of time engaged in an activity for no other reason than the “sheer sake of doing it” [59]. Subsequent research suggested that flow (a highly enjoyable mental state characterised by a “complete immersion in what one is doing”) is universal, that is, the experience is the same for people of different ages, genders, and cultures [61]. Moreover, flow has essentially the same elements regardless of the activity the person is engaged in [61]. The flow model has been refined over the years and the most recent publications (for example [61], [57]) have explicitly divided these elements into two sets: the characteristics of flow and the conditions of flow. The characteristics of flow describe the experiential elements of flow. The conditions of flow describe the conditions that must be met in order for flow to occur. Flow and learning are inseparable: in order to continue to
experience flow, a person must continue to learn new skills. In order to measure flow, three main methods have been used. These are interviews, surveys, and a self-report technique known as the Experience Sampling Method (ESM).

2.1.1 The Characteristics of Flow

The characteristics of flow describe the nature of flow, that is, the experiential elements of it. The characteristics of flow are: the merging of action and awareness, a sense of control and an altered sense of time.

The Merging of Action and Awareness

The most universal characteristic of flow is “a complete immersion in what one is doing”, described by many people as a “merging of action and awareness” [61]. This immersion in what one is doing means one’s attention is entirely absorbed, causing one to forget “worries and concerns that take up our attention in ordinary life” [61]. It also means one forgets oneself. Thoughts about oneself are usually negative, and these thoughts result in negative emotions. But in flow, this constant stream of negative thoughts is absent. A consequence of forgetting oneself is a “feeling of transcendence”, where one feels part of something bigger [61]. The climber “feels at one with the rock” [61]; the surgeon has “the sensation that the entire operating team is a single organism” [59].

A Sense of Control

The sense of control experienced in flow is more precisely described as the absence of anxiety about losing control – one does not consider the possibility of losing control [59]. It is a sense that the situation is in hand, that no matter what happens next, one knows how to deal with it [152]. This sense of control is also experienced by people in risky activities such as rock climbing, hang gliding and race-car driving. Rock climbers,
for instance, maintain that climbing a mountain is safer than crossing a busy street since they can predict – and deal with – the dangers far more easily on a mountain [59].

An Altered Sense of Time

In flow, one’s sense of time is distorted. Typically, time seems to pass much quicker than usual. Occasionally, the reverse occurs – a figure skater may report a turn that took a second to perform seemed as though it took 10 seconds [53].

2.1.2 The Conditions of Flow

Research has indicated three key conditions required to experience flow: an activity in which the perceived challenges are balanced by the perceived skills, a clear set of goals, and feedback [57]. If these three key conditions of flow are present, it is likely that a person will experience flow. This, as discussed in Section 2.1.5, does not mean that other conditions do not exist. However, at present, the three key conditions are the only conditions that are part of the flow model, and consequently are the only conditions considered in this thesis. The three key conditions are described below.

Perceived Challenges Balanced by Perceived Skills

Every activity has a set of “opportunities for action, or ‘challenges’ that require appropriate skills to realise” [59]. A challenge can be thought of as a goal [105]. Examples of challenges include closing a business deal, playing a particular piece on the piano, and climbing to the top of a cliff. In order to experience flow, one must engage in an activity in which the perceived challenges of the activity are matched by one’s perceived skills. If the perceived skills are insufficient, anxiety results, while if the skills greatly exceed those required, the result is boredom [59].
A Clear Set of Goals

The value of a goal (a desired state) lies in its ability to channel a person’s attention [57]. In contrast to everyday life where it is common to have incompatible goals vying for one’s attention or to be unsure of one’s purpose, in flow it is always clear what has to be done [53]. The clear set of goals can be supplied by the activity (for example, a musical score indicates what notes to play next) or the person can decide upon intermediary goals himself (for example, a climber who chooses the next hold) [55]. The end goals – coming to the end of a musical piece, arriving at the top of the mountain – are not the reason for doing these things, they are “simply an excuse for the activity”, and provide a finishing point for the activity [57].

Feedback

Feedback informs a person about how well they are doing in an activity [57]. The kind of feedback typically does not matter, as long as it is logically connected to a goal [59]. The task can supply the feedback (as with the tennis player who can see after each shot if the ball went where it was supposed to) or the person can have a standard internalised in his mind, and he can tell how well his actions measure up (as with the poet who reads the last line he wrote and knows if it is right) [55].

2.1.3 Flow and Learning

Flow and learning are inextricably linked. Every flow activity leads to growth of the self [59]. Figure 2.1 explains how the process works. Suppose a person A takes up a flow activity – tennis, for instance. At the beginning A has no skill to speak of, but the challenge he sets himself is simply to hit the ball over the net, which is about right for his minimal skills, so he is likely to experience flow (A1). From here, one of two things will happen. The first is that A continues to practise; his skills will inevitably improve
and A will find himself bored with the challenge ($A_2$). The second is that supposing the challenge is raised significantly – for instance, A might play an opponent vastly more skilled. Now the challenges facing A greatly exceed his skills, and he becomes anxious ($A_3$).

Both of these negative states motivate A to return to flow. From $A_2$, his only option is to increase the challenges – for instance playing someone only slightly better than himself. From $A_3$, he must increase his skill or reduce the challenges. In both cases, A returns to flow ($A_4$). But at $A_4$, flow can’t last – just as it couldn’t last at $A_1$. Before long, the cycle will repeat itself. Thus, maintaining flow requires that one’s skills continually improve, that is, flow and learning are inextricably linked.

### 2.1.4 Measuring Flow

Mobile context-aware flow applications need to measure flow so that when they observe a user is not in flow, they can take remedial action. Three major methods have been used to measure flow: interviews, surveys, and a self-report technique known as the
Experience Sampling Method (ESM). The advantages and limitations of each of these methods are described below.

Interviews

The research leading to the first publications on flow (first a journal article [58] and then a book [54]) was based entirely on surveys and interviews. While interviews were indispensable initially, they have severe limitations [62]. When people recollect and describe an optimal experience, the result is usually “quite stereotyped and uninsightful” [56]. When describing any experience, people are liable to fall victim to the “vagaries of memory” or inadvertently distort what occurred [62]. Moreover, interviews are capable of uncovering only the “most general, most obvious features of the landscape” [62].

Surveys

Surveys usually involve getting subjects to fill out a questionnaire either by mailing it to them or by asking the subjects to fill it in online. Researchers who have employed surveys to measure flow include Ghani and Deshpande who studied flow in Human-Computer Interaction [91] and Novak et al. who studied flow in online consumers [158, 102]. One example of a questionnaire designed to measure flow is the Flow Scale [139], which asks respondents to provide estimations of the frequency that they experience each of set of elements of the flow experience.

Limitations of surveys include the kind of questions that can be asked (usually only closed questions are included) and that surveys on flow typically ask subjects to rate factors of a generalised experience [83]. For instance, all the questions in Ghani and Deshpande’s questionnaire are about the subjects’ feelings while “using computers” [91]. And in the survey by Novak et al., the questionnaire items are about “using the web”, for example, one item subjects had to rate was “using the Web challenges me” [158]. Whether a person is in flow or not depends on several conditions, such as if their
skills are in balance with the current challenges of the activity. These conditions can change from moment to moment, but the survey demands a single answer, requiring the subjects to somehow amalgamate the many specific experiences they have had doing the activity in question into a single generalised experience. It is therefore unclear exactly what has been measured.

Not all surveys attempt to measure flow by surveying generalised experience. In a study of optimal experience on the web by Chen et al., the subjects were directed to recall specific experiences. For instance, subjects were asked if they have ever experienced the feeling of “time going too fast”. If they answered in the affirmative, they were asked to describe the last time it occurred. Asking the subject to recall the last time the experience occurred forces him to focus on a specific experience. But even if such measures are taken, the method remains limited. Surveys, like interviews, are limited by the “vagaries of memory” and by subjects inadvertently distorting their experience.

Another method of using surveys to measure flow is to design an activity for subjects to do in a controlled environment, and then ask them to fill in a questionnaire immediately afterwards. For example, in a study by Ghani et al., the subjects were set the task of creating a resumé using Microsoft Works. The subjects were asked only about the experience they have just had, so the problem of generalised experience is circumvented. However, the validity of this kind of study can be brought into question on the grounds that it is conducted in a controlled environment, and flow is a context-specific experience. That is, if such a study took place in a natural setting, the results might differ. More importantly, psychological states can change from moment to moment. This means a subject’s state may have changed many times as he did the activity, and although the questionnaire is administered directly after the activity, it will fail to capture these variations. Furthermore, even if such a questionnaire asked for details of the different states experienced during the activity, the problem of a subject
forgetting details of a memory – or distorting it – would arise once more.

A final limitation of surveys, which also holds for interviews, is that because both methods wait until after the activity to take measurements, the acquired information is less recent and hence less valuable. While the information could be used to improve a user’s subsequent experiences, it cannot be used to alter the user’s current experience. That is, if it is determined that the user was not in flow, it is too late to supply him with something that might help him go into flow, such as a more suitable task or more feedback.

**Experience Sampling Method (ESM)**

The Experience Sampling Method (ESM) was developed to overcome the limitations of interviews and surveys – chiefly the problem of people forgetting or distorting memories and the problem of requiring people to amalgamate many specific experiences into a single generalised “typical” experience [152]. It was designed to measure flow in everyday life [62] and it has been used in numerous studies to that end, for example [138], [131] and [77]. The validity of the ESM has been demonstrated in numerous validation studies; a review of some validation studies of the ESM can be found in [63].

Each subject of an ESM study is supplied with a pager and a booklet of identical self-report forms. The subjects are paged at random times during the day, usually about seven times a day for a week. When the pager signals, the subject fills in one self-report form, called an Experience Sampling Form (ESF), from the booklet. The ESF has over 30 measurement points for assessing the subject’s mental state. It includes open-ended items (such as “What was the main thing you were doing?”) and a number of 7 point Likert scales to measure the intensity of a range of emotions (such as happiness and alertness). The result is that the ESM can capture “high-resolution description[s]” of the subject’s mental states as they occur [62]. Moreover, because the subject is estimating his current state, he does not rely on memory, so the problem of
forgetting or distorting memories is minimised.

A number of different candidates exist for measuring flow from the items of the ESF. It could be argued that any one of the following could provide a measure of flow: the intensity of happiness, the degree of concentration, the measure of lack of self-consciousness, the amount of control, or the extent to which a person disagreed with the item “I wished to be doing something else”. Or, a value of flow could be computed by combining the values of each of these variables.

Csikszentmihalyi argued that while any of the above measures for flow could have been used, it would have been tantamount to defining flow as happiness or as high concentration, or as one of the other candidates [62]. This would mean the theoretical model of flow (which had been developed from extensive interviews) could not be falsified by the data gathered from the ESFs. The theoretical model predicted that flow would occur if challenge and skills were balanced. By defining flow in terms of this balance, the gathered data would either verify the model – that is, demonstrate that a balance of challenges and skills is correlated with a positive state – or else the data could suggest how the model might be improved upon. Indeed, the results of studies such as [131] led to a slight adjustment of the theoretical model for flow in everyday life, namely that to indicate flow, challenges and skills must not only be in balance, but also they must both be above a certain level (usually defined as the person’s weekly average).

A considerable advantage of the ESM is its flexibility. While some researchers have used the ESF (for example, [123]), others have used variations of it, or have developed their own self-reports tailored to their research goals (for example, [50]). In addition, when signalling the subject to fill in a form, there are two other methods aside from signalling at random times; these are signalling at regular intervals and signalling when events of interest occur [206]. Furthermore, instead of using a pager and a paper booklet, digital versions of the ESM have been produced on PDAs and
mobile phones (for example, [19] and [86]). This introduces new possibilities such as different alert types for different situations (for example, a vibration instead of audio if the subject is in a meeting), different output types (for example, using audio instead of displaying forms on the screen) and different input types (for example, using a microphone as an input device instead of writing – necessary in certain situations such as if the subject is driving) [50]. It is the flexibility of the ESM that has led it to be used for many applications other than the one for which it was designed (measuring flow in everyday life), for example investigating time use [60] and evaluating ubiquitous computing applications [50].

In this thesis, we focus not on flow in everyday life, but on flow in particular activities. This means sampling is desired only when a subject is engaged in the activity in question. The ESM’s flexibility allows it to be adapted for this situation in different ways, depending on the activity. If a tool is used in the activity, it could be possible to augment the tool so that it can sample experience. Chen adopted such an approach by augmenting a browser to study web activities [45]. With some activities, it is impractical and possibly hazardous to signal subjects randomly and expect the subject to fill out a form. An example of such an activity is white-water kayaking. But, even here, possibilities exist: a group of researchers managed to study flow in white-water kayaking by anchoring subjects at different points along a river; these points which were deliberately chosen to incorporate a variety of levels of challenges [108]. More recently, investigations have begun into Context-aware Experience Sampling (CAES) [103], in which sensors can identify when a subject is engaged in the activity of interest and restrict its sampling to those times.

A limitation of the ESM arises from the technique used to measure the balance between challenges and skills. When filling in the ESF, subjects are asked to assign to “the challenges of the activity” and to “your skills in the activity” a number between 0 (none) and 9 (high). However, both challenges and skills are complex variables and
rating them on a one dimensional scale may not supply reliable measurements [77]. In particular, the method of measuring challenges and skills gives rise to ambiguity [77].

What specific challenges and skills are measured? Subjects must rate “the challenges of the activity”. But almost every activity has a set of challenges, so which of them must be rated? The only challenges relevant to whether the subject is currently in flow are those that the subject is dealing with at that moment. That is, the level of challenge of the activity will be the level of challenge of the task at hand. For example, if a level 10 climber chooses the task of doing a certain level 7 climb, the level of challenge of the activity that he experiences will be low. However, if he adds additional challenges to the task at hand – for instance, doing the climb without equipment, or using only one hand, the level of challenge will be greater [54].

A subjects must also rate “[his] skills in the activity” [62]. Which skills in the activity must be rated? If all the skills of the activity are measured, then the set of skills that is not relevant for the task at hand could affect the overall rating of skills to the extent that the challenges and skills may match where they should mismatch or vice versa. Hence, only the skills relevant to the task at hand – a subset of the set of skills of the activity – should be considered. Therefore, in order for an accurate indication of flow, it must be ensured that only the challenges of the task at hand and the skills relevant to those challenges are measured in determining the challenges/skills ratio.

A second ambiguity in the method of measuring skills and challenges is: how exactly is a match between challenges and skills defined? Initially, a match was defined when the values of challenges and skills (both numbers from zero to nine) were equal [62]. However, after the adjustment was made to the model for flow in everyday life (challenges and skills had to be in balance and also had to be above a certain level – usually set as the person’s weekly average), the usual practice is to calculate z scores for the set of challenge scores and for the set of skill scores. Because the means can be
quite different (for example, in Massimini and Carli’s study [138], the mean skill level was 5.8 and the mean challenge level was 3.8), the matches of the raw scores can be quite different to the matches of the z scores [77]. That is, depending on the method used, the results can differ greatly – one method may claim that the person was in flow, while the other may claim that the person was not.

Another limitation of the ESM is the burden it places on subjects. Subjects are repeatedly interrupted while they are engaged in activities and asked to fill out a form that takes between 1 and 2 minutes to complete, which can become annoying. Some research has been done to alleviate this problem [100, 86]. Sensors and machine learning have been used to find times when the subject is likely to be most receptive to interruption. For example, [100] showed that people are more receptive to interruption at activity transitions than during activities. But in studying flow, it is descriptions of the subjects’ states as they are engaged in activities which are of interest, so this solution is of little help.

Sensors

Each of the methods of measuring flow described so far require explicit input by a user. However, it may be possible to measure flow by combining the input of a collection of sensors. Picard suggested using electroencephalogram (EEG), blood flow and “other factors known to correlate highly with cognitive load and with deep involvement” [174]. Using this method, it would be possible to determine if a user is in flow without interrupting him to ask him if he is in flow – a key benefit since an interruption could take him out of the state.

A piece of research was carried out in Northwestern University in which subjects played a videogame designed to induce flow, and to “examine neural activity associated with high performance on a flow-inducing task” [124]. Continuous EEG readings were taken as the subjects played the game. The EEG measures electrical activity in the
brain – brainwaves of different frequencies, which are subsequently transformed to five bands (Alpha to Gamma). The results indicated that greater Alpha (8-12Hz) frequencies in the left temporal lobe compared with the right temporal lobe predicted improved performance. Kramer mentions that flow is also a predictor of improved performance, but that further studies would be necessary to reveal the relationship between flow and this neural activity.

The greatest limitation of this work is that the degree to which each subject experienced flow while playing the videogame was not also measured using a subjective measuring technique. For a stronger correlation between flow and certain neural activity, it is necessary to have a more reliable indication that a person is in flow, rather than simply assuming this task will definitely induce flow. A current project at the University of Zurich is also investigating the measurement of flow with EEG [2]. Once the neural activity associated with flow is determined, it will be possible to objectively measure whether a person is in flow. Furthermore, since wireless EEG is available and reported to be comfortable [110], it will be possible to determine, regardless of the activity, whether a user is in flow without having to interrupt him.

Another possibility is not to measure flow directly, but instead to measure certain non-optimal states, and intervene if the user is determined to be in one of these states. For example, Burleson determines whether a user is in a non-optimal state he calls stuck, which occurs when a person is engaged in the task and perceives that he does not possess sufficient skills to succeed [12]. The stuck state is determined by measuring such elements as: a user’s posture, arousal level (using skin conductance), his head movements, mouth fidgets, smiles, blinks, and pupil dilations. The system developed by Burleson and his colleagues was reported to have 79% accuracy in determining if a user will give up at a given time [111], which is a good indication of the stuck state. Although much work remains, this seems a promising path indeed.
2.1.5 Limitations of the Flow Model

Two limitations of the flow model need to be mentioned. Firstly, the three key conditions of flow described in Section 2.1.2 are necessary for flow; they are not, however, always sufficient for flow [152]. Research indicates other factors can influence the extent to which an individual becomes engaged in an activity [57]. These include: the importance an individual places on succeeding in an activity; the congruence between “task-specific, behaviourally based goals” and “higher-level, more abstract goals” (for example, compare the goal of putting a flag on one’s antenna with the goal showing one’s patriotism); and absorption – a measure of hypnotic suggestibility [57]. Another possibility is what Csikszentmihalyi called the autotelic personality, which is characterised by a number of metaskills, including general curiosity, persistence, and low self-centredness [57].

Secondly, researchers have developed a number of ways of defining when a person is experiencing flow [152]. Situational measures have been used, for example, ratio of challenge and skill. State measures – for the most part, composite variables – have also been used; for example, Ghani measured the combination of enjoyment and concentration [92] while Trevino and Webster measured the combination of enjoyment, attention focus, and control [197]. The results of studies using different definitions may not be comparable. Therefore, to further knowledge about flow, it may be beneficial to assess each definition and agree to use one of them [152].

2.2 Flow in Computer Mediated Environments

This section reviews systems that influence the conditions of flow, while the user is engaged in an activity in a computer mediated environment.
2.2.1 Hyperlead

Hyperlead is a hypermedia learning environment designed to teach management by objectives to managers [121]. A study was conducted to determine, among other things, the extent to which people experience flow when learning with a hypermedia system [123]. The Experience Sampling Form (ESF), a self-report questionnaire, was used to assess subjects’ psychological states. The experiment, which took approximately 120 minutes to complete, involved assessments of the subject’s state, a 45 minute interaction with the hypermedia tool, three tests of the subject’s acquisition of knowledge from the hypermedia interaction and three further assessments of the subject’s state, 15, 30 and 45 minutes after the hypermedia interaction.

Hyperlead does influence the conditions of flow – 33.3% of the 60 people who took part in the experiment experienced flow. Tasks are supplied to the subjects, but the tasks same regardless of the subject. This means that only subjects whose perceived skills happen to match the challenges of the supplied tasks stand a chance of experiencing flow. Hyperlead provides feedback by means of a set of 20 questions at the end to let the subject know how they did, and in Hyperlead 3.0, a subsequent version of the software [122], by supplying tests of knowledge after each training unit. Hyperlead 3.0 also provides feedback with a short film sequence followed by questions to test if the subject could transfer their knowledge to a new situation. Feedback in Hyperlead could be improved if it was supplied while the user is doing the activity, not just at the end of the activity.

Hyperlead uses the ESF to measure flow, and while this does give “high-resolution description[s]” of the subject’s mental states as they occur [62], it has the drawback of being time consuming for the subjects (each measurement takes about two minutes), and this reduces the number of measurements which it is reasonable to make.
2.2.2 IT-Emperor

IT-Emperor is an educational game that aims to facilitate flow experience [115]. In the game, players are required to supply learning materials about usability. Players can either create the materials themselves or, if they have accumulated sufficient credits (by selling some materials they have produced), they may buy some of the materials from the marketplace. The game takes about 30 hours to complete and was designed using the experimental gaming model, a generalised description of educational game design devised by the author of IT-Emperor. In this model, challenges linked by a storyline are supplied to the player, and learning occurs through action in the game world [116]. A design aim endorsed by the model is to maintain the motivation of the player by supplying him/her with appropriate challenges.

In IT-Emperor, a player does tasks such as buying, making, or assessing pieces of material. However, it does not take the skills of the players into account, and players reported that the tasks were not challenging enough. IT-Emperor supplies some feedback to a player in four ways: other players assess his material; corporations assess his material; a player can see how successful he is in the marketplace; and the boss of the company provides evaluation reports. The players considered the amount of feedback adequate, although only 11/18 players reported that they did not have to wait too long for feedback.

A study was done of 18 students who completed the game over the course of 2 months [115]. Once a student had completed the game, he filled out an online questionnaire. This included measuring flow, and reported that only 8 of the 15 players who completed the questionnaire experienced flow while playing IT-Emperor. A means of improving this was suggested: a proposed adaptive game engine that aims to maintain flow by observing the challenge/skill balance and manipulating the challenge level [116], although no details were given about how this might be done.
2.2.3 Super Tangrams

In order to demonstrate the suitability of a model that aims to produce the characteristics of flow in mathematics learnware\(^1\) a specific system, called Super Tangrams, was built \([187]\). Super Tangrams (ST) is mathematics learnware designed to assist children in learning 2D transformation geometry, and as it does so, to facilitate the flow experience. Tangrams is a Chinese puzzle in which a person is given 7 pieces (each a simple 2D shape) and a picture of a more complex 2D shape. He must fit the 7 shapes together (without overlapping them) to produce the shape shown in the picture. Users of ST must solve, one at a time, a sequence of tangrams displayed to them. Part of the screen displays the puzzle, and part of it contains an interface that can be used to transform the 7 shapes (for instance, by rotating a particular shape by a certain angle, or by reflecting a shape through a line).

ST influences each of the three key conditions of flow. It recommends tasks – tangrams – and a tangram has a clear end goal. Moreover, it is possible for a person to set intermediary goals himself, such as to complete a certain section of a puzzle. ST contains 40 tangrams, divided into three levels. Solving the tangrams becomes progressively more difficult (that is, it requires greater mathematical knowledge) as a person moves through the sequence. At any stage, a user may access an instructional module that provides three kinds of items: explanations of concepts, like those found in mathematics textbooks; a hands-on guide to teach how the user interface works; and a set of strategies for solving tangrams.

In this way, ST aims to maintain a balance between challenge and skill. A drawback of this approach is that one sequence of tasks that every user does sequentially cannot ensure that a balance between challenges and skills is maintained. Each user’s skills could be different upon facing any task, and if the next task is either too easy or too difficult, no remedial action can be taken, the user must simply do the task and his

\(^{1}\)Software used for the purposes of learning. \([187]\)
experience will suffer.

ST supplies different kinds of feedback. One kind of feedback that is supplied is sounds accompanying certain events, such as if a transformation of a shape puts the shape out of bounds. A second kind of feedback given is a score which is based on the number of moves a user has made so far, compared with the minimum number of moves needed to solve the tangram.

ST measures flow using interviews, and [187] provides a transcript of one interview as evidence that flow was experienced using ST, and the content of the interview certainly suggests that he/she experienced flow using ST. However, this method of measuring flow has its drawbacks: it is limited by a person’s recall (see Section 2.1.4) and, in addition, it does not capture at specific moments during the activity if a person is in flow, or if the three key conditions of flow are present. This information could be used to improve the game. For example, if flow was measured after each task, patterns might emerge suggesting changes to the sequence of tasks (tangrams) to increase the amount of flow experienced.

2.2.4 Pearce et al.

Pearce et al. describe an attempt to make e-learning activities more engaging [169, 170, 168]. A small system was built to teach students about physics, specifically, the concepts of velocity and acceleration. Students did seven separate tasks, each involving viewing an animation of a moving cart, and then sketching graphs on paper depicting the velocity and acceleration of the cart. Flow was measured between each task and again at the end of the session.

The sequence of tasks that is supplied is the same for each student – that is, the student’s current skills are not taken into account. Feedback consists of velocity and acceleration graphs drawn as the cart in the animation moves. This is interpreted more effectively by those with better domain knowledge, which demonstrates a limitation of
this feedback: students without sufficient domain knowledge did not have a good idea of how well they were doing with the task.

Pearce et al. speculated that students would not remain in the same state throughout the session. For this reason, they decided to measure flow in two different ways. For the first method, the student’s perception of challenge and perception of skill were measured between each of the seven tasks, as an indicator of flow. These were measured in the standard way, that is, using a Likert scale; for example, “how challenging did you find this last activity?” had five possible responses from too low to too high. These responses, valued from 1 to 5, indicate flow if the values for challenge and for skill are equal. It was noted in Section 2.1.4 that a modification was made to the flow model that required not only that the values for challenge and skill to be in balance but also that each must be above its weekly average. However, for a single activity these weekly averages are not available (since flow is measured only during the session) and no suitable substitute for them was available either.

The second method for measuring flow was to use an 11-item survey based on control, engagement, and enjoyment. Analysis suggested using only two of these factors (enjoyment and control), and scores for these were summed to produce an overall value for flow. In order to compare the results of the two methods, a quantity called from-flow-distance was devised, which ranges from -1 (maximum anxiety) to 1 (maximum boredom) and where flow is represented by 0. The quantity is calculated by:

\[ \text{from-flow-distance} = 0.25 \times (\text{skill} - \text{challenge}) \]

The two measures of flow did not agree. Some students demonstrated a high level of flow in the survey, but flow was not evident in the challenge/skill ratios. Others gave a low overall value for flow, but a number of their challenge/skills ratios were equal to 1. One reason for this disagreement is that the overall flow value was measured with two variables – control and enjoyment, and while the variables of engagement, enjoyment and concentration have been shown to be a good proxy for the much more
complex state of flow [152], the same cannot be said of the two chosen here. The primary reason seems to be that, as discussed in Section 2.1.4, using such surveys, it is unclear exactly what has been measured and therefore it is not that surprising that the measures did not coincide. It is possible that if both measures were taken between each task, instead of only the first measure being taken between each task, they may have been in agreement.

However, following interviews with eight of the students, Pearce et al. came up with a different explanation. They claimed that sometimes the challenge came from the artefact (in this case the software) and sometimes it came from the task (in this case the task is solving the physics problem). Further, they claimed that since existing techniques for measuring flow do not take this into account, they are inadequate [168]. This is not the case. Activity Theory holds that artefacts mediate human behaviour, that is, people must engage with artefacts to do tasks [153]. In addition, artefacts can only provide a challenge if they are used to do something – the artefacts by themselves cannot provide a challenge. Therefore, it does not make sense to separate task and artefact and to measure challenge and skills separately for each.

A limitation with the approach to measuring flow taken mentioned by Pearce et al. is that there is no guarantee that the challenges and skills are measured by the same standards, and so the challenge skills ratio may not always be accurate. In addition, only a student’s perception of challenges and skills for a task, and not the other two of the three key conditions of flow are not take into account. In this way, an opportunity to make changes to the tasks is lost; these changes could make more it likely for subsequent students to experience flow with these tasks, such as improving tasks with unclear goal or improving the means of providing feedback if it was identified as a problem.
2.2.5 SingStar

SingStar [5] is a game for the Sony PlayStation, for the activity of singing. Players select one of the available songs, sing it into a microphone, and get a score reflecting how well they sang it. SingStar provides clear goals by graphing the frequency of the notes the user should be singing. It also graphs, in a different colour, the frequency of the notes the user sings as she sings them, giving her visual feedback letting her know whether she hit the note, or if she did not, she can see if the note she sang was too high or too low, and by how much.

The user’s moment to moment accuracy is summed to give a score; a perfect rendition gives a score of 10,000. Singstar has three difficulty levels; the higher the difficulty level, the more accurate the user needs to be to achieve a high score. This provides a simple means of balancing challenges and skills. This could be improved on by creating a detailed user model/profile that would enable, for example, well matched challenges to be recommended, for example, to sing a particular song and achieve a certain score on it.

2.2.6 Burleson

Burleson describes the design and evaluation of an intelligent tutoring system that facilitates learners’ development of affect skills for dealing with the negative feelings connected with failure [42]. He claims that by adapting a task to a user so that the opportunity for flow is increased, users miss out on the chance to learn these affect skills. For example, instead of giving up, learners could learn to use these negative feelings as a signal to try a different strategy. The system can measure learners’ affect using a set of sensors: a pressure mouse (shown to correlate to frustration), a skin conductance sensor (which indicates arousal), a chair that measures posture, and a facial expression camera (which can measure head movements, mouth fidgets, smiles, blinks,
and pupil dilations). Although it could be used to measure many different states, this work focused on measuring one: stuck, a state Burleson defined which occurs when a person is engaged in the task and perceives that he does not possess sufficient skills to succeed.

One activity was used in the intelligent tutoring system: the well-known Towers of Hanoi problem. It was chosen because of the extensive research done on the problem, and because it is an activity that can potentially be challenging and/or frustrating. It is also possible to determine at any moment how far the user is from the solution (the minimum number of moves can be calculated). The user interacted with the activity and also with the affective agent using a keyboard and mouse (an affective agent is depicted as an animated character; it can sense a user’s affect and respond by displaying its own affect).

The system supplies a user with a task that has a clear goal, and the system’s task support provides a way to balance challenges and skills – if the user is stuck, she is shown a screen with some hints. This approach is limited since each user gets the same task, that is a user’s current skills are not taken into account, and when a user is stuck it is always the same screen that is shown to a user, regardless of the state of the game. However, this limitation is unsurprising since affect support and not task support is the focus of the work. As for feedback, the user knows the end state, and has a visual representation of her current state, letting her know how well she is doing. The minimum number of moves required to complete the problem could also be displayed, although it isn’t in the current version.

An experiment was carried out with 61 children aged 11-13 who used the system. The purpose of the experiment was to compare the effects of “affective support” with “task support”, taking the user’s state into account. Task support is used in many intelligent tutoring systems, where the system monitors the user’s progress in a task and supplies help when it deems the user needs it. Affect support is dialogue between
an affective agent and a user to help the user learn skills and strategies to help him persevere with tasks. The procedure of the experiment involved some pre-activity tasks, followed by engaging with the activity (the Towers of Hanoi) for four minutes, followed by either “affective support” with “task support” depending on the user’s state, and finally, some post-activity tasks.

The experiment produced many interesting results not especially relevant to this thesis. What was of particular interest to this thesis was that flow was measured. This was done using a post activity self report that used 7 point Likert scales to measure control, the ability to concentrate, and whether the user thought she had the necessary skills to do the task. The same scale was used to measure stuck, where higher levels of flow indicate lower levels of stuck. The distribution of flow scores, that is, estimations from the scale of the degree of flow experienced by the user in the activity was not reported, probably because it was not the focus of the experiment. Much more interesting for this thesis, however, is the potential this system has to calculate flow scores in an entirely different way.

The classifier of the user’s state can “predict individual learner’s affective state (Stuck and Not-Stuck)”, that is, it can detect at any given moment if a user is likely to keep engaging in a task, or instead if the user is frustrated or likely to quit (seek outside help) with 79% accuracy (chance = 58%) [111]. The collected data enables a distribution of the learner’s state (stuck or not-stuck) for the activity to be constructed. From this, a distribution of learner’s state (in flow or not in flow) could potentially be constructed. This might involve accurately classifying many other states (at least the seven other states identified in [152]), since if the user is not in stuck, he is not necessarily in flow – he could be in any of the other states.

However, it seems possible that the system could learn to accurately classify the other states, and thus a distribution of learner’s flow state in an activity could be constructed. This would be invaluable for flow research: having such a degree of granularity
would allow researchers to try different strategies when a person is engaged in a task and to see what the effect was on flow. It would enable much more precision, as the reaction to a strategy would be recorded immediately rather than measuring after the task, when it is likely that other factors could have affected the flow measurement. The key benefits for flow applications is that it would enable them to respond in real-time to the user’s state without having to ask the user – if he is in flow, such an interruption could take him out of the state.

2.3 Conclusion

The first part of this chapter described the flow model and methods of measuring flow. The flow model divides the elements of flow into two sets: the characteristics of flow and the conditions of flow. The characteristics of flow are the experiential elements of it, while the conditions of flow describe the conditions required for flow to occur. In order for someone to experience flow on an ongoing basis, it is necessary for him to continually learn.

Four main methods have been used to measure flow: interviews, surveys, the Experience Sampling Method (ESM), and sensors. The foremost limitation of using interviews is that when people describe an experience they have had, they are liable to forget details or inadvertently distort what occurred. Surveys share this limitation and have other limitations, such as the infeasibility of asking anything other than closed questions. Another limitation is that the acquired information cannot be used to alter the user’s current experience.

The ESM was designed to overcome the limitations of interviews and surveys. This method allows high resolution descriptions of people’s mental states to be captured as they occur, and measures flow as the ratio of challenges and skills. The ESM was originally designed to measure flow in everyday life. This thesis focuses not on flow in
everyday life, but on flow in particular activities. However, the flexibility of the ESM enables it to be adapted to studying flow in a particular activity. Two limitations of the ESM are that its method of measuring challenges and skills gives rise to ambiguity and that it places a cognitive burden on its users. Some work has been done on measuring flow using sensors to measure indicators such as skin conductance, posture, head movements, blinks, and pupil dilation. This research area is in its early stages but seems most promising.

Two limitations of the flow model need to be mentioned. Firstly, the three key conditions of flow described in the chapter are not always sufficient for flow; research indicates other factors can be influential. Secondly, researchers have developed a number of ways of defining when a person is experiencing flow, and these may not be comparable.

The second part of this chapter reviewed systems that aim to influence the conditions of flow, while the user is engaged in an activity in a computer mediated environment. Each of the systems reviewed provides the user with tasks, but only two systems attempt to balance challenges and skills, and they do so in limited ways. SingStar provides three levels; the tasks are the same on each level, but scoring points is more difficult on higher levels. This is a simple means of balancing challenges and skills – users are required to choose tasks and the level themselves; it could be improved on by creating a detailed user model/profile that would enable, for example, well matched challenges to be recommended.

Super Tangrams aims to sequence its tasks so that they increase in difficulty, and while this certainly has the potential to balance challenges and skills, it is limited because since people learn at different speeds, each user’s skills could be different upon facing any of the tasks. It is therefore likely that on many occasions, the next task will be either too easy or too difficult, and when this occurs, no remedial action can be taken, the user must simply do the task and his experience is likely to suffer as a
result. In terms of producing the conditions of flow, task recommendation is of great importance, in particular balancing skills and challenges, but it receives little attention in any of the reviewed systems.

The feedback supplied by the systems is of varying degrees of quality, which can be measured by how well it lets the user know how well he’s doing with the task at hand. Different activities provide different opportunities for feedback. The feedback in Hyperlead is the number of questions a user gets correct in a post-test; in IT-Emperor, feedback is mostly from assessments of the content the user has created; in Super Tangrams, it is a score based on the number of moves the user has made so far; in Pearce, it is through graphs that can be compared with the user’s graphs; in SingStar, the user sees two different coloured graphs showing the correct notes and the notes she sang; in Burleson, the user knows the end state, and has a visual representation of her current state, letting her know how well she is doing.

SingStar is a good example of how context (in this case the frequency of the note the user is singing) can be used to enhance the feedback a user gets – for most people, this is much more detailed than they would otherwise get (by hearing the notes as they sing and comparing them with their mental representation of the song). There are other examples of this in the computer games domain, such as Wii Fit [157], which calculates the location of a user’s centre of gravity and uses this to give users visual feedback and a score. There are also examples for other activites, such as running: the Nike + iPod, which uses a small accelerometer in a shoe to provide a user with visual and audio feedback about goals relating to time, distance, and calories burned [104].

The measurement of flow in all the systems except Pearce is done via using a survey or interview after doing the activity. The main limitations are that people are liable to forget details or inadvertently distort what occurred, and that the information acquired by these methods cannot be used to alter the user’s current experience. Burleson also demonstrated a method that could potentially measure flow automatically. In addition
to a survey after the activity, Pearce measured flow by measuring the user’s level of skills and of challenge after each task. This method is limited in two ways; first, there is no guarantee that challenges and skills are rated by the user using the same standards, and second, because it measures just one of the three key conditions of flow.

If instead, all three key conditions of flow are measured, it would be possible to suggest action to take if one or more of the conditions were not present. Also, this could be used to improve the systems over time, so that subsequent users would be more likely to go into flow – something which none of the systems incorporate at present. Examples of this are to improve tasks that are identified as having unclear goals or improving the means of providing feedback if it was identified as a problem.
Chapter 3

Literature Review on Task Recommendation

In the previous chapter, it was observed that in terms of producing the conditions of flow, task recommendation is of great importance, but received little attention in any of the reviewed systems. This chapter focuses on the area of task recommendation. It examines approaches to task recommendation, reviews some representative systems, examines approaches to improving recommendations, and identifies limitations of existing approaches in recommending tasks to produce the key conditions of flow.

3.1 Approaches to Task Recommendation

Recommender systems is an area of high interest because it contains a wealth of problems and has an abundance of practical applications [8]. Some examples of these practical applications are: Amazon.com [134], which recommends books, CDs, and other items, MovieLens [136], which recommends movies, and ISIS [70], which recommends learning activities. Recommender systems focus on the recommendation problem, most commonly formulated as the estimation of ratings for items that have not been seen
The recommendation problem may be expressed more formally as follows. Let $U$ be the set of all users and let $I$ be the set of items that can be recommended. Let $r : U \times I \rightarrow R$ be a utility function that measures the usefulness of a given item to a given user, where $R$ is a totally ordered set representing the possible ratings an item can have (for example, integers from 1 to 5). Then, given a user $u \in U$, the required solution is those items of the set $I$, which have the highest utility, that is, the set \( \{ i \in I | r(u, i) \geq r_0 \} \), for some threshold value $r_0 \in R$. The core problem of recommender systems is that $r$ is usually only defined on a subset of $U \times I$, so it must be extrapolated to the whole set $U \times I$.

Recommender systems are usually classified in two ways. The first is by the recommendation technique used; recommendation techniques fall into two categories: model-based and memory-based. In model-based methods, the set of ratings users have given items is used to construct a model that predicts ratings; examples of models include Bayesian networks, neural networks, and Markov processes. There is empirical evidence that model-based methods outperform memory-based methods.

In memory based methods, a heuristic is used to predict ratings using the complete set of ratings users have supplied. Unlike model based methods, in which the model must be rebuilt to incorporate new data, memory based techniques “continuously analyze” all available user and item data. An example of a heuristic is to estimate the rating of an item $i$ for a user $u$ by aggregating the ratings given to the item by the users most similar to $u$.

The second way recommender systems are usually classified is by recommender approach; recommendation approaches fall into three categories: content-based, collaborative, and hybrid. In content-based approaches, items that score highest against a user profile are recommended. Collaborative approaches recommend items to a user based on whether the items were liked by similar users. Hybrid approaches mix content-
based and collaborative approaches.

This section describes each of these three approaches (content-based, collaborative, and hybrid) in detail for recommending items in general, after which the question of what is different about recommending tasks is considered.

### 3.1.1 Content-based Methods

Content-based recommender systems have a means for describing items (an item profile), a means for describing users (a user profile/model), and a means for comparing item profiles and user profiles that enables items to be recommended [166]. Examples of content-based recommenders include LIBRA [150], which recommends books, Newsweeder [129], which recommends news articles, and SMMART, which recommends products based on users’ shopping habits and current location (such as beside a certain store) [125].

An item profile consists of a set of attributes that characterise the item, and is usually constructed by extracting a set of features from the item [8]. For example, a movie could be characterised by such attributes as the cast, director, subject matter, genre, and running time. A user profile/model may contain several different types of information, such as a user’s preferences and a user’s interactions with a recommender system [166]. For example, in a movie recommender system, a user profile could contain information such as particular actors or directors the user likes. User profiles can be constructed from information obtained explicitly (for example, through questionnaires) or implicitly (for example, by inferring it from the items the user has rated) [8].

Once the item profiles and user profiles have been constructed, it remains to compare item profiles with a user profile to determine the best matching items. One commonly used method assumes that the user profile comprises a vector $\mathbf{w}_u$, each element of which is the weight of an attribute (representing the degree of the user’s preference of that attribute). The item profile is represented in a similar way, except here the weights
represent the degree to which the attribute is present in the item; for example, in the Fab system [13], the items recommended are web pages, and each item is represented by the 100 most important words on the page. A utility function, usually defined by some heuristic, can then be used to measure how similar the vectors are. A popular method is the cosine measure, defined on page 48.

Besides such traditional heuristics, other techniques have also been used for comparing user profiles and item profiles, including Bayesian classifiers, decision trees, and neural networks [8]. These methods learn a function that takes the user profile and item profile as parameters and predicts the user’s interest in the item [166]. Such functions either produce a numeric value representing degree of interest, or the probability that the user will like the item; in both cases a sorted list of recommended items may be produced [166].

Limitations

Content-based recommender systems have a number of limitations:

- Limited content analysis
  Because content-based techniques rely on the features of the items it recommends, it must be able to extract features from the items – either automatically or manually [8]. If the items are composed of text, the features may be extracted using information retrieval techniques. It is much more difficult to automatically extract features from non-text items, such as multimedia data or physical items [189]. If automatic extraction isn’t possible, then the features must be extracted manually. However, this is often impractical due to limitations of resources [189].

- Overspecialisation
  In content-based recommenders, a user profile is usually constructed from items the user has already rated. As a result, only items similar to those the user
has already rated can be recommended [14]. This restriction is often alleviated by introducing an element of randomness, for example, with the crossover and mutation operations of a genetic algorithm [14]. Another problem caused by over-specialisation is that recommended items might be too similar to items a user has already seen [8]. Such items can be filtered out, as in the news article recommender system, the DailyLearner [23].

- The New User Problem

A user profile/model is usually constructed using items the user has rated. Since a new user will have rated very few items, his user profile will be inadequate and will not provide accurate recommendations [8].

### 3.1.2 Collaborative Methods

Collaborative recommender systems recommend items that similar users liked. Typically, a set of the most similar users (nearest neighbours) is calculated for each user [14]. Given a user, similar users are identified by calculating the correlation between a given user’s ratings and a candidate user’s ratings [14]; usually the N most similar users are taken into account [8]. A recommendation score for an item a user hasn’t seen is then predicted using a combination of the ratings the nearest neighbours gave that item [14]. Examples of collaborative recommender systems include: Amazon.com [134], which recommends books and other items, MovieLens [146], which recommends movies, and Jester [93], which recommends jokes.

More formally, $r(u, i)$ is the rating a user $u$ gave an item $i$. If this is unknown, it can be estimated using $r(u', i)$, for each user $u'$ of the $N$ most similar users who has rated the item $i$. Of the many ways of combining the similar users’ ratings, the most common approach is to calculate the weighted sum [8]:

$$r(u, i) = k \sum_{u' \in N(u)} sim(u, u') r(u', i)$$
where \( N(u) \) is the set of \( N \) users most similar to \( u \), \( k \) is a normalising factor, usually set to
\[
\frac{1}{\sum_{u' \in N(u)} |\text{sim}(u, u')|}
\]
and \( \text{sim}(u, u') \) is a measure of similarity between the users \( u \) and \( u' \).

Essentially, the measure of similarity between two users, \( \text{sim}(u, u') \), is a distance measure, and determines the weight a particular user’s rating should have; that is, the more similar the user, the greater the weight \[8\]. It can be measured in a variety of ways, most of which are based on \( I_{uu'} \), the set of items that both \( u \) and \( u' \) have rated; the most popular methods are correlation and the cosine measure \[8\]. The correlation approach uses the Pearson correlation coefficient to calculate similarity \[189\]:
\[
\text{sim}(u, u') = \frac{\sum_{i \in I_{uu'}} (r(u, i) - \bar{r}_u)(r(u', i) - \bar{r}_{u'})}{\sqrt{\sum_{i \in I_{uu'}} (r(u, i) - \bar{r}_u)^2 \sum_{i \in I_{uu'}} (r(u', i) - \bar{r}_{u'})^2}}
\]
The cosine measure is calculated by constructing a vector of dimension \( |I_{uu'}| \) for each user containing that user’s ratings for each item in the set, and taking the cosine of the angle between the two vectors\[9\]:
\[
\text{sim}(u, u') = \frac{\vec{u} \cdot \vec{u}'}{||\vec{u}|| ||\vec{u}'||}
\]
Many performance enhancing modifications have been suggested. Estimating ratings, in particular calculating a user’s nearest neighbours is expensive, but almost immediate recommendations are required; for example, Amazon.com probably has a time constraint of a small fraction of a second \[185\]. A common approach to improving efficiency is to calculate \( \text{sim}(u, u') \) for all users in advance, and to repeat this calculation periodically; this works well because in a short time, the nearest neighbours do not change dramatically \[8\]. Other ways of reducing processing time and memory
consumption include subsampling, in which a subset of users to compute the nearest neighbours, and clustering, in which a user is compared not to individual users but to a cluster of similar users [185].

Another performance enhancing modification arose from the observation that different users attach different meanings to the same rating; for example, one user’s rating of 3 may be equivalent to another user’s rating of 4 [185]. This limitation of the weighted sum can be addressed by using deviations from the user’s average rating in place of raw ratings [8]:

\[
r(u,i) = \bar{r}_u + k \sum_{u' \in N(u)} sim(u, u') (r(u', i) - \bar{r}_u)
\]

Other performance enhancing modifications include default voting, inverse user frequency, case amplification, and weighted majority prediction [8]. In default voting, for example, it has been demonstrated empirically that if missing ratings are given a default rating value, accuracy of rating prediction improves [26].

In contrast to the memory-based methods just described, model-based methods construct a model from the collected ratings, and use this to predict unknown ratings. For instance, a probabilistic approach could calculate ratings in the following way [26]:

\[
r(u,i) = E(r(u,i)) = \sum_{j=0}^{n} j P(r(u,i) = i | r(u,i'), i' \in I_u)
\]

That is, the unknown rating is estimated as the expected value of the rating, and the probabilities can be estimated using a cluster model or a Bayesian network [26]. Other model-based collaborative approaches proposed include: linear regression, a maximum entropy model, and Markov processes [8].

Limitations

Content-based recommender systems have a number of limitations:

- The New User Problem

As with content-based approaches, in order for recommendations to be accurate,
the recommender system must acquire information about a user from the ratings he has given; a new user is unlikely to have rated a sufficiently many items \[8\]. Of the techniques proposed to address this limitation, most use a hybrid approach \[8\], described in the next section.

- **The New Item Problem**

  New items are frequently added to recommender systems. For a collaborative recommender system to recommend an item, it requires a considerable number of users to have rated the item before it can be recommended \[8\]. This shortcoming can also be addressed using a hybrid approach \[8\], detailed in the next section.

- **Sparsity**

  In most recommender systems, the number of items that are rated is usually greatly exceeded by the number of items that are not \[8\]. This sparsity leads to problems. For example, a large number of movies in a movie recommended system might be rated by just a few users. As a result, these movies will seldomly be recommended \[8\]. Also, there will be few users similar to users with unusual taste, and consequently users with unusual taste will receive poor recommendations \[14\].

  One way of dealing with this shortcoming is to use user profile information, such as gender, age, education, etc. to calculate the similarity between users \[167\].

### 3.1.3 Hybrid Methods

Hybrid recommender systems produce their output by combining content-based and collaborative approaches, helping them to avoid some of the limitations resulting from using one of the approaches on its own \[8\]. There are numerous ways to do this, some of the key methods are described below.
Weighted

A weighted hybrid recommender computes the rating of an item by combining the ratings of multiple recommendation techniques. One way to do this is to use a linear combination of the ratings, as in the P-Tango system [48], which has a collaborative component and a content-based component each initially assigned an equal weight. As users rate items, the relative success of each component is calculated and the weights are adjusted accordingly. A limitation of this approach is that it assumes that the value of the different recommendations strategies is roughly uniform on the set of items [40]. However, recommendations strategies can have different strengths for different kinds of items. For example, the performance of a collaborative strategy will be worse for items with a small number of ratings [40]. A possible means of overcoming this is to use a switching approach (described next) to switch the recommender strategy depending on the context.

Switching

A switching hybrid recommender selects one strategy from the recommendation strategies available in a system, depending on the current context. To make this decision, a reliable criterion must be available. A simple example of a criterion is a measure of agreement between user ratings and the recommendations produced by a particular recommendation already strategy; this approach was used in [196], which is an e-commerce recommender. Another example of a criterion that has been used is a confidence value that each recommender produces; this approach was used in the DailyLearner system [23], which recommends news stories. This means that, for example, in cases where a new item is being recommended, a content based approach would be used, since unlike collaborative approaches, content based approaches do not suffer from the new item problem. However, determining a confidence value for recommendation is not straightforward, and the question remains an area of active research [41].
Feature combination and augmentation

There are several ways in which hybrid systems can combine or augment features from multiple recommendation approaches. Collaborative data (user ratings) can be used as an additional feature for each item, creating a set of augmented items, over which content-based techniques can be used, as described by Basu et al. in [20]. In this way, the system does not rely exclusively on collaborative data, and consequently the effect of the new user problem is reduced. Essentially, the algorithm treats facts like user 1 and user 2 liked movie X in the same way as actor1 and actor2 were in movie X [41]. While this did significantly improve precision (over a pure collaborative approach), it was only achieved by hand filtering content features; when all the content features were used there was no improvement in precision [40]. Another approach is to create filter-bots [94] – content analysis agents that are treated as additional users in a collaborative approach. A similar approach, described in [142], is to take the usual users’ rating vectors of a collaborative algorithm and to augment them with additional ratings from a content-based method.

Meta-level

A meta-level hybrid recommender takes a model learned from one recommender and uses it as input for another recommender [41]. This differs from the feature augmentation approach because it generates the entire input for the next recommender, not just part of it. An example of a meta-level hybrid is given by Pazzani, who described a restaurant recommender that built user models using a content-based method, and used these models as input into a collaborative recommender, which used these user models to identify peers [167]. Pazzani mentions a benefit of the approach: that it allows some sparsity related problems to be overcome – pure collaborative approaches typically do not have many pairs of users with a significant number of commonly rated items [167]. A similar example is Fab [14], whose authors mention another benefit: that
items can be recommended either from similar users or directly (by comparing an item with a user’s profile). However, it is not always straightforward or feasible to construct a metalevel hybrid for any two recommenders, since the first recommender needs to produce a model that the second recommender can take as input [41].

Knowledge-based Techniques

Knowledge-based techniques, such as case-based reasoning, can be used to augment hybrid recommender systems, and address some limitations of recommendation systems, such as the new user problem and the new item problem [8]. For example, Entrée [39] is a restaurant recommender that uses some domain knowledge (such as *seafood is not vegetarian*) to produce its recommendations. The primary drawback of knowledge-based techniques is that they require knowledge; this could limit the application of this approach to domains where domain knowledge is available in some machine-readable form, such as an ontology [8]. For example, [145] uses a research paper topic ontology to recommend research articles.

3.1.4 Recommending Tasks

This section so far has described approaches to recommendation of items in general. However, different recommendation algorithms may perform better for different datasets, due to differences in domain features, for example [99]. That is, recommendation algorithms are domain dependent, and recommendation algorithms are only likely to achieve similar results on two different domains if they have similar characteristics [69].

A vital difference between recommending items and recommending tasks to produce flow is the challenge of finding tasks whose challenges match user skills. Consequently, the challenges of tasks and the user’s current skills need to be taken into account. Moreover, as a user learns, the level of his skills changes, and essentially, the user becomes
a different person. Burke referred to this as the “stability versus plasticity” problem [41]. A recent introduction to recommender systems cited this as a future direction of research in recommender systems, and labelled the problem “preference and assortment dynamics” [80]. While with other domains, changes to the user can occur over time, for example, as the user’s interests drift, nowhere are user changes more sharply evident than with knowledge and skills, where large changes can occur even within a single session.

This means that a user’s ratings are dependent on certain characteristics of the user when he rated the item. For the case of recommending tasks for flow, it means that a user’s ratings are dependent on the skills he had when he rated the task. An implication of this problem is that it makes some recommendation algorithms unsuitable for recommending tasks to produce flow, and this is an important consideration when selecting suitable recommendation approaches (see Section 4.1). Collaborative approaches are unsuitable for this reason, and this means a content-based approach is required, giving rise to a second challenge of task recommendation: accurately determining from tasks the information required by the recommendation algorithm.

The necessary information is much more difficult to obtain for tasks than for the usual items recommender systems are built for. For example, the necessary attributes of movies can be easily extracted from an internet movie database. With tasks, some attributes of particular importance to flow are the skills a task requires. Skills can be selected either manually or automatically. In a manual approach, the skills are manually associated with a task. In some cases, such as in the KBS-Hyperbook [97], weights are used to specify the importance of each of the skills in completing the task. Errors cannot be prevented even when attribute values are selected by a domain expert.

In an automatic approach, skills are automatically associated with a task, but this is suitable for very few domains [192]. One such domain is programming, and [192] describes an approach in which for each task, a solution file is parsed to produce a list
of required skills. A limitation of this approach is that it determines only whether a skill is required by a task, and not the importance of the skill. This is no small matter; suppose a user has little confidence in a certain skill, then whether that skill features very heavily or hardly at all in the task will greatly affect how difficult the user finds the task.

In sum, both the manual and automatic approaches are prone to error, leading to misguided values of the attributes, which in turn lead to poor recommendations. This motivates approaches to improving recommendations, introduced in Section 3.3.

### 3.2 Task Recommendation in Education

Almost all systems involving task recommendation are in the domain of education, in particular the areas of intelligent tutoring systems (ITS) and adaptive hypermedia (AH); a brief description of each of these areas follows. Intelligent tutoring systems are modelled on human tutors [13], primarily because individual human tutoring remains the most effective means of education in existence today [128]. An intelligent tutoring system typically has four components [210]. The domain model (expert model) represents the domain knowledge. It consists of knowledge elements each of which represents a subset of an expert’s knowledge [30]. The student model represents a student’s current knowledge. An example of a student model is an overlay model that represents the student’s knowledge as a subset of an expert’s knowledge by tagging each knowledge element in the student model with a yes or no depending on whether the knowledge element is known to the student [30]. The tutor model is responsible for making decisions about teaching, such as choosing content to present to students and adopting certain pedagogical principles. The final component of a typical ITS is the user interface.

Two key functions common to many tutoring systems are knowledge sequencing and
task sequencing [30]. Knowledge sequencing selects a sequence of knowledge elements from the domain model that is most suitable for a given student. Task sequencing, on the other hand, produces a sequence of learning tasks (examples, questions, problems, etc.) that is most suitable for a given student. That is, it provides an “optimal path” for the student to traverse through the learning material [31]. A third key function common to many tutoring systems is the provision of feedback to students who are engaged in a learning task, letting them know how well they are doing.

The area of adaptive hypermedia (AH) developed from two other areas: hypertext and user modelling [32]. AH systems differ from traditional hypermedia systems, which supply the same content to each user, by adapting content and links to the user. Nearly all adaptive hypermedia systems today do this with the combination of a domain model and an overlay student model described above [34]. Some AH systems provide navigation support, which was influenced by knowledge and task sequencing [31]. In addition, some AH systems supply users with feedback.

This section reviews some representative examples of systems that recommend tasks, in particular from the areas of intelligent tutoring systems, adaptive hypermedia, and recommender systems. The purpose of the review is to determine the extent to which they support the requirements set out in Section 1.3 focusing in particular on requirement R1, which is concerned with influencing the key conditions of flow.

### 3.2.1 KBS-Hyperbook

The KBS-Hyperbook System is a framework for designing and building adaptive hyperbooks [97], and it was used to create the CS1 hyperbook, a hyperbook whose domain is Java programming, designed for an introductory programming course. The domain a hyperbook is concerned with is decomposed into a set of knowledge items (KIs). Each hyperbook consists of a set of HTML pages and each page is associated with a set of KIs. The user model – a model of a user’s knowledge – is constructed from the
complete set of KIs, along with their dependencies. Users update their user model each time they complete a project.

Some of the KIs are linked, so that when a user rates a KI, it could influence the rating of the KIs linked to it. Essentially, this relationship expresses a belief: knowledge about KI, requires knowledge about KI. This is achieved by constructing a Bayesian network with the KIs as vertices, and edges linking each KI with those KIs deemed to be learning dependencies of it. For example, if a user is familiar with sorting, it is assumed that he is familiar with the quicksort algorithm. A conditional probability table is attached to each vertex, which determines the strength of these relationships.

Learning goals, which can be defined by the user or can be generated automatically, consist of mastering a specific set of KIs, and these goals, along with the user model are used to recommend suitable projects and suitable pages to read. KBS-Hyperbook calculates the suitability or fitness of a project for a particular user as follows. Suppose a user \( U \) has a goal \( G \). For each \( KI \in I(P) \), where \( I(P) \) maps a project \( P \) to the associated set of KIs, an estimation of \( U \)’s knowledge of that KI is calculated, giving a score between 0 and 100:

\[
\text{knowledge}(KI, U) = \left( P(KI = E).1 + P(KI = F).\frac{2}{3} + P(KI = A).\frac{1}{3} \right) .100
\]

Note that \( P(KI = E) \) is the probability that the user’s knowledge of that particular KI is expert level. There are four levels in total: expert (E), advanced (F), beginner (A), and novice (N). A project’s fitness, also a number between 0 and 100, is defined as:

\[
\text{fitness}(G, P, U) = \frac{\sum_{KI \in I(P) \setminus G} \text{knowledge}(KI, U)}{|I(P) \setminus G|}
\]

Fitness ascertains if “the actual knowledge of a user is sufficient for performing the suggested project without too many difficulties” [97], which is quite similar to estimating if a user believes he has sufficient skills to deal with the challenges of a task.
However, from the point of view of creating the key conditions of flow, the approach taken suffers from a number of limitations. For skills to be balanced with challenges, it is undesirable for either quantity to far exceed the other. But here, a maximum fitness of 100 could be achieved if a user’s knowledge greatly exceeds what is required, when it should report a poor fit. Additionally, for a user to update his model, he must pick one of the four levels mentioned above for each $KI$ associated with the project from the four options. Determining levels of knowledge in this way is similar to a user determining his perceptions of his skills. However, given a $KI$, for example, *multidimensional_array* from the CS1 hyperbook, it is unclear what precisely the $KI$ entails, and this may lead to inaccurate rating of $KIs$.

The conditional probability tables are estimates by the authors of the hyperbook of the strength of these relationships – there is no guarantee of the degree to which they reflect reality. Moreover, only a small number of distinct tables were produced and these were used as “blueprints”, ignoring the individual characteristics and thus compounding the inaccuracy. *This limitation could possibly be overcome by using a Dynamic Bayesian Network [182], and this could also allow the system to continuously improve its recommendations – something that is not currently supported.*

### 3.2.2 ELM-ART

ELM-ART (ELM Adaptive Remote Tutor) is a web-based ITS that supports learning to program in LISP [36]. It is an “interactive intelligent textbook”, and like a standard programming textbook, its contents are organized in a hierarchy [203]. Lessons (the top level) contain sections, which contain subsections, which contain units (the bottom “leaf” level). Each unit is on a separate page and could consist of an exercise, a question, a problem, or an explanation. Each unit has a coloured bullet beside it indicating if it is already learned, ready to be learned or not ready to be learned. A student can click on the “next” link that recommends the next unit to learn or she can
explore whatever units she desires. ELM-ART can also provide a detailed diagnosis of a student solution and example-based problem solving support (that is, the system supplies examples previously developed by the student that are most relevant to the problem at hand).

ELM-ART recommends tasks (the units described above), which potentially have clear goals. Like the KBS-Hyperbook, it determines if the student’s knowledge is sufficient to do a task, but it does so quite differently. The domain model is composed as follows. All the important concepts in the course, along with the relationships between them are represented in a conceptual network. The relationships used are “is a” and “part of”, and these can be used by the system to establish if one concept is a prerequisite of another. Each unit is indexed by a set of concepts related to the unit. This set, along with the kind of relationship that exists between each concept and the unit, is called the spectrum of the unit. The kinds of relationship include: a concept could be presented, or summarised, or a concept could be a prerequisite of the unit.

The student model is a combination of a multi-layered overlay model and an episodic student model. The episodic model is used for diagnosis and problem solving support and plays no part in determining the suitability of a unit; we therefore focus on the multi-layered overlay model. Automatically updating the student model gives rise to the problem of uncertainty: how can one be sure, from the evidence collected by the system – such as results of tests and exercises – that a student actually knows a particular concept? For this reason, the student model can also be updated by the student – he can mark particular units as already known. This is accomplished with the four layers of the overlay model: visited state (true if the user has visited the page corresponding to the given unit), learned state (true if the user has successfully done a certain number of exercises about requiring the concepts of the given unit), inferred state (true if the user knows a unit that the given unit is a prerequisite of), known state (true if the student decides that he knows the given unit).
A limitation of the approach to task recommendation in terms of flow is that the concepts can either be known or not, and this low level of precision may not be sufficient to maintain the “delicate balance” \[51\] between challenges and skills required for flow. Also, the relationships between the concepts and units are constructed manually, and may contain inaccuracies. Detecting and improving on such inaccuracies is not considered, but could result in improved recommendations. However, the content of the course was improved by getting direct feedback from users via the communication tool, which is part of ELM-ART.

ELM-ART provides feedback: when a student does an exercise, it can check the answer and inform him if he is right. Similarly, when a student does a problem, ELM-ART can determine if the solution is complete. The student model also contains the progress bar for each concept, letting students know how close they are to mastering each concept. An interesting effect was observed in ELM-ART: that it can increase student motivation to work on optional educational content \[37\]. This increased intrinsic motivation suggests that ELM-ART can increase the amount of time the key conditions of flow are present.

### 3.2.3 ADAPTS

Adaptive Diagnostics and Personalized Technical Support (ADAPTS) is an electronic performance support system for maintenance technicians \[34\]. The system adapts to who the technician is and to what he is doing. This is primarily done with three models: a domain model, a task model, and a user model. The domain model represents knowledge about the domain – such as an aircraft that requires maintenance. The task model represents a large set of maintenance tasks, and the user model represents an estimate of a technician’s experience with system tasks and components of the system.

The technician interacts with an adaptive hypermedia interface as he performs a task. Each task comprises a sequence of subtasks and steps, such as a variety of
operations on equipment and observing the value of certain measurable variables. The technician confirms whether all subtasks were completed, the values of his observations or whether he failed with a task. This information is passed to the diagnostic engine, which dynamically selects the next task to do.

The user model estimates a technician’s expertise using a weighted sum of a number of aspects including whether the technician has observed a task (weight 3), whether he believes he can do a task (weight 3), whether he has done a simulation of a task (weight 4), whether he has done a task (weight 5), and whether he has been certified on a task (weight 6). The user model is updated by the technician’s interactions with the hypermedia interface (for instance, when he confirms that all subtasks of a task are done or when he does a simulation of the task). The technician is supplied with information, the detail of which depends on his experience. An example of this is a technician who is inexperienced with a procedure will be shown an outline of the subtasks, whereas a technician who is experienced will not. Another example is the type of content supplied; technicians inexperienced with a step will be given links to fundamental concepts and background information, along with video clips and simulations, while the experienced technician will receive more concise information.

This approach has the potential to balances skills with challenges, since a task can be made easier by showing the necessary steps and making content such as videos clips, diagrams, equipment photos, and simulations available to the technician. A possible concern is that the user model represents an estimation of a user’s competence, which could be at variance with the user’s perception of his competence.

The technician receives feedback: colour coding and icons are used to mark out completed and uncompleted steps, giving him some idea of how close he is to finishing a task. This feedback could be improved by estimating the sizes of the steps and subtasks to get a more accurate idea of how much the task was done; for example, the first five out of ten steps could comprise 70% of a task.
3.2.4 Personal Recommender System for Learning Networks

*Learning networks* are composed of learners who can create, modify, and rate learning activities [72]. An ambitious project is currently underway involving the design, deployment, and evaluation of a Personal Recommender System (PRS) that recommends learning activities from a learning network [72, 71, 68]. The motivation of a PRS is to improve the effectiveness (in terms of goal attainment) and efficiency of the learning process [72]. Drachsler et al. identified seven requirements for a PRS in a learning network:

- **learning goal** (what the learners want to learn)
- **prior knowledge** (the proficiency level of learning activity should match the proficiency level of the learner)
- **learner characteristics** (including individual needs such as time constraints and whether the learner prefers distance education or problem-based learning)
- **learner grouping** (group learners by similar interests, studying at similar times, etc.)
- **rated learning activities** (aggregated ratings of learning activities could provide valuable information for other learners)
- **learning paths** (learners could benefit from the successful learning behaviour of more advanced learners; the most effective and efficient paths could be recommended)
- **learning strategies** (a PRS should use accepted learning strategies, which could use rules like “gradually decrease the amount of contact and direct guidance”; the strategies would require the metadata of the learning activities).
Drachsler et al. have determined that a hybrid recommender system is most suited to the purpose, and intend to use a simulation to determine different recommendation techniques to use depending on the condition of the learning network [68]. The intended approach to one of the learning strategies is of particular interest to this thesis. This strategy uses Vygotsky’s zone of proximal development [200], which suggests that recommended learning activities should have a competence level that is slightly above the learner’s current competence level [68].

To achieve this, Drachsler et al. suggest representing competence levels as integers (for example, subject X, competence level 2) [68]. This approach is the scalar model, which is reviewed in Section 4.1.2 on page 74; the main drawback of the scalar model is its low precision. While the scalar model simplifies indexing a learning activity, its imprecision is a major concern for maintaining the “delicate balance” [54] between challenges and skills required for flow.

Additionally, the goal of producing sequences of learning activities is certainly a desirable one. The proposed approach of finding sequences that successful learners followed and recommending them to other learners has a shortcoming in terms of creating the conditions of flow. While the skills of the successful learners may have improved in such a way that they were balanced by the challenges at each step along the sequence, it is likely that other learners’ skills will increase a different rate, conceivably keeping challenges and skills out of balance – and consequently the learner out of flow – for much of the sequence.

### 3.3 Recommendation Improvement

Recommender systems that do not improve over time will respond in the same way to users who find themselves in similar situations to previous users. This is not a problem when the previous users received good recommendations, but when the previous users
received poor recommendations, the current users will receive the same poor recommendations. In the case of task recommendation for flow, this will lead directly to the absence of the conditions of flow. This provides considerable motivation to continually improve recommendations. Existing approaches to improving recommendations include:

- **Collaborative approaches**
  Probably the majority of recommendation systems use a collaborative approach (described in Section 3.1), and recommendations in collaborative approaches improve automatically. As more users rate items, the number of users similar to a given user is likely to be greater, the information about items improves, and thus a user receives better recommendations.

- **Hone algorithms for a particular domain**
  Another approach is simply to develop better algorithms for a particular domain. In 2004, Herlocker et al. speculated that recommendation algorithms for movie datasets had reached an optimal level (taking human variability into account) [99]. However, that improvement, although extremely challenging, was still possible is illustrated by the Netflix prize, a competition organised by Netflix, an online DVD-rental service [153]. The goal of this competition was to provide a recommendation algorithm that improves on the accuracy of Netflix’s current algorithm by at least 10%. The grand prize of $1,000,000 has been pursued over the last three years by over 50,000 contestants. The accuracy of the submitted algorithms inched closer to the goal, until after almost 3 years, the goal was reached.

- **Hybrid approaches**
  Hybrid approaches to recommendation combine collaborative and content-based approaches, enabling them to improve recommendations by avoiding certain lim-
itations these approaches have. Hybrid approaches are described in more detail in Section 3.1.3.

- Multi-criteria approaches

Most recommender systems involve single criterion rating systems, that is, a user rates an item by rating a single criterion [9]. In a single criterion rating system, a user gives an item, such as a movie, an overall rating (for example, on a scale of 1 to 5). In a multi-criteria rating systems, a user rates an item on several criteria. For example, a movie might be rated along criteria such as story, acting, cinematography, etc. Recommender systems can use these multi-criteria ratings to improve the accuracy of recommendations [9].

- Using context

Integrating context into recommender systems extremely important for improving the quality of recommendation results [80]. For example, [7] showed that, by extending a typical movie recommender system to take context (such as when, where and with whom a movie is seen) into consideration, it was capable of outperforming the original system.

All of these approaches have limitations. For example, collaborative approaches suffer from the new user problem, the new item problem, and sparsity; honing algorithms for a particular domain is extremely challenging and time-consuming, as illustrated by the movie domain with the Netflix prize; and hybrid approaches reduce, but do not entirely remove, the limitations of the individual approaches comprising them. However, the main limitation is that, while each of these approaches has the potential to give better recommendations than a system that doesn’t use the approach, only the collaborative approaches continuously improve the recommendations over time. Moreover, it is demonstrated in Section 4.1.1 on page 73 that a collaborative approach can’t be used for recommending tasks for flow because collaborative approaches assume that

66
users’ characteristics remain static.

3.4 Conclusion

This chapter focused on the area of task recommendation. It examined approaches to task recommendation, reviewed some representative systems, examined approaches to improving recommendations, and identified limitations of existing approaches in recommending tasks to produce the key conditions of flow.

There are essentially three types of approaches to item recommendation: content-based, collaborative, and hybrid. In a content-based approach, items that score highest against a user profile are recommended. The limitations of this approach are: the new user problem, over specialisation, and a limited means of extracting necessary information from items to be recommended. In a collaborative approach, items are recommended to a user based on whether the items were liked by similar users. The limitations of this approach are: the new user problem, the new item problem, and sparsity. Hybrid approaches mix content-based and collaborative approaches, and as a result avoid some of the limitations of using one of approaches on its own. Some of the key hybrid methods were described along with their limitations. For example, one method involves switching between recommender systems depending on the confidence value each recommender system attaches to a recommendation. The main limitation of this method is that it is not straightforward to calculate this confidence value; indeed, it remains an area of active research.

A crucial difference between recommending tasks to produce flow and recommending other items is the challenge of finding tasks whose challenges match user skills. This means that the challenges of tasks and the user’s current skills need to be taken into account. Furthermore, as a user learns, the level of his skills changes, and essentially, the user becomes a different person. This has been called the “stability versus plas-
Table 3.1: Summary of the support of flow application requirements

<table>
<thead>
<tr>
<th>Requirement Description</th>
<th>KBS-Hyperbook</th>
<th>ELM-ART</th>
<th>ADAPTS</th>
<th>PRS</th>
<th>Hyperlead</th>
<th>IT-Emperor</th>
<th>SuperTangrams</th>
<th>Pearce et al.</th>
<th>SingStar</th>
<th>Burleson</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1.1 (recommend task with clear goal)</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>R1.2 (balance skills and challenges)</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>R1.3 (supply or enhance feedback)</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>R2 (improve recommendations)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td></td>
</tr>
<tr>
<td>R3 (measure flow)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td></td>
</tr>
</tbody>
</table>

○ = fully supported ● = partially supported

ticity” problem [41], and a consequence of it is that some recommendation algorithms are unsuitable for recommending tasks to produce flow. In particular, collaborative approaches are unsuitable for this reason, and this means a content-based approach is required, giving rise to a second challenge of task recommendation: accurately determining from tasks the information required by the recommendation algorithm.

Almost all systems involving task recommendation are in the domain of education, and some representative examples of these systems were reviewed with the aim of determining the extent to which they support the requirements of a flow application, set out in Section 1.3. Table 3.1 gives a summary of the extent to which both these systems and the computer mediated flow systems (see Section 2.2) support the requirements. With regard to satisfying the task recommendation requirements, the review of related work identified a number of limitations.

Firstly, the chief limitation of automatically updating the user model (as done, for example, in ELM-ART [203], ADAPTS [34], SuperTangrams [187], and the Cognitive
Tutor [52], and semi-automatically in the KBS-Hyperbook [97]) is that it is an objective measurement of mastery of skills, and there is no guarantee that this will coincide with a user’s perception of his mastery of the skills (required to meet one of the key conditions of flow). For instance, completing a task need not lead to an increase in perceived skill, or a skill may be forgotten or partially forgotten.

Secondly, all of the systems reviewed are limited by low precision, which is a concern for maintaining the “delicate balance” [54] between challenges and skills required for flow. SuperTangrams and SingStar [5] both use a Stereotype model, the PRS [72] and ADAPTS use a scalar model, and ELMART uses a multi-layered overlay model and the user’s knowledge of a concept is a binary value (known/not known). The KBS-Hyperbook allows users to update their perception of their knowledge of a particular concept by giving it one of four values. However, given a concept, for example, multi dimensional_array from the CS1 hyperbook created using the KBS-Hyperbook, it is unclear what precisely the concept entails, and this may lead to inaccurate rating of concepts, leading to poor recommendation.

Thirdly, tasks also need clear goals not only to meet the key conditions of flow, but also so that the task can be indexed properly (how can the required skills be determined if it is not clear what the goal is?). Some of the reviewed systems have clear goals (SuperTangrams and SingStar), and while the others potentially have clear goals, whether they actually do is not taken into account.

Recommender systems that do not improve over time will repeatedly produce the same poor recommendations, and in the case of task recommendation for flow, this will lead directly to the absence of the conditions of flow. This provides considerable motivation to continually improve recommendations. Of the reviewed systems, only ELM-ART considers continuous improvement: the content of the course was improved by getting direct feedback from users via the communication tool, which is part of ELM-ART. While this approach may identify content that needs to be improved, it
is limited because it only collects the opinions of users who choose to give it, and not the required information from all the users who used a certain piece of content. For example, from a handful of messages from users, how can one determine if the goal of certain content isn’t clear?

Other approaches to improving recommendations include collaborative approaches, honing algorithms for a particular domain, hybrid approaches, multi-criteria approaches, and using context. All of these approaches have limitations, but the main limitation is that, while each of these approaches has the potential to improve recommendations over a system that doesn’t use the approach, only the collaborative approaches continuously improve the recommendations over time. Collaborative approaches, however, can’t be used for recommending tasks for flow because collaborative approaches assume that users’ characteristics remain static, which is not the case when recommending tasks for flow.
Chapter 4

Design of a Task Recommendation System for Flow

It is possible to experience flow in almost any activity \cite{59}, and this motivated the goal of adapting or developing a set of task recommendation strategies that can support the creation of flow in a wide range of different activities. That is, given an activity, we would like to be able to use one of this set of task recommendation strategies in a flow application for that activity. In addition, since an activity often has characteristics specific to itself, it is necessary that the task recommendation strategies are extensible, so that they can be adapted to the activity at hand.

To get to a point where given almost any activity, a suitable task recommendation strategy is available for use in a flow application, many flow applications must be developed. When a flow application is being developed, if a suitable task recommendation strategy is available, it is used; otherwise a suitable task recommendation strategy must be found or developed. The more applications that are built, the more general the resulting system becomes \cite{181}, and the more likely it is that, given an activity, a suitable task recommendation strategy is available for it. However, developing applications is a lengthy process. Therefore, in order to produce this extensible task
recommendation system in a reasonable amount of time, a limit must be imposed on the number of applications developed; [181] suggests that this limit should be three.

This chapter describes the design of an extensible task recommendation system. First, a set of recommendation strategies are analysed against the requirements for a flow application, set out in Section 1.3 to determine which were suitable. Next, the design of each of the selected approaches applied to recommending tasks for flow are described. Finally, strategies for improving recommendations are considered, and a new strategy is developed and detailed.

4.1 Selection of Task Recommendation Strategies

In order to choose the three approaches for task recommendation, the candidate solutions were analysed against the relevant requirements for a flow application, set out in Section 1.3. In short, these requirements are that a recommended task must have clear goals, must balance the challenges of the task with the skills of user, and must enable the flow application to supply or enhance feedback. Also, the quality of the recommendations should improve as the flow application is used. In addition to these requirements, an important goal is that between them, the task recommendation approaches are suitable for a diverse range of activities.

During the analysis of approaches for suitable task recommendation strategies, we first considered what context (see Section 1.4) could be useful for the strategies to have at their disposal. The following context emerged: (i) an estimation of a user’s skills, (ii) the degree to which the key conditions of flow are present, (iii) a list of the completed tasks, and (iv) current task progress. For the analysis, it is assumed that this context is available. The context is acquired by means of context snapshots, which are a means of taking a snapshot of the state of the current context. How the context is acquired, and how it is modelled is described later in Section 7.1.9.
It was noted that in the review in Section 3.1 that there is empirical evidence that model-based methods outperform memory-based methods [20]. However, model-based methods have a critical disadvantage: a large amount of data (more than 10,000 items) is required to build a model that provides accurate recommendations [72]. It was not expected that any flow application would have such a vast number of items, certainly not until it was deployed for many years. Consequently, model-based methods are excluded as candidates. The candidates examined are collaborative approaches, content-based approaches, hybrid approaches, and Multi-criteria Decision Making (MCDM) approaches.

4.1.1 Collaborative Approaches

It was also noted in the review in Section 3.1 that recommendation approaches can be classified in a second way, as one of: collaborative, content-based, or hybrid. Collaborative approaches have many advantages, and an especially desirable one here is that recommendations improve as the system is used. However, it is not possible to use a collaborative approach to solve this problem.

This is because an underlying assumption of collaborative approaches is that a user’s characteristics do not change over time, that is, if a user rates an item that he has already rated, he will give it the same rating he gave it previously. If this is not the case, and the user would in fact make many changes to the ratings he has set down, then this user is, essentially, a different person from the one he purports to be.

To illustrate this, consider the following analogy in a different domain, movie recommendation. Suppose A, a 10-year-old who loves action films, uses a movie recommendation system and rates a hundred movies. Suppose then, that A does not use the system again until he is 30, whereupon, he rates a hundred movies, all different movies from the ones he rated when he was 10. He is much changed: now he finds action films pointless, but greatly enjoys character driven drama films. Now, because
another user, B, shares A’s passion for character driven drama films, A is selected as the user most similar to him, and as a result, B is recommended to watch an action film, which, needless to say, he cannot stand.

Depending on a user’s perception of his skills, he will almost certainly rate his confidence of doing a task quite differently. A user with low skills is essentially a different person from the user with high skills that he becomes, much as the 10-year-old is essentially a different person from the 30-year-old. As a result, users calculated to be similar to a user are unlikely to actually be similar to the user, and the accuracy of the user’s recommendations will suffer heavily. Burke referred to it as the “stability versus plasticity” problem and gave another clear example of it: in a restaurant recommender, a steak-eater who decides to become a vegetarian will continue to receive recommendations for steakhouses [41].

4.1.2 Content-based Approaches

In content-based approaches, a user model/profile is compared against item profiles to determine the most suitable items to recommend the user. The different content-based approaches differ in how the user is modelled, and how it is compared with an item profile. The main feature that must be modelled is a user’s skills, since they are required to recommend tasks of a suitable skill level. A number of different approaches to modelling users’ skills are considered. Although some may described as models of users’ knowledge, they can generally also be used for skills, which while different to knowledge, they are similar enough to be modelled in the same way – how well a person knows a concept or procedure is analogous to how confident a person is of performing a skill.

Scalar Model

The simplest model of user knowledge is the scalar model, which estimates the level of
a user’s knowledge of a particular domain by means of a single value \[35\]. This could be a quantitative scale (such as number from 0 to 5), or a qualitative scale (such as excellent, good, average, poor, none). Although it has been used effectively in a number of adaptive systems, the scalar model suffers from a significant limitation: low precision \[35\]. In any domain of a reasonable size, user knowledge can differ hugely for different elements of the domain; with word processing, for instance, the user could be an expert at annotating text, but a complete novice at inserting formulae \[35\]. Essentially, the scalar model provides is an average of the user’s knowledge of the domain. While the scalar model could be used for task recommendation, it would only be effective for very small domains. As we have set down as an important goal that the task recommendation strategies can be used for a diverse range of activities/domains, it must be concluded that the scalar model is not suitable.

**Stereotype Approach**

The next approach we consider is probably the oldest approach to user modelling: the Stereotype approach; it was developed about 30 years ago by Rich \[180\]. The essential idea is that users are placed into groups (stereotypes), and users from the same group receive the same treatment. When a user’s skills change, he is, if necessary, simply assigned to a different stereotype \[35\].

Returning to the problem at hand, once the user model is chosen (in this case, the Stereotype approach), it remains to compare the user model with the item (task) profiles. This can be achieved by associating suitable tasks with each stereotype. So R1.1 (matching skills and challenge) can be met. As for R1.2 (clear goals) and R1.3 (feedback), these can be met by assuming that unless proved otherwise, all the tasks have clear goals and provide adequate feedback. Then R2 (recommendation improvement) can be met by identifying tasks with poor recommendations from user ratings (that is, skills and challenges mismatch, unclear goals, or inadequate feedback), and

75
modifying the tasks accordingly. Finally, there are many activities for which this task recommendation strategy is suitable, since it is a commonly used model, for example piano (grades), karate (belts), computer games (levels), and courses (sections/units). In sum, an approach based on the Stereotype approach seems promising.

Overlay Model

The scalar model and the Stereotype approach both suffer from low precision. An approach that improves on this is the structural models, which assume that a certain domain of knowledge can be separated into independent fragments [35]. These fragments are called domain items; different systems use different names for them, such as concepts, knowledge items, and knowledge elements. The most popular form of structural model is the overlay model, which represents a user’s knowledge in terms of the domain model, a representation of the knowledge of an expert of that domain.

The simplest form a domain model can have consists of the set of domain items with no internal structure [35], that is, there are no relationships between the domain items. In more complex domain models, domain items can be connected to other domain items. There are two main types of such connected models; in the first, large domain items are progressively decomposed into smaller domain items; in the second, domain items can be connected by different kinds of relationships [35]. These relationships include “is a prerequisite of” (for example, [204]), and “is a” and “part of” (for example, [34]). The value of such links is that it makes it possible to propagate knowledge beyond direct observation; for example, if evidence of knowledge is observed about a domain item, then evidence of knowledge of prerequisite of domain items can be deduced.

There are several possible methods of representing the user’s mastery of a domain item. In some systems (for example, NavEx [38]), a domain item can have one of two states: mastered or not mastered. This means that at each moment in time, a user’s
knowledge is an exact subset of the expert’s knowledge. However, once again, this lacks precision. For example, a basketball player who sinks an average of one free throw out of ten is indistinguishable from a player who sinks an average of five out of ten. This has been improved by representing the degree to which it is known/mastered; it is then possible to distinguish different levels of skill. This could be qualitative, such as poor/average/good [33] or quantitative, such as real numbers from 0 to 100 [65]. Another possibility is to use an uncertainty-based model, where the user’s mastery of a domain item is usually represented by a probability (for example, [193]) or a probability distribution (for example, [97]).

Once again, returning to the problem at hand, the approach to modelling the user chosen, it remains compare the user model with a task. The challenge of a task may be characterised by the skills required to do it. The set of skills required to do a task can be mapped to the task by a process usually called indexing [34]. By comparing the required skills with a user’s perceived skills, a measure of how well they match may be estimated. A commonly used method is to calculate if the prerequisite domain items are considered to be mastered/known (for example, Navex [33]). Another method is to calculate a score for a task by weighting the contribution of each domain item. For example, a domain item that is well-known to a user is given a greater weight than an domain item for which the user has only beginner’s knowledge (for example, [97]).

In this way, the most suitable task can be found, which provides a means for meeting R1.1 (matching skills and challenges). As for R1.2 (clear goals) and R1.3 (feedback), these can be met by assuming that all the tasks have clear goals and provide adequate feedback, and observing user ratings to indicate otherwise. With regard to R2 (recommendation improvement) can be met by identifying tasks with poor recommendations from user ratings. With unclear goals, or inadequate feedback, they can be tagged for a content developer to modify. With mismatching skills and challenges, the user ratings can be used to estimate a better index for the task in question. Finally, there are many
activities for which this task recommendation strategy is suitable, since most domains can be decomposed into a set of domain items. Overall, an approach based on the overlay model approach can meet the requirements.

Updating the User Model Automatically

In the approaches described so far, the user model can be updated manually by the user; but it can also be updated automatically. In automatic approaches, a user’s behaviour is observed and appraisals of his knowledge or skills are inferred. The required evidence to prove mastery of a domain item varies. For example, it could be that the user must successfully complete a certain number of tasks associated with that domain item (such as in ELM-ART [203]). A limitation of this approach is that it doesn’t take forgetting into account; forgetting will result in the user model being at variance with the user’s actual ability at that moment.

The alternative is to automatically update a user model using an uncertainty-based approach. Such approaches model uncertain information; for example, the user failed to complete a certain task, so probably he has not mastered the domain item I. An example of a system that uses this approach is the Cognitive Tutor, which observes whether a user applies a rule each time he has an opportunity to apply it, and these observations are used to update the probability that the rule is in the learned state[52]. Most researchers who use an uncertainty-based approach choose to use Bayesian networks [35].

A Bayesian network is a directed acyclic graph G; each vertex in G is a random variable, and there is an edge from X to Y whenever Y is dependent on X; finally, each vertex is associated with a conditional probability table that quantifies the effects of its parent variables [97]. Once the network is defined, it can be given evidence and used to reason in the diagnostic direction (what are the likely causes?) and in the prediction direction (what is the probability that a certain configuration of variable states
Figure 4.1: A simple example of a Bayesian network, taken from [35]

will occur?) [35]. The following concrete example, taken from [35], illustrates how this works. Figure 4.1 shows a basic Bayesian network, the “Add” node represents the skill of adding natural numbers, and the “Multiply” node represents the skill of multiplying natural numbers. Suppose a student correctly solves “3+4”. This evidence can be used to reason in the diagnostic direction (that is, compute the probability that the student knows the domain item “Add”), and also to reason in the prediction direction (that is, to compute the probability that the student will correctly solve “3*2+7”).

Although the automatic approach is convenient for the user, this convenience comes at a price: one cannot be sure of the accuracy of the model [203]. For example, despite successfully completing a set of tasks, a user might not have mastered a domain item, or he may have mastered it, but have subsequently forgotten it, resulting in an inaccurate model. Or, in the case of a Bayesian network, the conditional probability tables are obtained either using a domain expert’s opinion or using learning algorithms [35], and both of these approaches can give rise to inaccuracies. In addition, a user has to do many tasks in order for the system to acquire the model. Essentially, the user has to prove to a system that he has certain knowledge or skills, and this could be extremely time-consuming. However, the principal argument against the automatic approach for flow applications is that it is an objective measurement of mastery of skills, and there is no guarantee that this will coincide with a user’s perception of his mastery of the
skills required to meet one of the key conditions of flow.

That is not to say that the automatic approach to updating user models has no place in flow applications. One extension of the overlay model is the \textit{layered overlay model}, which stores several representations of the user’s knowledge/mastery of a domain item\cite{35}. An example of a system that uses a layered approach is ELM-ART \cite{203}, which uses self-reports, tests (multiple choice), and exercises to gauge whether a domain item is known. In addition, a user can view the user model and mark a knowledge item as “already known”. This means that one of the disadvantages of automatic approaches is removed: if a user already has certain knowledge, he does not need to do a set of tasks to prove to the system that he has the knowledge. The layered approach could also assist in improving the accuracy of the user model – the layer that is automatically updated could be used to alert the user that he needs to manually update certain parts of his user model. This is discussed as future work in Section 8.3.2.

4.1.3 Hybrid Approaches

The final category of recommender systems is the hybrid approach, which involves a combination of collaborative and content-based recommendation. At first, it may seem as though this candidate should be ruled out immediately since it has already been shown that collaborative approaches are unsuitable because of the underlying assumption of a static user model. However, \textit{multi-criteria ratings}, a recent addition to the field of recommender systems has changed that. Most recommendation problems solved up to now have involved single criterion rating systems, that is, a user rates an item by rating a single criterion \cite{9}. In a single criterion rating system, a user rates an item, such as a movie, on a scale of 1 to 5. However, sometimes multiple criteria are required. The Zagat guide, for instance, rates restaurants according to four criteria: food, decor, service, and cost \cite{6}. Recommender systems can use these multi-criteria
ratings to provide more accurate recommendations [9].

The rating function of a multi-criteria recommender system may be written as:

$$r : U \times I \rightarrow R_0 \times R_1 \times \ldots \times R_k$$

where $R_0$ is the set of possible overall ratings an item can have, and $R_i$ is the set of possible ratings the $i$th criterion of an item can have ($i = 1, 2, \ldots, k$). Multi-criteria recommendation problems can be solved by decomposing the multi-criteria problem into $k$ separate single criterion problems [9]. Thus, hybrid recommendation can be used by separating the recommendation problem into three single criterion problems, one for each of the key conditions of flow. It was argued above that a content-based approach is suitable for matching skills and challenges (meeting R1.1). A content-based approach cannot be used for clearness of goal or feedback since they cannot be measured objectively. However, it reasonable that a collaborative approach could be used for these, since it seems likely that similar users will share opinions on the clearness of goals, and quality of feedback (meeting R1.2 and R1.3). Regarding R2 (recommendation improvement), this can be met using the approach described above in the Overlay Model subsection. Thus, a hybrid approach using multi-criteria ratings can meet the requirements.

4.1.4 Multi-criteria Decision Making (MCDM)

A limitation of the recommendation strategies considered so far is that they all require the tasks to be prepared in advance. For example, the Stereotype strategy requires that tasks are associated with an appropriate stereotype, and the content-based strategies require that the tasks are indexed. It is desirable that a user of a flow application could add his own tasks as he thinks of them, and get the most suitable tasks recommended to him. For example, a musician might add songs that he would like to be able to play. The strategies just mentioned would require users to position the tasks they create
within an appropriate unit, or to index the task. Not only is this time consuming, it is also not something that can be assumed of the average user. However, something that can be assumed of the average user is that for each task a user adds, he can estimate his confidence that he can do the task, how clear the goal of the task is, and the quality of the feedback (that is, at any point while doing the task, will he know how well he is doing?). The problem then is: given a set of such tasks each with a rating for each of the three attributes, find the most suitable tasks.

This problem is addressed by the field of Multi-criteria Decision Making (MCDM), which is concerned with the following general problem: from a set of alternatives (either finite or infinite), choose one (or a subset) of the alternatives by evaluating two or more criteria (also referred to as attributes); in general, this involves maximising a utility or value function (or the expected value of such a function in the case of uncertainty) [201]. MCDM problems fall into two categories: multiple criteria optimisation problems and multiple criteria discrete alternative problems [201]. Multiple criteria optimisation problems may involve an infinite number of alternatives, which are defined by systems of equations and inequalities that identify feasible regions; an example of this type of problem is engineering component design [201]. Multiple criteria discrete alternative problems typically involve moderately sized sets of alternatives, and unlike multiple criteria optimisation problems, the utility function is usually explicitly represented mathematically; an example of this type of problem is choosing which nuclear power plant to decommission [201].

The problem at hand is a multiple criteria discrete alternative problem, and we assume the ratings of the criteria are certain. This assumption is reasonable because the flow conditions are based on user perception. For example, if a user rates his confidence as 3, meaning that he believes he stands a chance of succeeding with the task, then this information is considered to be certain. The following approaches: the Analytic Hierarchy Process (AHP) [184], outranking methods (see Part 3 of [81]), and
Multi-attribute Utility Theory (MAUT) [112] are the three main approaches to solving multiple criteria discrete alternative problems [201].

The AHP was rejected as it is believed by many to be “fundamentally unsound” [190], and although this is hotly debated (see, for example, [89]), it seemed safer to use a different approach. Outranking methods consist of all methods that involve pairwise comparisons of alternatives along each criterion [81]. The main drawbacks of this approach is that it can be computationally expensive [49], and not as intuitive as MAUT, which is considered by many to be the “topmost” approach [201]. For these reasons, the MAUT approach was chosen.

4.2 Task Recommendation

From the analysis in the previous section, the three most promising approaches to task recommendation for flow were identified. These were the Stereotype approach, the MAUT approach, and the MCR approach. The aim of each approaches to recommend tasks that will produce the key conditions of flow, that is, they have clear goals, they supply users with feedback letting them know how well they are doing, and they balance the users’ perception of their skills with the perception of the challenge offered by a task. The last of these conditions is measured using confidence level, which is described in detail in Section 5.1.3 along with the reasons for using it. This section describes the design of each of these approaches for recommending tasks for flow.

4.2.1 Stereotype

The main idea of the Stereotype approach is that users are placed into groups (stereotypes), and users from the same stereotypes are treated the same. Each stereotype represents a certain mixture of features, such as knowledge, interests, and goals, but these are ignored in stereotype modelling, the stereotype is considered as a whole [35].
When a user’s features (in this case skills) change, he is assigned to different stereotype. Often, activation conditions are associated with a stereotype; these are conditions that allow the user to be identified as a member of that stereotype [118]. As we are dealing with skills, it seems that proof of mastery of the skills should be required to move to a different stereotype; mastery learning offers an appropriate approach.

Mastery learning is an instructional strategy developed by Benjamin Bloom [24]. The material to be learned is subdivided into units. Instruction, which can be of any kind (mastery learning does not place any restrictions on it), is completed for a particular unit, and the students are assessed. The assessment is based on criteria of mastery that have been decided upon in advance, and if a student successfully completes the assessment, he will have mastered the unit. If a student does not successfully complete an assessment, he is given further tasks to do and is subsequently re-assessed. This process is repeated until the student has mastered the unit. Once a unit has been mastered, students can engage in enrichments allowing them to take on “additional student-selected unit challenges” [25].

An example of system that uses a version of stereotypes combined with mastery learning is the WPS-Tutor, an intelligent tutoring system that teaches word problem-solving skills to children [205]. Problems are separated into levels, and each level is slightly more difficult than the previous one. Students are characterised entirely by the level they are currently on. In order to advance to the next level, students must successfully complete two problems from the current level without any help.

In order for this task recommendation strategy to be suitable for as many activities as possible, two alterations were made to make it more general. First, the default mastery criteria may be set by the developer. Second, rather than having a single sequence of units (levels), a hierarchy of units was used. In this way, if a user has mastered all of a unit’s prerequisite units (or if a unit does not have any prerequisite units), tasks from that unit can be recommended. In the simplest case, the hierarchy reduces to a
single sequence of units. Additionally, this task recommendation strategy facilitates an increase or decrease in difficulty—users can skip a task that is too easy, and set aside a task that’s too difficult until they have developed sufficient skills.

The recommendation process is formalised in the activity diagram shown in Figure 4.2. First, the set of all units that can be worked on is obtained (that is all units whose prerequisite units have been mastered). The first unit is selected and the context enables the last completed task of the unit to be determined. If no tasks have been done from that unit, the first task in the unit is set as the current task. If the selected task has been given a confidence level of 5 by the user, it is considered too easy and the next task above it that does not have a confidence level of 5 is set as the current task. If the last task is reached then this is set to the current task, even if it has a confidence level of 5. If the current task has been set aside (that is, the user found it too difficult), and the task directly below it has not been completed, then that task is set as the current task. The current task is added to the recommended list, and the process is repeated for all available units.

4.2.2 Multi-Attribute Utility Theory (MAUT) Approach

A limitation of the other recommendation strategies considered in this thesis is that they require the tasks to be prepared in advance. For example, the Stereotype strategy requires that tasks are associated with an appropriate stereotype, and the content-based strategies require that the tasks are indexed. It is desirable that a user of a flow application could add his own tasks as he thinks of them, and get the most suitable tasks recommended to him. For example, a musician might add songs that he would like to be able to play, or an office worker might add work tasks to do as they arise. The Stereotype and MCR strategies would require users to position the tasks they create within an appropriate unit, or to index the task.

Not only is this time consuming, it is also not something that can be assumed of
Figure 4.2: The recommendation process used in the Stereotype approach.
the average user. However, something that can be assumed of the average user is that for each task a user adds, he can estimate his confidence that he can do the task, how clear the goal of the task is, and the quality of the feedback (that is, at any point while doing the task, will he know how well he is doing?). The problem then is: given a set of tasks $T_1, T_2, \ldots, T_n$, each with a rating for each of the important attributes, find the most suitable tasks.

This problem is addressed by the field of Multi-criteria Decision Making (MCDM). Section 4.1.4 examined candidate approaches of MCDM, and selected Multi-Attribute Utility Theory (MAUT) [199] as the most appropriate for the problem at hand. MAUT is a method of evaluating the utility of items which have several competing attributes. Trade-offs between attributes are quantified with weights, and an aggregation function combines the values of the individual attributes. Usually, an additive aggregation function is used. Thus, given an item $i$, which has $k$ attributes, the utility of $i$ may be written as:

$$ r(i) = \sum_{j=1}^{k} w_j r_j(i) \quad (4.1) $$

where $w_j$ is the relative importance of the $j$th attribute, $r_j(i)$ is the rating given by a user to the $j$th attribute of the item $i$, and $\sum_{j=1}^{k} w_j = 1$.

In the problem at hand, there are three attributes: confidence level, clear goal, and feedback. The weights may be chosen by the user; default values are .5, .25, and .25, respectively. These values were chosen because Csikszentmihalyi noted that the balance between skills and challenges (that is, confidence level) is theoretically the most important [59]. The approach is extensible: if in a given activity, other attributes are important in addition to the three key conditions flow, they can also be taken into account. For example, an extra attribute might be interest — how interested the user is in doing the task, or environment — where the task can be done (outside or on a train, for instance).
4.2.3 Multi-Criteria Recommendation (MCR) Approach

The recommendation problem may be defined as follows: given a set of tasks $T$, find a utility function $r : U \times T \rightarrow [0, 100]$ that takes a given user $u$ and a task $t$, and produces a number representing the likelihood of the key conditions of flow being present if the user does the task $t$. 0 represents the conditions almost certainly won’t be present, and 100 that they almost certainly will be present. This is a multi-criteria recommendation problem (as described in Section 3.1), since it has three criteria of interest: clear goal (how clear the goal of task is), feedback (the quality of the feedback provided by the task) and balance of skills and challenges (the degree to which a user’s perceived challenges and perceived skills is balanced).

In order to solve this multi-criteria recommendation problem, the aggregation-function-based approach (outlined in [9]) was chosen because, unlike alternative approaches, such as the similarity based approach (also described in [9]), it does not limit the choice of recommendation algorithm. The rationale of this approach is that the overall rating of an item is based on the ratings of each of the item’s criteria, or, more formally:

$$r_0 = f(r_1, r_2, \ldots, r_k)$$

where $f$ is an aggregation function, $r_0$ is the overall rating, and $r_1, r_2, \ldots, r_k$ are the ratings for each of the $k$ criteria. The first step is to decompose the multi-criteria problem into $k$ separate single criterion problems (in this case, $k = 3$); the second step is to determine the aggregation function; and the final step is to use this function and the multi-criteria ratings to compute the overall ratings.

Confidence Level

The first single criterion problem concerns the confidence level criterion. A collaborative approach would be preferred since with this approach recommendations improve as the system is used. However, it is not possible to use a collaborative approach to
solve this problem because an underlying assumption of collaborative approaches is that a user’s characteristics do not change over time, as explained in Section 3.1.4.

As a collaborative method could not be used, the alternative, a content-based method, was chosen. In a content-based method, content is evaluated using a user profile. It can also be evaluated using a user’s context instead of a user profile such as in just-in-time information retrieval (JITIR) agents [179] and Context-aware Retrieval [109], which recommend text documents to a user, depending on his context.

The user’s perception of his skills, which could be considered either as context or as part of a user profile, can be modelled in a number of different ways, as described in Section 4.1.2, which suggests that the overlay model is a suitable approach for the problem at hand. The overlay model approach assumes that a domain can be broken down into independent fragments and a user’s current knowledge or skills is represented in terms of those of an expert.

As well as the representation of a user’s skills, it must be decided whether the skills model should be updated manually or automatically. To solve the problem at hand, a manual approach was chosen because although the automatic approach is convenient for the user, one can be sure of its accuracy. In particular, there is no guarantee that the objective measurement of mastery of skills that results from an automatic model approach will coincide with the user’s perception of his mastery of the skills. It may, however, be possible to improve upon this approach using a mixed approach, and this is discussed in Section 8.3.2.

While skills can be specific (e.g., throw a basketball through the hoop), they can also be generalised (e.g., write a Java method), and the skills model should be able to contain either. The problem with measuring generalised knowledge directly (as in the KBS-Hyperbook [97]), is that it can be unclear precisely what the knowledge fragment entails unless the user has already mastered it, applies equally to measuring generalised skills. A solution to the problem of measuring generalised skills is to estimate a user’s
confidence of a generalised skill by measuring his confidence of completing a set of related tasks, since one way of viewing skills is as generalised tasks. Suppose then, that we take a skill \( s \) that has \( m \) related tasks \( t_1, t_2, \ldots, t_m \), and that \( c_i : T \to \{1, 2, 3, 4, 5\} \) is a function giving the user \( u_i \)'s confidence of completing a task \( t \in T \), where \( T \) is the set of all tasks.

One way to estimate the user’s confidence of a skill would be to take the arithmetic mean of the user’s confidence of completing \( t_1, t_2, \ldots, t_m \). But this loses a lot of information. For example, a skill in which half of the tasks are given a rating of 1 and the other half a rating of 5 would be represented in the same way as a skill in which all the tasks have a rating of 3. A better approach to estimate a user \( u_i \)'s confidence of a skill \( s \) is to represent it as a random variable, \( C_{s,u_i} \), where the sample space is \( \{1, 2, 3, 4, 5\} \) – the set of possible confidence values a skill can have, and the probability that \( P(C_{s,u_i} = k) \) is calculated from the number of tasks for which the user \( u_i \) rates his confidence level as \( k \). Formally, this can be written as:

\[
P(C_{s,u_i} = k) = \frac{1}{n} |\{t_j : c_i(t_j) = k \text{ for } 1 \leq j \leq m\}|, \text{ where } 1 \leq k \leq 5
\]

The challenge of a task may be characterised by the skills required to do it. By comparing the required skills with a user’s perceived skills, a measure of how well they match may be estimated. The set of skills required to do a task can be mapped to the task by a process usually called indexing \([34]\). Suppose a domain is decomposed into \( n \) skills, \( s_1, s_2, \ldots, s_n \). The index function is defined as:

\[
I : T \to \mathbb{R}^n
\]

so that given a task \( t \),

\[
I(t) = (w_1, w_2, \ldots, w_n)
\]

where the weight \( w_i \) represents the relative importance of the skill \( s_i \) in the task \( t \), where \( 1 \leq i \leq n \) and \( w_1 + w_2 + \cdots + w_n = 1 \). The weights are estimated by a domain
expert. In order to estimate a user’s confidence level of completing $t$, the random variable $C_{t,u}$ is calculated:

$$C_{t,u} = \sum_{i=1}^{n} w_i C_{s_i,u}$$

(4.2)

$C_{t,u}$ is interpreted to be in one of three states; an example of each is illustrated in Figure 4.3. In the first state, “definitely won’t succeed/more than likely won’t succeed”, the challenges of the task exceed the perceived skills of user. In the second state, “definitely will succeed”, the perceived skills of the user exceed the challenges of the task. And in the third state, “stand a chance/probably will succeed”, the challenges of the task and the skills of the user are about the same. A measure of how well matched the challenges and skills are is $P(C_{t,u} = 3 \text{ or } 4)$, which is in the range $[0, 1]$, and the closer the value is to 1, the better the match.

![Figure 4.3](image)

**Figure 4.3**: Task confidence level: (i) “definitely won’t succeed/more than likely won’t succeed”; (ii) “definitely will succeed”; (iii) “stand a chance/probably will succeed”.

**Clear goals and feedback**

The second and third criteria are having clear goals and adequate feedback. Some attributes are difficult or impossible to measure objectively. For example, in restaurant recommender systems, one key attribute is food, but there is no objective measurement of food – it depends on a person’s taste. Since the values of such fields cannot be calculated objectively, as can, for example, a user’s distance from a restaurant or an
estimate of cost (€ to €€€€€€), it is not possible to use a content-based method. Clear goal and feedback are two such fields, and hence a content-based method could not be used.

The clear goal and feedback dimensions can be estimated using collaborative filtering, where the recommendation score of an item $i$ to a user $u$ is:

$$r(u, i) = z \sum_{u' \in N(u)} sim(u, u') r(u', i)$$

(4.3)

where $N(u)$ is the set of $N$ users most similar to $u$, $z$ is a normalising factor, set to:

$$\frac{1}{\sum_{u' \in N(u)} |sim(u, u')|}$$

and $sim(u, u')$, the measure of similarity between the users $u$ and $u'$ is calculated using the cosine measure.

The main limitations of collaborative systems – the new user problem, the new item problem, and sparsity – can be seen by examining the terms of Equation 4.3. In the new user problem, there isn’t enough information known about $u$ for $sim(u, u')$ to be accurate. In the new item problem, $r(u', i)$ will be 0, for most users $u'$. With sparsity, $r(u', i)$ will be 0 for most users $u'$ and most items $i$.

A user can be defined as a “new user” if he has rated fewer than $a_0$ items, where the value of $a_0$ can be chosen depending on the domain and the number of items available to recommend. It was decided that in such cases, since insufficient information is known about $u$, to use the average user instead. This is likely to yield better recommendations than by considering the user in the usual way [26]. The average user rating is obtained using:

$$r(u, i) = \frac{1}{|U_i|} \sum_{u' \in U_i} r(u', i)$$

(4.4)

where $U_i = \{ u' \in U : r(u', i) \neq 0 \}$ is the set of users who have rated the item $i$.

In the cases of the new item problem and sparsity, it is because particular items have so few ratings that they are rarely recommended. It can be seen from Equation
that such items, that is, items that satisfy $|U_i| < a_1$ (for some chosen $a_1$) are at a considerable disadvantage in comparison to items that have significantly more ratings – even if most of these ratings are poor. In order to give such items a fairer chance at being recommended, the utility function in Equation 4.4 is used instead of the utility function in Equation 4.3 for such items.

**The aggregation function**

The aggregation function is required to combine the values of each of the criteria into one overall value. Three methods for obtaining the aggregation function have been suggested: statistical techniques (such as linear and non-linear regression), machine learning techniques (such as artificial neural networks), and domain expertise [9]. The first two methods both require that the user, in addition to providing each of the multi-criteria ratings, also provides an overall rating for each item he rates. Suppose, for example, that linear regression is used to obtain the aggregation function. In this case, the aggregation function would be a linear combination of the multi-criteria ratings, $r_0 = w_1r_1 + w_2r_2 + \cdots + w_kr_k + c$ and the $w_i$’s and the constant $c$ may be estimated from the multi-criteria ratings (that is, $r_1, r_2, \ldots, r_k$) and the overall rating (that is, $r_0$) for each item that has been rated.

It is easy to see how an overall rating for an item can be provided in some domains in this manner. For example, in a movie recommender, a user could supply a rating for the story, the cinematography, the acting, and an overall rating for the movie. But with recommending tasks for flow, it is not so simple: the overall rating is a measure of the flow the task produced. This is not something a person can measure directly – he must instead rate it by rating components of it. This will change if it becomes possible to measure flow directly, such as by using EEG; this is discussed further in Section 2.1.4. For the present, however, it is not possible to obtain an overall rating directly, and consequently the first two methods (statistical techniques and machine
learning techniques) could not be used.

The third method, domain expertise, can be used. The basis of this method is that there is some particularity of the domain that will suggest the aggregation function. Certainly, each of the multi-criteria ratings (confidence level, clear goal, and feedback) is necessary. But are they equally important? Csikszentmihalyi claims that the balance between skills and challenges (described in this thesis as confidence level) is theoretically the most important [59], and for this reason the aggregation function chosen gives a higher weight to this component:

$$r_0 = f(r_1,r_2,r_3) = 0.5r_1 + 0.25r_2 + 0.25r_3$$

where $r_1$ represents confidence level, $r_2$ represents clear goal, and $r_3$ represents feedback.

### 4.3 Improving the Recommendations

Recommendation approaches in which recommendations do not improve over time can have a most undesirable consequence. Systems using such approaches will respond in the same way to users who find themselves in similar situations to previous users. Certainly, no problems results when the previous users received good recommendations. However, when the previous users received poor recommendations, the current users will receive the same poor recommendations, and this leads directly to the absence of the conditions of flow. This undesirable consequence provides considerable motivation to identify and improve upon the sources of poor recommendations, so that recommendations continuously improve over time, and the key conditions of flow are increasingly present.

A review of approaches to improving recommendations was given in Section 3.3. The main limitation is that, while a system using one of these approaches has the potential to improve recommendations over a system that doesn’t use the approach, none of the approaches continuously improve the recommendations over time (with the
exception of the collaborative approaches which are unsuitable for recommending tasks to produce the conditions of flow – see Section 4.1.1 on page 73.

An approach that can continuously improve recommendations is to modify items and/or items’ metadata, based on items identified using user ratings. No recommender systems could be found that take this approach. That no systems could be found that modify the items themselves is not perhaps all that surprising since most recommender systems recommend items that cannot be easily changed. For example, if a movie receives poor ratings, the movie is not remade and re-released, similarly with books and CDs. Additionally, that no systems could be found that change an item’s index based on user ratings is not that surprising either. For most recommender systems, it is extremely unlikely that the index is wrong because of the method of indexing, and consequently, the idea of changing the index would make little sense. For example, with the domains of movies, a movie’s attributes (such as actor, director, etc.) are automatically parsed from an online database, which would be very unlikely to produce an error.

However, tasks are different from the items found in the usual domains recommender systems are built for because firstly, they can, unlike remaking a film for example, be readily modified, and secondly, because some of the item attributes are not objective. Almost all recommender systems for task are in the domain of education. Here the items (usually called learning objects) are usually stored in repositories, such as the National Digital Learning Repository (NDLR) [154], which allow users to rate items, and these ratings could be used to identify items that need to be modified. The rating for an item is typically an overall rating, and a limitation of this is that it does not give the content developer information about the specific aspects of the item that need to be modified. For example, if the item is a task, is the low rating a result of an unclear goal or of something else entirely?

Indexing items is also different in the domain of education because some item at-
tributes in this domain are not objective. Unlike, for example, a film’s attributes, which are objective (the director of a film is not a matter of opinion), an attribute of a learning object, such as the specific topics the object is relevant to, does not have one set answer. Additionally, some attributes of particular importance to flow are the skills a task requires. Skills can be selected either manually or automatically. In manual approaches, the skills are manually associated with a task. In some cases, such as in the KBS-Hyperbook [97], weights are used to specify the importance of each of the skills in completing the task. In automatic approaches (for example, [192]), skills are automatically associated with a task.

Both of the manual and automatic approaches are prone to error, leading to misguided values of the attributes, which in turn lead to poor recommendations. In the manual approach, errors cannot be prevented even when attribute values are selected by a domain expert. The automatic approach is suitable for very few domains [192]. One such domain is programming, and [192] describes an approach in which for each task, a solution file is parsed to produce a list of required skills. A limitation of this approach is that it determines only whether a skill is required by a task, and not the importance of the skill. This is no small matter; suppose a user has little confidence in a certain skill, then whether that skill features very heavily or hardly at all in the task will greatly affect how difficult the user finds the task.

There is some interesting work on evaluating attributes of learning objects with the aim of alerting users about instances of low quality [160, 161]. In this work, a number of quality metrics are used on the attributes; an example of one is readability, which calculates a score of readability, giving higher scores (lower quality) where structures such as acronyms and complex sentences are present [161]. However, coming up with a quality metric for the skills attributes would be extremely difficult. Moreover, this work deals with identifying low quality attributes, but does not go into taking action, for example how to improve these low quality attributes, which is likely to result in
improved recommendations. Indeed, no recommender system that aims to improve the index of an item using user ratings could be found.

The following subsections describe a design for the approaches for recommendation improvement that was introduced above. In the case that the items themselves are changed – suitable for the clear goal and the feedback attributes, tasks that fall below certain criteria are identified (that is, tasks with unclear goal or substandard feedback) and flagged to allow content developers the opportunity to improve upon the content, rather than simply allowing items to fall out of use. In the case that items’ metadata is changed – suitable for using with confidence level, tasks with an unsuitable confidence level (that is, the task was either too easy or too hard) are identified and a better skills index is calculated. In both cases, the aim is that subsequent recommendations of the task will be better received.

4.3.1 Clear Goal and Feedback

The final context snapshot, made by a user upon finishing with a task (either completing the task or setting the task aside), contains a rating for clear goal and feedback. These ratings can be used to determine whether the goal and feedback of a task fall below a certain standard. There are a number of different ways of defining the standard. The simplest of these is probably to get the arithmetic mean of the ratings and compare this with 3, the minimum rating for each of these conditions of flow to be present.

Another approach is to use a t-test [95] on the values obtained from the sample of users who have rated the task enables a conclusion to be drawn about the population of all users. Thus, it is possible to determine if the mean rating of the population for clear goal or for feedback is less than a certain standard, such as a rating of 3, and to flag tasks that fall below the standard. This can be done by setting up a one tailed t-test where the null hypothesis is $H_0 : \mu > \mu_0$ and the alternative hypothesis is $H_1 : \mu \leq \mu_0$. 

97
\( \mu_0 \) is the standard just described, and if the evidence shows that \( H_1 \) is true then the task is flagged.

Tasks that resulted in poor recommendations because of an unclear goal or substandard feedback are flagged, using one of the above methods. In this way, the content developer who supplied the task can be notified, and can improve upon the clearness of the goal or the feedback component, and re-release the task. Without adopting such an approach, a task whose goal isn’t clear or whose feedback is poor would be recommended less and less and eventually drop out of use entirely.

### 4.3.2 Confidence Level

In this approach, the confidence level a user \( u \) has in doing a task \( t \) is estimated as

\[
C_{t,u} = \sum_{i=1}^{n} w_i C_{s_i,u}
\]

as described in section 4.2.3, where for each task \( t \), the index of the task (that is, the vector \( w = (w_1, w_2, \ldots, w_n) \in \mathbb{R}^n \)) is estimated by a domain expert.

The accuracy of this estimation can be tested by considering how likely it was that the chosen confidence value would be selected. Each time a user finishes with a task (that is, either the user completes the task or else he sets it aside), he makes a final context snapshot, which includes a rating of how confident he was doing the task. From this confidence value, the quantities \( c(s_i, u) \) are calculated for each \( i = 1, 2, \ldots, n \):

\[
c(s_i, u) = \begin{cases}
    c_h(s_i, u) = P(C_{s_i,u} = 1) + P(C_{s_i,u} = 2), & \text{if the confidence value is 1 or 2} \\
    c_r(s_i, u) = P(C_{s_i,u} = 3) + P(C_{s_i,u} = 4), & \text{if the confidence value is 3 or 4} \\
    c_e(s_i, u) = P(C_{s_i,u} = 5), & \text{if the confidence value is 5}
\end{cases}
\]

(4.5)

For each user, the quantity \( \sum_{i=1}^{n} c(s_i, u) w_i \) is an estimation of the likelihood of selecting the confidence level that the user selected. We can define the recommendation
to be successful by comparing this quantity to a modifiable threshold value, $t_0$, where $0 < t_0 < 1$, and the higher the value of $t_0$, the stricter the cutoff for success is. More formally, the recommendation is successful if the following linear inequality holds:

$$\sum_{i=1}^{n} c(s_i, u)w_i > t_0$$  \hspace{1cm} (4.6)

An inequality of this form exists for each user who did the task, giving rise to a system of linear inequalities. The accuracy of the estimation of the $w$ can be gauged by the number of the linear inequalities that hold. The task is identified as a poor recommendation if the number of the linear inequalities that hold is sufficiently low; this number is also modifiable to allow different levels of strictness in the definition of a poor recommendation.

Tasks that have produced a poor estimate of confidence level require a more accurate skills index, $w$, that is a $w$ that satisfies more of the system of linear inequalities described above than the current $w$. We therefore consider approaches to finding optimal solutions to a system of linear inequalities.

We note that in the problem at hand, we have a system of nonhomogeneous linear inequalities (that is, the right-hand side is not 0). It is considerably more difficult to solve a nonhomogeneous system. Fortunately, in this case, it is possible to obtain an equivalent homogeneous system. Since $\sum_{i=1}^{n} w_i = 1$, $\sum_{i=1}^{n} c(s_i, u)w_i > t_0$ may be written as $\sum_{i=1}^{n} c(s_i, u)w_i > \sum_{i=1}^{n} w_it_0$, which becomes:

$$\sum_{i=1}^{n} (c(s_i, u) - t_0)w_i > 0$$ \hspace{1cm} (4.7)

As has been noted, the inequality in Equation 4.7 exists for each user who did the task $t$, thus if we suppose $m$ users, $u_1, u_2, ..., u_m$, did the task, then the following homogeneous system of inequalities arises:

$$Cw > 0$$
where $C$ is an $m \times n$ matrix, and $C_{ji} = (c(s_i, u_j) - t_0)$, where $C_{ji}$ is the element on the $j$th row and $i$th column of $C$.

Consider a general system of homogeneous linear inequalities:

$$AB > 0$$

where $A$ is an $m \times n$ matrix with $m \geq n$ and $B$ a vector of dimension $n$. Let $J(B)$ be the number of inequalities satisfied by $B$. The system is consistent if there exists some $B^*$ that satisfies all the inequalities (that is, $J(B^*) = m$), and inconsistent otherwise. We require a solution that obtains a vector $B^*$ which maximises $J(B)$ in both cases. While many techniques exists to solve the consistent case, and also the inconsistent case in a least-squares sense, few techniques that maximise $J(B)$ exist. The algorithm supplied by Mengert [143] is not guaranteed to produce a solution in a finite number of steps. Linear programming techniques either cannot yield an optimal solution or else have serious shortcomings in terms of computational complexity [202]. The best solution that could be found, and hence the one used, is supplied by Warmack and Gonzalez [202], and it is outlined below.

Each row of $A$ is a point in $n$-dimensional Euclidean space ($E^n$) and hence induces a hyperplane (the $n$-dimensional equivalent of a plane in two dimensions):

$$H_i = \{ B | \alpha_i B = 0 \}$$

for $i = 1, 2, \ldots, m$. These $m$ hyperplanes form a finite set of convex polyhedral cones. Some of these cones have the property that any point in that cone maximises $J(B)$; these cones called minimum-error cones. A search sequence is constructed, containing at least one edge (defined in $E^n$ as the intersection of $n-1$ hyperplanes) of the boundary of every minimum-error cone. These edges are examined in turn, and a vector $B$ lying on the edge is calculated. The first vector $B^*$ in this sequence that gives the largest value of $J(B^*)$ for any vector in the sequence is moved from the boundary to the interior of the cone. $B^*$ represents an optimal solution.
In the problem at hand, we can calculate an optimal solution \( w' \). But should we change to \( w' \) immediately? Specifically, what is likely to happen as more ratings come in? We would like \( w' \) to satisfy some criteria before accepting it. Consider \( n \) ratings of the task. The chances of getting \( r \) successes out of \( n \) (that is, \( J(w') = r \)), can be modelled by the Binomial distribution \([163]\). Another criterion that can be used is \( \text{lowest}_n \) – the lowest number of ratings the task must have in order for the strategy to be permitted to change the index of the task.

4.4 Conclusion

This chapter described the design of an extensible task recommendation system using the ‘Three Examples’ development approach. Given an activity, one of the set of task recommendation strategies available in the system can be used either as it is or as the basis of an extended strategy in a flow application for that activity. First, a set of recommendation strategies were analysed against the requirements for a flow application, to determine which were suitable. From this analysis, the three most promising approaches were identified. These were the Stereotype, MAUT, and MCR approaches. Next, the design of each of these approaches for recommending tasks for flow was described:

In the Stereotype approach, a set of stereotypes are defined, each of which characterises the skills of the user. Users change from one stereotype to another when they have mastered the skills associated with the stereotype. The Stereotype strategy is suitable for activities in which tasks can be readily ordered by difficulties, such as relaxation. One benefit of this approach is that tasks can be readily added by a domain expert, as long as tasks can easily be ordered by difficulty in the domain under consideration. Another benefit is that it facilitates an increase or decrease in difficulty of the task is suggested to the user. However, this approach has three chief limitations.
First, clear goals and feedback are not taken into account. It is assumed that the tasks have been designed so that the goals are clear and the feedback adequate. Second, the user’s choice of what he can do next is limited – he cannot simply add a task he thinks of, that he believes he stands a chance at doing. Third, a user who has some of the skills already will have to do some or possibly many prerequisite tasks, to prove to the system that he has the skills he knows he has.

The MAUT approach is quite different from the Stereotype approach. In this approach, a user rates relevant attributes of tasks, and the relative importance of these attributes are quantified, resulting in an overall utility value for each task. The tasks with the highest utility value are suggested to the user. This approach has the potential to give the user much greater choice: any task that produces a high overall utility value is suitable. Moreover, a user can add to tasks, rate these tasks, and immediately these tasks have the potential to be suitable. In addition, unlike the Stereotype approach, a user who possesses some of the skills does not have to do any tasks to prove to the system that he has the skills. A drawback of this approach is that as the user’s skills increase, he will have to re-rate the tasks. If he does not, he is likely to get poor suggestions. A second drawback is that tasks that haven’t been rated by the user cannot be suggested. In applications in which a user adds the tasks himself, this isn’t a problem, but in applications in which other users supply most of the tasks, a user might have to trawl through hundreds or thousands of tasks.

In the multi-criteria recommender approach, the tasks are indexed by a set of skills, and the tasks are rated using the user’s perception of his skills. Clear goal and feedback are also taken into account in arriving at the overall score task. The primary advantage of this approach is that tasks that have not been seen by a user can be suggested to him. Another advantage is that difficulty can be increased or decreased by the user by reappraising his skills. The principal drawback of this approach is that indexing a task so that accurate suggestions result can be time-consuming, and not even a domain
expert can guarantee the accuracy of the results.

A promising approach to improve recommendations of tasks for flow is to modify items and/or items’ metadata, based on items identified using user ratings. No recommender systems could be found that take this approach. This informs the decision process governing the change to a new index. The main benefit of identifying subpar items (that is tasks with unclear goal or inadequate feedback) is that it gives content developers the opportunity to improve upon and re-release the items, rather than simply allowing items to fall out of use. The main drawback is the effort it requires from users (users must provide an estimation of the attributes for each task they do).

The approach for supplying better item metadata (that is, a better index) involves finding an optimal solution to a system of homogeneous linear inequalities. Once an optimal solution has been found, a probability distribution is used to consider the likelihood that the solution will continue to be a better skill index than the original index over time. The main advantage of the approach to supply better item metadata (that is, a better index) is that it is automatic – it does not require the input of a domain expert but only user driven context snapshots. Its main limitation is the number of users who must do a task before it can produce a more accurate index with a low risk of error (see Section 6.3.3 on page 161); this is unavoidable since sufficient data must be collected, and an alternative approach would necessarily have the same limitation.

While the approach for clear goal and feedback works for all three of the task recommendation strategies, the approach described for confidence level works only with the MCR task recommendation strategy, since only this strategy has a skill index for each task. In the MAUT approach, improving recommendations in this way is not possible because relationships are not defined between the tasks and skills as is the case in the Stereotype approach and the MCR approach. That is, recommendations are made solely on the information the user provides.

In the Stereotype strategy, while the approach could be used to identify poorly
recommended tasks, it is doubtful if an improvement strategy could work. Certainly, if a task has been identified as poorly recommended, it means that a task is misplaced. The question is how? If the tasks have been placed manually, by a domain expert, it could be a result of human error, and it will then be up to the domain expert to reposition the misplaced task. The task could be moved up or down in the hierarchy, additional tasks could be placed before the task, or the task could be removed. If the task has been placed automatically, for example if an estimation of tasks difficulty is estimated by a function, then what is to be done? Essentially this is a limitation of the Stereotype strategy – it really is only suitable for activities in which tasks are readily ordered by difficulty.

Two other limitations were identified with this approach. One limitation is that the rows of $A$ must be linearly independent (see page 99). However, this can readily be overcome by constructing $A$ row by row, and when a row is added that causes linear dependence, a very small random number ($< 10^{-8}$) can be added to one (or more if necessary) of its components. The result will be that the rows will be linearly independent, and the effect on the solution will be negligible. Another limitation is the requirement that $m \geq n$. This means that at least $n$ ratings are required before an optimal solution for the weights can be obtained. However, it is necessary for many ratings to be collected before enough information is available to determine whether the current weights are inadequate, so this would only really be a limitation if $n$ was very large, which, since it represents the number of skills in the domain, is unlikely to be the case.

The next two chapters include descriptions of three flow applications that were built, containing the Stereotype approach (Inka see Section 5.1.3), the MAUT approach (Musika see Section 5.2.1), and the MCR approach (Inka 2 see Section 6.1.3).
This thesis investigates the recommendation of tasks for producing the conditions of flow. In order to test the effectiveness of the recommendations strategies proposed by this thesis (described in the previous chapter), the general method used was to build and deploy a flow application containing the recommendation strategy being tested. The effects of the strategies were then assessed by measuring response variables – variables that can be used to test the hypothesis [117], which in this case is that each strategy effectively produces the conditions of flow.

This chapter describes two pilot studies, one whose aim was to measure the effectiveness of the Stereotype strategy, and the other whose aim was to measure the effectiveness of the MAUT strategy in producing the conditions of flow.

5.1 Pilot Study 1

The aim of this study was to measure the effectiveness of the Stereotype task recommendation strategy in producing the conditions of flow. The method used was to build and deploy a flow application containing the Stereotype strategy, and to measure the effects by measuring response variables. This section describes the setting, the sample,
the materials required (the flow application), the variables chosen to measure the effects of the Stereotype recommendation strategy, the procedure used, and finally, the findings of the study.

5.1.1 Setting

The study took place over three weeks in a meeting room in the Computer Science department of Trinity College Dublin.

5.1.2 Sample

As the purpose of Inka is to assist users to experience flow in the activity of Java programming, the population of interest in this experiment comprises beginners of Java programming – having varying degrees of knowledge and skill, but at least knowledge of the fundamentals: how to write, compile and run a simple program. The only accessible sample of this population was the students taking the Introduction to Programming course, which is taken by all first year students studying Computer Science in Trinity College Dublin. It was expected that the data would be collected in the classroom as part of the evaluation, but in the end permission to use Inka in the classroom could not be obtained. And, as no funding was obtained for the experiment, the students were contacted personally and asked to participate in this experiment, without payment.

As a result of this, a smaller than desired sample had to be settled on. Using the students’ grades for the labs and tutorials for the year to date, each interested student was placed into one of three strata: good, average, and weak. It was intended that the sample should be representative of a population in which each stratum was of equal size. However, from those who responded, the best that could be achieved was to choose one from the good stratum, two from the average, and three from the weak. While this was not ideal, it was still somewhat representative. That is, the sample
consisted of a mix of levels – albeit an unequal mix. All six volunteers were male, but this is unsurprising since some 37 of the 39 students in the class were male.

### 5.1.3 Materials

Inka\(^1\) is a flow application designed to cultivate flow in the activity of Java programming.

**Motivation for Inka**

Inka was designed to be used primarily in the tutorials of the Introduction to Programming course, part of the Computer Science course in Trinity College Dublin. The class is usually divided into small groups of around 14 students, and a teaching assistant is assigned to each group. In the tutorials, a problem, which must be solved before the end of class, appears on a big screen at the front of the classroom and the students write their solutions on paper. The students can seek assistance from their teaching assistant, who walks around the classroom. As the year progresses, the problems become more difficult. While some students manage to complete them with minimal help from their teaching assistant, many do not – even with considerable assistance from the teaching assistant. It is plain that these students do not possess skills that match the challenges of the task. Inka set out to produce the conditions of flow by making slight modifications to the way the tutorials were run.

**Design Challenges of Inka**

The design challenges of developing a flow application for the domain of introductory computer programming include:

\(^1\)Inka, a Japanese word, means acknowledgement by a Zen Master that a pupil has completed his training.
• Selecting and developing a task recommendation strategy that is suitable for the environment in which it is to be deployed (a classroom, and in which a course that has already been designed is run).

• Deciding what kind of feedback should be given to let the user know how well he is doing. Can it be generated automatically (in a similar manner to [96], for example)?

• Developing a method for measuring flow that is not inconvenient for the user but at the same time acquires sufficient information for improving task recommendations for current and subsequent users. Also, when should flow be measured so that sufficient data is gathered and at the same time the user isn’t unnecessarily interrupted?

Overview of Inka

In designing Inka, the following requirements were identified:

• The application must influence (that is, support the creation of) the three key conditions of flow. This requirement is composed of two sub requirements:

  – The application must recommend tasks, since to go into flow, one needs to be engaged in a task. Moreover, these tasks must have clear goals, and must be chosen so that the challenges of the task (as the user perceives them) are balanced with the user’s skills (as he perceives them).

  – The application must supply or enhance feedback from the recommended tasks, since receiving feedback is another of the conditions of flow.

• The application must measure whether the user is in flow, so that when he is not in flow, action can be taken to aid its return.
- The application must be mobile since it must be possible for the tutor to move around the classroom.

The key features of Inka are that it recommends suitable tasks to users, supplies users with feedback, and monitors the conditions of flow so that action can be taken of them at present. The approaches to task recommendation, feedback, and measuring flow taken in Inka are described below.

**Approach to Task Recommendation**

Recommendation of suitable tasks is imperative for any flow application, in order to experience flow, a person must be engaged in a suitable task. A suitable task has clear goals, and its challenges (as the user perceives them) are balanced with the user’s skills (as he perceives them).

Inka uses the Stereotype strategy, the design of which is described in Section 4.2.1. In Inka, the units are based on the chapters of the course book [43] that the course closely follows. Each unit is associated with a set of tasks that help a student master the unit. Each unit can have pre-requisite units which must be mastered before a student can begin doing tasks from a given unit. In addition, the units are ordered, based on the order they appear in the course book.

Each task that is already part of the course (that is, the task is used in the tutorials) can be readily associated with a unit. Many additional tasks needed be added to this set of tasks so that there is always a suitable task for a student to do. The original tasks were already ordered by difficulty from the course, and the additional tasks were placed in between the original tasks, so that the full sequence of tasks within a unit was in order of difficulty.

A class diagram showing the key classes of Inka is depicted in Figure 5.1. The `RecommendationManager` class is responsible for supplying a list of recommended tasks. It requires some implementation of the `MasteryCriteria` interface to determine
whether a given unit has been mastered. The implementation used stipulated that in order for a unit to be mastered, its top two tasks had to be successfully completed. The `ContextManager` class captures, stores, and retrieves context, including how confident the user is about doing a task and the user’s progress in a task. This context enables the `Unit` class to determine from which of the units tasks can be chosen, by determining if the prerequisite units have been mastered.

**Approach to Feedback**

Feedback, that is, information that lets a user know how well he is doing, is one of the three key conditions required for flow. In Java programming, it is possible to provide feedback in the form of an estimated percentage of the task complete. This can be

---

**Figure 5.1:** A class diagram depicting the key classes used in Inka
Figure 5.2: The session plan shows the name and progress (percentage complete) of each task. The main buttons are: go (show current task), edit (edit session), and snapshot (create context snapshot).

Figure 5.3: The student’s current task, showing the goal, progress, link to available materials (task resources), and a camera icon for creating a context snapshot.

achieved because it is possible to break the task of writing a program into stages, each of which has a weight. This can be encoded into a form, an example of which is shown in Figure 5.4. When the teaching assistant updates this form an estimated percentage is calculated and supplied to the ContextManager class which creates a new context snapshot. This update of context triggers a message to the student’s PDA, resulting in the student’s most up-to-date progress to appear on the student’s PDA (see Figure 5.3).
Figure 5.4: An example of the form for updating progress.
The feedback (the estimated percentage complete) is manually generated by the teaching assistant by means of the form shown in Figure 5.4. While there are Intelligent Tutoring Systems such as ActiveMath [141], that can automatically calculate a student’s progress in a task, such a mechanism is not possible for this activity, since the programs are written using pen and paper. Furthermore, even if computers were used in the class, the first half of each problem involves writing down an algorithm to solve the problem, using pseudo code or an activity diagram. Unless significant limitations were placed upon what could be written down, it would be extremely difficult for a computer to evaluate such an algorithm.

Approach to measuring flow

It is important for a flow application to measure flow so that action can be taken when a user is not in flow. The Experience Sampling Method (ESM), described in Section 2.1.4, is the most suitable candidate for measuring flow since it allows flow to be measured as an activity is happening. However, modifications were necessary to reduce the impact of its limitations.

One limitation of the ESM is the length of time required to fill in the ESF (the form that needs to be filled in as part of the ESM). The ESF has in excess of 30 measurement points, resulting in an average time of between one and two minutes to fill it in. While many of these points are related to flow, only two of them (challenges and skills) are generally used to indicate flow. Since we are interested only in whether a user is in flow and would also like the measurement to be taken quickly, we retain only challenges and skills from the ESF.

Secondly, the method of measuring skills and challenges in the ESM is ambiguous and may provide unreliable measurements [77]. The reasons for this are that it is not clear what specific challenges and skills are being measured, and because of the practice of transforming the measurements into the z scores, it is not clear what exactly
constitutes a match between challenges and skills. These problems are discussed in greater detail in Section 2.1.4.

In order to avoid the above problems, it was decided that the challenge/skill ratio would be measured indirectly. A person’s perception of challenges and skills can be viewed as his “confidence regarding what [he is] able to do in a situation” [105]. By measuring a person’s confidence level of completing the task at hand, we get an indication of whether the challenges of the task at hand and the relevant skills are in balance. Csikszentmihalyi puts it slightly differently by equating having a balance between challenges and skills with having a task “we have a chance of completing” [59]. The table in Appendix A.1 shows the possible options for confidence level. The low values of confidence level (1, 2) indicate the challenges exceed the skills, a value of 5 indicates skills exceed the challenges, and a value of 3 or 4 indicates the challenges and skills are about the same.

Thirdly, the ESM uses only one of the three key conditions of flow to predict flow experience. Measuring all three of the key conditions would provide stronger evidence of flow. More importantly, this information can be used to improve the clearness of goals and the feedback for the current user or subsequent users.

For these reasons, the variables confidence level, clear goal, and feedback were included in the form. Two other elements of flow also measured using the form were concentration and sense of control. The final element of the form was “meaningful”, which, though it isn’t mentioned explicitly as a condition of flow, it is suggested as a possible condition in, for example, [59] and [57]. It is a measure of how meaningful or important the activity is to the person, and high values usually arise from activities that are freely chosen. The form just described has considerably fewer variables than the ESF, hence the burden on the user is reduced.

In order to measure whether a user is in flow at a particular moment in time, the user must make a context snapshot by filling in the form shown in Figure 5.8. The
options available on the form are explained in the table in Appendix A.1. In the ESM, this calculation involves comparing the value of skill and the value of challenge and returning a positive response if they are equal. In the modified approach, the calculation returns that the user is in flow if the variables clear goal and feedback have a value greater than 2, and the confidence level variable has a value of 3 or 4. This represents a range of situations which are deemed to be “in flow”.

An issue that must be addressed is whether the adjustment that was made to the model for flow in everyday life (challenges and skills had to be in balance and also had to be above a certain level) should apply in the case of a single activity. Csikszentmihalyi suggests that it should not: the original model (see Figure 2.1 on page 21) illustrates “how the flow experience proceeds through time, in a single activity, from enjoyment of small challenges when a person’s skills are limited, to an ever complexifying enjoyment of higher challenges requiring increasingly rare skills” [62]. The reason he suggests for the difference is that in everyday life “activities that require skills alternate with routine, unchallenging episodes”. Pearce, faced with the same difficulty, argued that it was appropriate to use the original model [171]. It may be that challenges and skills must both be above a certain level, even in the case of a single activity. However, no research has been done to settle this point. As there is no evidence to the contrary, the accepted view – that the adjustment should not apply in the case of a single activity – is taken in this thesis.

5.1.4 Variables/Measures

In each case, the variable supplies a way of measuring the effectiveness of the strategy in producing the conditions of flow. The candidates considered for variables that could be measured to test the hypothesis were:

- Percentage of the time that the conditions of flow were present. This could be operationalised in several ways, which are described shortly.
• The number of red snapshots made in a session (where red snapshots are snapshots indicating the conditions of flow are absent). If it is assumed that the conditions are present unless a red snapshot is made, and that action is taken to bring about the conditions of flow once a red snapshot has been made, then it follows that the fewer red snapshots made, the more effective the strategy is.

• The conditions of flow for a task as a whole. After each task is done (that is, either successfully completed or else set aside), the user assesses the conditions of flow for the task as a whole.

In order to decide which one of these candidates to employ, the candidates were considered in terms of how informative each is. Measuring the conditions of flow for a task as a whole does not capture any information about how the conditions changed as the task was being done. The number of red snapshots made gives some indication of the changing conditions as the task was being done, but it is difficult to interpret because it isn’t uniform. For example, a user may make 4 snapshots in one task, and 4 in another, but in the first they have been made over one minute, and the second may have been over ten minutes. The variable does not capture any distinction between these two very different situations.

The final candidate, the percentage of the time that the conditions of flow were present, does not have this limitation, and was therefore chosen as the response variable to use. There are several ways of operationalising this variable:

1. Alert users at regular intervals, for example, once a minute, and ask them to assess the conditions of flow. If the intervals are short enough, this would certainly produce an accurate reading. However, since the technique used to measure flow involves interrupting the user each time, it is not feasible to employ this approach with short intervals since it will result in disrupting the user many times, while he is in the middle of a task, greatly affecting his concentration and thus taking
him out of the flow state. The same holds if, instead of regular intervals, the intervals are randomly spaced.

2. After a task or a session, the user estimates the amount of time he has been in the red. However, the user’s estimates are liable to differ considerably from the actual figures, especially because of the time distortion characteristic of flow, in which time generally appears to pass faster than normal (see Section 2.1.1).

3. Each time a user notices the conditions of flow have changed, he makes a snapshot. Assuming that the conditions are present if the most recent snapshot was green, and they are absent if most recent snapshot was red, the total time in the red can be calculated by considering each snapshot to be a coloured point on a line, and the line segment joining two adjacent points coloured the colour of the leftmost point. The total time in the red is then the total length of red line segments, as shown in Figure 5.5. This approach is likely to be less disruptive than alerting the user at intervals, because a user becomes aware of the absent conditions while focusing on a task, so the interruption is internal (for example, a user becomes aware that he doesn’t know how well he’s doing). This contrasts with the first approach in which a user is focused on a task and an alert, quite unrelated to what is happening in the task, disrupts his focus.

The above analysis of these three approaches led to the adoption of the third approach of operationalising the percentage of time in flow variable.

5.1.5 Procedure

As described in Section 5.1.3, the mastery criteria used in Inka were that the top two default tasks of a requirement must be successfully completed. Thus, prior to the first session, each student’s stereotype is set by creating a context snapshot for each of the tasks he successfully completed in the classes so far during the year.
Figure 5.5: The total time in the red is then the total length of red line segments; in this example, the total time is $a + b + c$.

The first session begins with an introduction to the procedure of the study. Subjects are briefed on the purpose, the meaning of the variables, and use of Inka, in particular making context snapshots, for which they received a handout they can consult throughout the session (see Appendix A.1).

An overview of the session is given by the activity diagram depicted in Figure 5.6, and details of each step are given below. The teaching assistant goes over to each of his students at the start of the tutorial and plans the session, which consists of a list of suitable tasks for that student (see Figure 5.2). The student sets to work on the current task which, instead of being presented on the big screen, is displayed on the PDA (see Figure 5.3). In this way, each student can be given a different problem, whose challenges are matched by the skills of the student.

A student may also use any of the resources that are available to him; an example of a resource is shown in Figure 5.7. Straightaway, he creates a context snapshot (see Figure 5.8) and subsequently, each time he becomes aware that one or more of the conditions of flow have changed, he creates another snapshot. These snapshots pop up on the teaching assistant’s PDA, marked with a green circle if the conditions of flow are present (this is defined in Section 5.1.3) and a red circle otherwise.

If one or more of the conditions are absent, the teaching assistant goes over to the student at his earliest convenience to rectify the matter. This is done in two main ways: providing feedback and supplying resources. To provide feedback, the teaching
Figure 5.6: An overview of a session using Inka
assistant estimates the percentage of the task complete. He updates the student’s progress on his PDA and a context snapshot is automatically created which causes the student’s most up-to-date progress to appear on the student’s PDA (on the screen shown in Figure 5.3). The teaching assistant can release a resource to a student, which can make a task easier, or make the goal of the task clearer.

If both of these measures fail (that is, if one or more of the conditions continue to be absent), the task is set aside – the teaching assistant modifies the session so that an easier task is placed in front of, or else replaces, the current task of the session. If the student completes a task, the completed task is greyed out and the next available task in the session becomes the current task. Or, if all the tasks in the session plan are completed, the teaching assistant can add more tasks. As before, the student sets to work on the new task. The process continues until the time (90 minutes) has elapsed, and the session is saved so that in the next session, the student can take up where he left off.

5.1.6 Findings

In the study, students took part in sessions individually. Each student did a minimum of two ninety minute sessions (one student did three, two students did four, and the remainder did two). Table 5.1 shows the total time in the red for each session a subject did, and also the percentage of the time the subject spent in the red (see also Figure 5.9). The mean percentage of time subjects spent in the red is 19.3%. This means that the average percentage of the time subjects spend with the three key conditions of flow present is 80.7%. This result should be considered from the perspective of the ideal: that subjects spend 0% of the time in the red, that is, the conditions of flow are present 100% of the time.

An unexpected and unwelcome behavioural pattern was observed in the subjects. Many times, after the teaching assistant had taken action and left the subject in a
Figure 5.7: An example of a task resource. Task resources make a task easier, or make the goal of the task clearer.

position where he felt he stood a chance of succeeding with the task, the subject would omit to make a context snapshot to indicate that the conditions of flow had returned. The error incurred by such an omission can be severe, as illustrated by the following simple example. A subject makes a red snapshot, the teaching assistant takes action, and 3 minutes later, the conditions of flow have returned. The subject continues on without needing to make a snapshot for 24 minutes. Once again, the teaching assistant takes action and once again 3 minutes later, the conditions of flow have returned. This time, the subject does make a context snapshot. However, the percentage time in the red is now reported as 30% instead of the actual much lower figure of 6%.
Mostly this was noticed and the student was reminded by the teaching assistant to make a snapshot, so that the effect on the reliability of the collected data was minimal. A consequence is that the average percentage of the time subjects spend in the red is even lower than the reported 19.3% when the effect of the subjects’ omission to make context snapshots is taken into account. This is because each omission causes a subject’s total time to be greater than it actually is, and consequently causes the mean percentage of the time subjects spent in the red to be greater than it actually is.

While the result does provide evidence of the effectiveness of the Stereotype strategy, the small sample size limits the strength of the result. In order to obtain a more reliable result, a larger sample would be required; this would enable statistical analysis to be performed on the data. Moreover, a more accurate measure (without the drawback of the measure used) would have to be employed.
Table 5.1: Time with the conditions of flow absent broken down by session and subject.

### 5.2 Pilot Study 2

The aim of this study was to measure the effectiveness of the MAUT recommendation strategy. The general method used was, as in Pilot Study 1, to build and deploy a flow application containing the recommendation strategy, and to measure the effects by measuring response variables. This section describes the materials required (the flow application), the variables chosen to measure the effects of the MAUT recommendation strategy, the procedure, and finally, the findings of the study.

#### 5.2.1 Materials

Musika is a flow application that assists musicians in experiencing more flow, while practising and playing music.

**Motivation for Musika**

Playing music is a very different activity to computer programming, and it was chosen primarily because of this characteristic. One goal of the task recommendation strategies described in this thesis is that they should be usable for building flow applications for
almost any activity (since it is possible to experience flow in almost any activity). To make such generalisation possible, flow applications for a range of activities must be considered since different activities may require different considerations. For example, when playing music, a person often attends the same task and number of times before it succeeds, which is not the case with computer programming. Another motivation for choosing the activity of playing music is that the method of supplying feedback is quite different to computer programming, and will result in having more methods of feedback available for use in other flow applications for activities that have a similar structure of feedback.

Application Overview: Musika

In designing Musika, the following requirements were identified:

- The application must influence the three key conditions of flow. This requirement is composed of two sub requirements:
The application must suggest tasks that have clear goals and whose challenges (as the user perceives them) are balanced with the user’s skills (as he perceives them).

The application must supply or enhance feedback from the suggested tasks.

- The application must measure whether the user is in flow, so that when he is not in flow, action can be taken to aid its return.

- The application must be mobile since it must be possible for the musician to use the application wherever he usually plays or practices music: at home, in a rehearsal room, in a hotel, etc.

A fundamental difference between the domains of Java and music is that in music, a person often attempts a task many times before he succeeds. Moreover, musicians often work on easier tasks that are stepping stones to the end task (such as playing only the left hand of a section, or playing a set of bars at a slower than normal pace). Different people working on the same piece may choose different tasks as stepping stones to the end task.

To accommodate these differences, some concepts and behaviour that were not present in Inka, were introduced. These are included in the class diagram in Figure 5.10, which shows the key classes of Musika. First, users can create their own tasks, and these tasks can have subtasks. Second, users can create projects to store related tasks. Finally, it was necessary to introduce the concept of a repeated task – a task that has to be done more than once. The user can browse through projects, adding tasks and subtasks, and rating tasks. To rate a task, a user must create a flow snapshot, which gauges if the conditions of flow would be present if the user was doing this task now.

The key features of Musika are that it recommends suitable tasks to users, and supplies users with feedback. The approaches to task recommendation, feedback, and
Figure 5.10: A class diagram showing the key classes in Musika
measuring flow taken in Musika are described below.

**Approach to Task Recommendation**

Musika uses the MAUT approach to task recommendation, which is described in Section 4.2.2 since it enables users to add tasks on-the-fly, whenever they think of piece of music they want to be able to play. In this approach, tasks are ranked according to several competing attributes, the importance of which are quantified with weights, and are combined with an aggregation function. For flow applications, there are three base attributes: confidence level, clear goal, and feedback. However, it is an extensible approach that allows other relevant attributes to be considered in the evaluation process to arrive at a recommendations score for a task.

In Musika, an extra attribute, desire to play, was added; allowing users to rate the level of their desire to play a particular piece. The weights may be chosen by the user; default values are .45 (confidence), .225 (clear goal), .225 (feedback), and .1 (desire to play). In this way, the highest ranked tasks are those tasks most likely to produce the conditions of flow, which the user most desires to play.

**Approach to Feedback**

J.S. Bach once remarked “There is nothing to it. You only have to hit the right notes at the right time, and the instrument plays itself”. As a musician plays a piece, probably the most fundamental piece of feedback he can get is knowing whether he has hit the right note. To do this, a musician must be able to compare what the piece should sound like with what it does sound like. This depends entirely on having a detailed mental representation of the piece of music. However, even if the musician does not have this, it is still possible to get this feedback for some musical instruments, as described below.

Musical instruments are either polyphonic or monophonic. With a polyphonic instrument, such as a piano, several notes can be played at a time; whereas, with a
monophonic instrument, such as a clarinet, only one note can be played at a time. Although automated real-time transcription of polyphonic music is quite susceptible to error, monophonic music can be transcribed in real-time “fairly reliably” [140]. A number of Digital Signal Processing techniques can be used to achieve this. One method is to use a Fast Fourier Transform to obtain the most powerful frequency of a sound [137], which is the note that has been played (each note has a unique frequency, for instance, the A above middle C has a frequency of 440 Hz). By encoding a song as a sequence of frequencies, it is possible to compare the frequency of the note a musician actually plays with the frequency of the note he should have played.

One way to provide the user with feedback from this is illustrated by the karaoke game SingStar [5]. SingStar graphs the frequency of the note the musician should have played, and in a different colour, it graphs the frequency of the note he actually plays. Thus, it is possible for the singer to see if he has hit the note, or if he has not, he can see if the note he sang was too high or too low, and how far away from the right note he was. Musika models its feedback representation on SingStar; as shown in Figure 5.11 – the green graph represents the frequency he actually played, and grey represents the goal frequencies. It also shows the percentage complete of the task (time elapsed), and score, calculated from the frequency comparison.

In Musika, sensing the sound and converting it into frequencies was simulated, because it is nontrivial to implement (see [137] for an example of an implementation) and moreover, it is not the focus of this work. As the class diagram in Figure 5.10 shows, there are two instances of Frequency. The first is the simulation just described – it represents the note that the musician is currently playing. The second represents the correct frequency; the correct frequencies are read from a file. Both instances of Frequency are observed by FrequencyComparisonView, which is responsible for displaying the visual feedback. The FrequencyComparisonScore uses the cumulative difference between the correct and actual frequency used to calculate a score, and
Figure 5.11: Task screen of Musika showing: comparison between goal notes (grey) and notes played (in green), \% task complete (time elapsed), and score.

This numerical feedback is displayed by the ScoreView class.

**Approach to measuring flow**

Musika measures flow in much the same way that Inka does. However, it wasn’t clear in the earlier publications (such as [59]) which elements of flow are characteristics of flow, and which are conditions of flow. This was clarified in later publications (such as [57]), in which the elements were explicitly divided into conditions and characteristics. It transpired that two elements used in Inka as conditions of flow, namely concentration and sense of control are actually characteristics of flow. Nevertheless, the cost of this error was minimal, since it merely resulted in gathering some superfluous data. The only consequence was that it took longer for a user to create a context snapshot.

Thus, Musika, unlike Inka, includes only the three key conditions of flow in the context snapshot. As in Inka, the user must make a context snapshot by filling in the form – the options for each variable on the form are given in Table 7.1 on page 188. The calculation is identical to the one used in Inka, that is, it returns that the user is in flow if the variables clear goal and feedback have a value greater than 2, and the
5.2.2 Variables/Measures

One crucial difference between programming and playing music is that when playing music, the opportunity to take a couple of seconds to make a context snapshot does not arise. Snapshots could only be made after a task has been attempted. Thus the only suitable variable from the candidates given in Section 5.1.4 is the conditions of flow for the task as a whole.

5.2.3 Procedure

The user begins by rating the set of available tasks by creating snapshots (as in Inka, except it contains four attributes: confidence, clear goal, feedback, and desire to play, as discussed above). These are the initial snapshots and represent the state of the conditions of flow if the subject was to do the task at that moment. He can also add his own tasks to the set of available tasks. From the task recommendations he receives, he then makes a session plan which, as in Inka, is composed of a set of tasks. When he clicks on a task in the session, he is shown the task screen (shown in Figure 5.11), which shows both the goal and feedback.

The task is either aborted or completed (completing a task may require doing it several times if it is a repeated task). The user then creates a final snapshot, which gives a rating of the suitability of the task having done the task. At this point, the user can edit the session or continue on with the session as is; in either case, the next task becomes the current task. This process continues until the session is over (either all the tasks of the session are completed or the user decides to end the session, which he can resume on another occasion).
5.2.4 Findings

It was planned to measure the effectiveness of the MAUT strategy using the same variable used in Study 1 and Study 2 (see Chapter 6), that is, the conditions of flow for a task as a whole. However, a problem with this was identified. The MAUT strategy produces a recommendation score for each of the available tasks according to the user’s initial snapshot. It follows that a highly rated task will generate a high task score (defined in Section 6.1.4) if the initial snapshot closely resembles the final snapshot.

However, whether the initial snapshot closely resembles the final snapshot depends on the user’s ability to correctly gauge tasks, and the data gathered from the studies so far suggests that this requires a lot of practice. To claim that the MAUT strategy performs poorly, when in fact the cause for its poor performance lies in the user’s poor ability to correctly gauge tasks, would be erroneous.

To determine a user’s ability to correctly gauge tasks would require a large-scale investigation, in which each user would rate a large set of tasks, do the tasks, and then re-rate the tasks. There is no substitute for doing the tasks, which is an extremely time-consuming process. The study could not be conducted due to a lack of resources. It is intended to study whether the difference between the user’s initial snapshot and final snapshot diminishes over time as part of a future longitudinal study.

5.3 Conclusion

This thesis investigates the question of how tasks can be recommended so that the key conditions of flow are produced. Two strategies for doing this are the Stereotype strategy and the MAUT strategy, both described in the previous chapter. This chapter described two studies aimed at evaluating the effectiveness of these two recommendation strategies. The general method used was to build and deploy a flow application containing the recommendation strategy being tested, and to measure the effects using
relevant response variables.

In Pilot Study 1, the effectiveness of the Stereotype strategy was evaluated by building a flow application that contained the strategy for the activity of introductory programming. In order to evaluate the effectiveness of the Stereotype strategy in producing the conditions of flow, the percentage of the time that the conditions of flow were present was measured. However, an unexpected and unwelcome behavioural pattern was observed in the subjects, whereby many times, subjects would omit to make a context snapshot to indicate that the conditions of flow had returned. This resulted in the recorded time with the flow conditions absent being longer than it actually was.

The mean percentage of time subjects spent with the conditions of flow absent was found to be 19.3%. This result compares well with the ideal of 0% (that is, subjects spending the entire time with the conditions of flow present). However, it was expected that the data would be collected in the classroom, and in the end permission could not be obtained. As no funding was obtained, subjects were asked to participate in this experiment without payment. This resulted in a smaller than desired sample size of six. Consequently, while the result does provide evidence of the effectiveness of the Stereotype strategy, the small sample size limits the strength of the result. In order to obtain a more reliable result, a larger sample would be required; this would enable statistical analysis to be performed on the data.

An insight gained from this study was that, due to unexpected user behaviour, the percentage time with the conditions of flow absent measure lacks accuracy. The only approach to addressing this issue that could be conceived was to automate a reminder. This would entail the application observing when the subject hasn’t made a context snapshot within a short time (such as half a minute) of the teaching assistant’s departure, and it could remind him with an auditory alert. However, this is very similar to the approaches discussed in Section 5.1.4, in which subjects are alerted at regular or random intervals, and asked to create a snapshot.
Like those approaches to measuring effectiveness, this approach disrupts concentration, and hence flow. That is, the method of measurement is capable of changing the quantity it measures. As this solution is unsatisfactory, and no better solutions could be conceived, it was decided that this approach for measuring effectiveness of the Stereotype strategy should be discontinued and a different approach, without this drawback, should be employed to measure the effectiveness of subsequent strategies.

In Pilot Study 2, the aim was to evaluate the effectiveness of the MAUT strategy by building a flow application that contained the strategy for the activity of practising or playing music. This activity was chosen primarily because playing music is a very different activity to computer programming, and this is conducive to generalising the task recommendation strategies, so that they could be used for practically any activity. It was planned to measure the effectiveness of the MAUT strategy using the conditions of flow for a task as a whole. However, a problem with this was identified. The MAUT strategy produces a recommendation score for each of the available tasks according to the user’s initial snapshot, but an accurate recommendation score depends on the user’s ability to correctly gauge tasks; the data gathered from the studies so far suggests that this requires a lot of practice.

To determine a user’s ability to correctly gauge tasks would require a large-scale investigation, in which each user would rate a large set of tasks, do the tasks, and then re-rate the tasks. There is no substitute for doing the tasks, which is an extremely time-consuming process. Consequently, the study could not be completed due to a lack of available resources. However, it is intended to study whether the difference between the user’s initial snapshot and final snapshot diminishes over time as part of a future longitudinal study. Moreover, although this study was unable to produce data on the effectiveness of the MAUT strategy, it nevertheless does have some merit – it makes a necessary contribution to development of the framework for flow applications (see Chapter 7).
Chapter 6

Main Studies

This chapter describes three studies that evaluate the effectiveness of recommendation strategies described in this thesis. These studies address a number of shortcomings of the pilot studies. The aims of the first study are to evaluate the effectiveness of the MCR strategy and to identify any usability problems with the flow application used in the study. This study has a larger sample size than the pilot studies, as funding was secured making subjects easier to attract. Other shortcomings from the pilot studies addressed in this study are that it uses a different measure so that subjects’ omission to create snapshots is no longer an issue, and it also evaluates usability – this is an important aspect of the research question since producing effective recommendations will be of limited value if the user is not satisfied with how the recommendations are produced.

The aims of the second study are the two aims from the first study, along with a third: to evaluate the strategy for identifying and improving poor recommendations. In addition, it addresses some shortcomings of the first study. Rather than users individually doing sessions with the flow application, the second study more faithfully implements the actual teaching scenario for which the flow application was designed – all the users are in a classroom at the same time and do their sessions in parallel.
Moreover, the study enabled considerably more data to be gathered since it took place in classes over 10 weeks of a single semester course, compared with the two ninety minute sessions used in Study 1.

The third aim of the second study required substantially more users than were available. Consequently, in order to provide a better evaluation of the strategy for improving poor recommendations, a third study was conducted which sidestepped the problem of lack of users by simulating them. The aim of the study was to evaluate the effectiveness of the strategy in improving poor recommendations caused by an inaccurate skills index.

6.1 Study 1

This study had two aims. The first aim was to measure the effectiveness of the MCR task recommendation strategy in producing the conditions of flow. The second aim was to identify any usability problems with the flow application used in the study. This is an important aspect of the research question since producing effective recommendations will be of limited value if the user is not satisfied with how the recommendations are produced. Adaptive systems can have usability problems that can outweigh the benefits they confer [106], and it is therefore important to identify any such problems. For example, if users find the time investment too great or find certain aspects of the system to be extremely inconvenient, it may well cause them to avoid the system.

In order to measure the effectiveness of the MCR recommendation strategy, the same approach that was adopted in the pilot studies was used. That is, a flow application containing the recommendation strategy was built and deployed, and its effects were measured using a suitable variable. In order to identify usability problems, users, after two sessions using the flow application (Inka 2), filled in a questionnaire (reproduced in Appendix A.4), and took part in a short semi-structured interview based on
the questionnaire. This section describes the setting, the sample, the materials required (the flow application), the variables chosen to measure the effects of the Stereotype recommendation strategy, the procedure used, and finally, the findings of the study.

6.1.1 Setting

The study took place over four weeks in a classroom used for tutorials in the Computer Science department of Trinity College Dublin.

6.1.2 Sample

Subjects for this study were, as in Pilot Study 1, taken from the students taking the Introduction to Programming course in the Computer Science department at Trinity College Dublin. It may be that students with low skills, average skills, and high skills respond differently. In order to ensure that each type was represented in the sample, stratified random sampling was used. In this method of sampling, the population is divided into disjoint subsets (or strata), each representing a particular characteristic. In this case, we have three strata, one for each of weak, average, and good students. Random samples are taken from each stratum, in proportion. In the absence of information to the contrary, it was assumed that the population consists of an equal number of each type.

Students were contacted and asked to fill out an initial questionnaire (see Appendix A.3), which included a question about whether they would be willing to participate in this experiment, for a fee, as funding was obtained for this experiment. There were 30 respondents, and 28 expressed interest in participating in the experiment. The students were each placed in one of the strata using the students’ grades for the labs and tutorials for the year to date, and these categorisations were verified by the teaching assistants. A sample of 15 students was selected, 5 randomly selected from each stratum, using Microsoft Excel’s RAND() function.
6.1.3 Materials

For this study, two materials were required: a questionnaire (given in Appendix A.4), designed to identify usability issues, and Inka 2, a flow application containing the MCR recommendation strategy, which is described below.

Application Overview: Inka 2

The application used with a study developed was Inka 2, which was based on Inka. Ideally, this application would have been developed for a domain different to the domains of the first two applications. However, the cost of content development was prohibitive, and this led to the decision to revisit the first domain, Java programming.

The requirements identified in designing Inka 2 are slightly different from those of Inka, and are as follows:

• The application must influence (that is, support the creation of) the three key conditions of flow. This requirement is composed of two sub requirements:

  – The application must recommend tasks since to go into flow one needs, at any moment, to be engaged in a task. Moreover, these tasks must have clear goals, and must be chosen so that the challenges of the task (as the user perceives them) are balanced with the user’s skills (as he perceives them).

  – The application must supply or enhance feedback from the recommended tasks, since receiving feedback is another of the conditions of flow.

• the application must measure whether the user is in flow, so that:

  – when he is not in flow, action can be taken to aid its return.

  – subsequent users in similar situations should be more likely to experience flow.
• The application should increase its ability to assist its users to experience flow as it is used.

• The application must be mobile since the tutor must be able to move around the classroom.

Inka 2 is used in much the same way as Inka. One notable difference is that users must create a final context snapshot when they finish with a task (that is, they either complete a task or they else set it aside). A user rates how confident he was doing the task, the clearness of the goal, and the level of feedback he experienced, having had the experience of actually doing the task. This information is used for making recommendations of the task to subsequent users.

**Approach to Task Recommendation**

Inka 2 uses the MCR approach to task recommendation, described in Section 4.2.3. In this approach, the tasks are indexed by a set of skills, and the tasks are rated using the user’s perception of his skills. Clear goal and feedback are also taken into account in arriving at the overall score task. Task difficulty can be increased or decreased by the user by reappraising his skills. The principal drawback of this approach is that indexing a task so that accurate suggestions result can be time-consuming, and not even a domain expert can guarantee the accuracy of the results.

**Approach to Feedback**

Feedback in Inka 2 is as it was in Inka, with two differences. Firstly, additional visual feedback is provided (see Figure 6.1). Secondly, in addition to receiving feedback from the teaching assistant, the student can also manually update the TaskProgress context. The teaching assistant must provide feedback at least once, that is, at the
least, when the student has completed the task, to certify that it has indeed complete.

**Approach to Measuring Flow**

Flow is measured in Inka 2 in the same way it was measured Inka, with one difference. After Inka was completed, a clarification of the flow model showed that there are three key conditions of flow. Therefore, the extraneous conditions used in Inka were not used in Inka 2, which measures only the three key conditions of flow. As in Inka, the measurement is made by means of a context snapshot (see Section 5.1.3).

**6.1.4 Variables/Measures**

As a result of Pilot Study 1, it was decided to discontinue using the percentage time in flow as a measure of effectiveness of task recommendation strategy (see Section 5.1.6). Consequently, one of the other candidates for measuring flow had to be selected. Mea-
suring the number of red snapshots suffers from the same shortcoming as percentage 
time in flow (users omitting to make context snapshots). Thus, the remaining candi-
date was selected – the conditions of flow for a task as a whole would be assessed by 
the user after each task.

Measuring the conditions of flow for the task as a whole is done almost identically 
to making a context snapshot, except some of the questions and possible responses are 
slightly different to take into account that it is for a task that was just done as opposed 
to a task currently being done. These questions and possible responses are shown 
in Appendix A.2. In order to more easily evaluate task recommendations, a variable 
called task score was introduced that consolidates the values of the conditions of flow 
into a single number. It is similar to the aggregation function used in the multi-criteria 
recommender strategy (see Section 4.2.3). It may be calculated as follows:

\[
\text{task score} = (0.5r_1 + 0.25r_2 + 0.25r_3) \times c
\]

where

\[
\begin{align*}
  r_1 &= \begin{cases} 
    5, & \text{if confidence = 3 or 4} \\
    0, & \text{otherwise}
  \end{cases} \\
  r_2 &= \begin{cases} 
    \text{clear goal, if clear goal} \geq 3 \\
    0, & \text{otherwise}
  \end{cases} \\
  r_3 &= \begin{cases} 
    \text{feedback, if feedback} \geq 3 \\
    0, & \text{otherwise}
  \end{cases}
\end{align*}
\]

The coefficients (0.5, 0.25, and 0.25) are the default values, chosen because of the 
relative importance of the conditions of flow, as described in Section 4.2.2. Multiplying 
by the constant \( c \) is a linear transformation that enables the score to be transformed to 
a particular range. By choosing \( c = 20 \), the task score is an integer in the range \([0,100]\), 
making it easier to interpret – 100 represents the ideal situation, and 0 represents the 
worst situation. In the ideal situation, the skills and challenges are in balance, the goal
could not be clearer, nor could the feedback be improved. In the worst situation, the
skills and challenges are not in balance, the goal could not be any more unclear, and
feedback is non-existent or erroneous. The minimum score at which the conditions of
flow are present is 80, which corresponds to \((r_1, r_2, r_3) = (3, 3, 3)\).

Recommender systems are usually evaluated using coverage and accuracy metrics
[8]. Coverage is the percentage of items the recommender system is capable of recom-
mending. In this system, the coverage is 100%. Accuracy measures are categorised as
either statistical or decision-support [98]. Statistical accuracy metrics generally com-
pare the estimated ratings with the actual ratings – examples include the Root Mean
Squared Error (RMSE) and the Mean Absolute Error (MAE) [8]. These metrics are not
well suited to evaluating this system, since the estimated ratings and actual ratings use
different scales. Decision support measures calculate how well the recommender sys-
tem predicts highly rated items – examples include precision (the percentage of highly
rated items in the set predicted to be high), recall (the percentage of all the highly
rated items that are recommended), and F-measure (the harmonic mean of precision
and recall) [8].

Calculating recall in this system is extremely difficult because it would entail find-
ing all the highly rated tasks for a particular skills model. To achieve this, each task
would have to be done and rated by a user with that particular skills model. That a
user’s skills model may have changed after doing each task compounds the problem.

The focus, therefore, is on the precision metric. A standard approach to obtain
more reliable results from a relatively small dataset is to use 10-fold cross-validation
[9]. In this approach, the dataset is randomly cut into ten disjoint subsets; nine of
these subsets are used as training data to predict the data in the remaining subset,
and the process is repeated ten times. An underlying assumption of this approach is
that there is a fixed user model, and the more data that has been acquired, the better
the approximation of the user model. Here, however, the user model is dynamic and
changes as the user learns, and thus the cross-validation approach cannot be used here.

The approach taken instead to measure precision is to perform the same process on the dataset in its entirety, that is to count the number of tasks predicted to be highly rated that were highly rated by users. We define a task to be highly rated if it has a task score of 80 or more, as this indicates that the three key conditions of flow were present during the task.

6.1.5 Procedure

In the first session, the subject is briefed on the purpose of the study, the meaning of the variables, and the use of Inka. As an additional aid, he receives a handout that he can consult throughout the session (see Appendix A.2). Next, the subject initialises his skills model, by rating his perception of confidence that he can do each of a set of tasks.

The remainder of the session is as described in Section 5.1.5 – Inka is used just as it would be used in the classroom. In addition, subjects are required to make snapshots only when the conditions of flow are not present (to allow action to be taken to bring about a return of the conditions) and when they have finished with a task. Previously, a snapshot was required any time the conditions changed, but this is no longer required because of the new method of measuring the effectiveness of the recommendation strategy.

After 90 minutes, the session ends and is saved so that the subject can take up where he left off at the start of the second session. After the second session, the subject fills in the questionnaire described in the materials section above (Section 6.1.3), followed by a short semi-structured interview, based around the questionnaire.
6.1.6 Findings

Effectiveness of the MCR Strategy

A total of 25 tasks were completed during the study – all the subjects did at least one task, and most did two. All 25 of the tasks were predicted to be highly rated (that is, the task has a task score of 80 or more). Of these, all 25 were highly rated, giving a precision of 100%. Figure 6.2 illustrates this more precisely by depicting the frequencies of the task scores. The average tasks score was 93.4.

Figure 6.2: The frequencies of the task scores.
Assuming that the task scores are normally distributed, it is possible to estimate the mean task score using the t-distribution, which given a small sample of a population enables the mean of the population to be estimated [95]. Given a sample of size \( n \), a confidence interval for the population mean may be obtained by:

\[
\bar{x} \pm t^\alpha_{\nu} \sqrt{\frac{s^2}{n}}
\]

(6.1)

\( \bar{x} \) is the mean of sample; \( \alpha \) is the significance level; \( \nu \) is the number of degrees of freedom of the distribution; \( s \) is the estimated standard deviation; and \( n \) is size of the sample [164]. In this case, \( \nu = n - 1 = 24 \), and choosing a significance level of 5% (that is, \( \alpha = .05 \)), the following result is obtained: 93.4 \( \pm \) 1.85.

That is, with 95% confidence, the mean task score is in the range [91.5, 95.25]. This result is well above the minimum of 80 (the minimum requirement for the three key conditions of flow to be present), and combined with a precision of 100% certainly supplies evidence of the effectiveness of the MCR recommendation strategy. However, it is important to observe that this result is limited, since despite the time-consuming nature of the gathering of this data, it is still a relatively small dataset (just 25 tasks were rated). In order to obtain a stronger result, more data is essential.

Usability

Table 6.1 summarises the responses gathered from the questionnaire (reprinted in Appendix [A.4]). As the purpose of this part of the study is to identify usability issues, only those items for which possible usability issues arose are discussed in detail below, along with possible solutions.

As flow is characterised by a deep sense of concentration, and it follows that distractions from the task at hand (item 2) may be detrimental to flow. 80% of the sample did not find the flow application to be distracting. The remaining 20%, which comprised three subjects, found it somewhat distracting. However, one of the subjects, Subject
<table>
<thead>
<tr>
<th>Item no.</th>
<th>Responses</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>v inconvenient</td>
<td>inconvenient</td>
<td>neither</td>
<td>convenient</td>
<td>v convenient</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>33</td>
<td>47</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>v distracting</td>
<td>distracting</td>
<td>somewhat distracting</td>
<td>not distracting</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>87</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>93</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>completely unnecessary</td>
<td>sometimes necessary</td>
<td>necessary more often than not</td>
<td>necessary most of the time</td>
<td>always necessary</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>47</td>
<td>47</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>40</td>
<td>40</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>40</td>
<td>20</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>strongly disagree</td>
<td>disagree</td>
<td>neither</td>
<td>agree</td>
<td>strongly agree</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>60</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>yes</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>47</td>
<td>53</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 6.1:** A summary of the responses to the questionnaire items (in percentages).
2 said it was “just because it’s new and I’ve never really used PDAs before. Once you got used to it, it would be fine”. Another of these subjects, Subject 1, echoed these sentiments. The last of these subjects, Subject 4, had found it “distracting because you keep having to think how well you’re doing and how your confidence is all the time, rather than just focusing on the task”. However, it is only at points in the task where the key conditions aren’t present, and one is as a result not immersed in the task, that one’s attention should be drawn to these variables, and he conceded this point.

It is important that context snapshots are convenient to create (item 1) because it is something used in most flow applications, and also could affect the level of distraction. Creating context snapshots was found to be convenient or very convenient by 67% of the sample, and the remaining 33% found it neither convenient nor inconvenient. Though these results show that making context snapshots is not perceived as inconvenient, there is certainly room for improvement. It may be that better results would be obtained after the subjects had more practice. Subject 5 observed that he didn’t get used to making context snapshots, but noted “I imagine you would get used to making them”.

When a task is recommended to a user, it may occasionally transpire that the recommended task is not suitable. In such cases, the user must be willing to set the task aside (item 3); otherwise, flow will not be possible. In the case study, 87% of the sample said they would not have any difficulty setting such a task aside. Two subjects, expressed that they would have difficulty. For example Subject 9 said: “it depends on how far through it I was, I guess. I mean, if I was halfway through the task, yeah, I probably would.” However, both the subjects were labouring under the misapprehension that once a task was set aside, it could not be resumed at some point in the future. Once this was explained, the objections were dropped.

In this study, one of the top rated tasks was chosen for the user. But do users want more choice (item 9)? 43% of the sample said they would like more say, while 57% maintained that they would not like more say. If users were given more say,
most subjects said they would choose a task that was rated highly and that seemed interesting to them; in the words of Subject 4 “I’d probably look through it and see... I’d look for interesting sounding ones ... I’d choose it high enough on the list as well”. The issue of more choice of task depends on the size of the content repository – it needs to be very large indeed in order that it should have many highly rated tasks available for each possible state the skills model can be in.

However, this is quite within reach – lack of content is no longer an issue, instead the issue is profusion of choice [162]. A vast repository would solve the issue of lack of choice, but the resulting long lists of highly recommended tasks would lead to a need to extend the MCR recommendation strategy. One possibility would be to combine it with MAUT, so that, certain the user could indicate a preference (for example, tasks involving games or about real-world applications) would appear higher on the list.

The feedback objects, an example of which is shown in Figure 5.4 on page 112, are important as they assist in supplying feedback to the users. Did the users find them useful (item 4)? All but one of the sample found them useful. This subject, Subject 10, didn’t find them useful because as she put it “I do that in my head myself. It just irritated me that I had to click it.” Moreover, the automatic calculation of percentage completed, praised by many of the subjects, was not well received by Subject 10, as it did not always concur with the calculation made by Subject 10: “The ones there were useful but ... I know myself that some of the stuff would take me longer than others – the percentage wouldn’t always be right.” However, Subject 10 mentioned that this limitation could be remedied by allowing users the option of altering the percentages: “having a list and a number myself would be fine”.

While a number of issues related to usability were identified, no serious issues were identified. That is, in each case, a solution for overcoming the issue readily suggested itself.
This study had three aims. The first two aims of the study were identical to those of Study 1 (Section 6.1), that is, to measure the effectiveness of the MCR task recommendation strategy in producing the conditions of flow, and to identify any usability problems with the flow application used. The third aim was to evaluate the strategy for identifying and improving poor recommendations. Although the same approach was taken, this study addresses a limitation of Study 1: that students took part in sessions individually. In this study, an entire class group (consisting of 12 students) took part in sessions together – the teaching scenario for which the flow application was designed. Moreover, the study enabled considerably more data to be gathered since it took place in classes over 10 weeks of a single semester course, compared with the two 90 minute sessions used in Study 1.

There were two parts to the study. Firstly, in order to measure the effectiveness of the MCR recommendation strategy and the strategy for identifying and improving recommendations, a flow application (Inka 2) was deployed and used for recommending tasks, and data was collected by means of the context snapshots. Secondly, in order to identify usability problems, at the end of the 10 weeks, users filled in a redesigned and improved questionnaire (reproduced in Appendix A.6). From the experience of Study 1, it was decided that the questionnaire would be sufficient to gather the required data, and that a supplementary interview would not be necessary.

This section describes the setting, the sample, the materials required (the flow application and the new usability questionnaire), the variables chosen to measure the effects of the Stereotype recommendation strategy, the procedure used, and finally, the findings of the study.
6.2.1 Setting

The study took place over ten weeks in the laboratory classes of an introductory programming course in the Institute of Technology, Carlow.

6.2.2 Sample

An accessible sample for this study was found in the Institute of Technology, Carlow, where a group of first year students were taking an introductory programming course. The class comprised 12 students, and had roughly equal number of students in the weak, average, and good categories. The students were categorised, as in Study 1, using the students’ grades for the labs for the year to date, and were verified by the teaching assistants.

6.2.3 Materials

Two materials were acquired for this study: a flow application, Inka 2, slightly modified from Study 1, and a usability questionnaire (reproduced in Appendix A.6), the design of which is described below.

Inka 2

The introductory programming course in which this study took place did not have any tutorial classes – all of the non-lecture classes took place in laboratories. As a result, mobile devices weren’t necessary, since each student has a desktop computer in front of them. A new user interface, shown in Figures 6.3 and 6.4, was developed for use in this environment.

Study 2 Questionnaire

To design a good questionnaire, “considerable effort and thinking” are required.
Figure 6.3: Switching tasks in Inka 2.

**Figure 6.4:** The show task screen of Inka 2.
The questionnaire itself is given in Appendix A.6 and its design is described here. The aim of the questionnaire was to identify any usability problems. That is, the construct being measured by the questionnaire was usability. Usability is a high-level construct, and high-level constructs must first be defined by a set of measurable items [28]. Using multiple questions to measure a single construct or a dimension of a construct reduces the risk of the subject supplying an incorrect or incomplete answer because he failed to understand a specific question or neglected to consider all of the issues involved [28]. The number of questions required will be determined by the complexity of the construct [28].

Usability is defined by ISO standard 9241-11 as “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in stratified context of use” [85]. These criteria are usually measured using more practical criteria [29]. In deciding on the specific items to use to measure the usability construct, validity must be taken into consideration. Validity is concerned with whether the questionnaire measures what it claims to measure [28]. One threat to validity is poorly written questions. For example, ambiguous and vague questions result in vague answers; leading questions (such as “Do you also hate this ugly mobile design?”) indicate an expected answer [28].

Along with validity, the other important issue that needs to be taken into consideration is reliability. Reliability concerns the consistency of the results of a measurement – one approach would be to administer the same questionnaire twice under the same conditions and determine how similar the results are [28]. Another approach is to calculate consistency of the measurement of a single construct, using, for example, Cronbach’s alpha [28]. A value of Cronbach’s alpha greater than 0.7 is considered a satisfactory indication of reliability [135].

Another decision that must be made is what balance should be struck between open questions (question which subjects answer by writing what they wish) and closed
questions (where the format of the answer is provided, such as where subjects must select an answer from a selection of possible answers). Closed questions are easier to answer and to analyze, while open questions require more time to answer and to process give richer information [28]. Another means of acquiring more detailed information about usability is through the use of component-based usability evaluation [29], which instead of providing information about the system as a whole, provides information about individual components – arguably more useful for identifying limitations more precisely.

In order to reduce threats to validity, the questionnaire used was adapted from a standard usability questionnaire. Some standard usability questionnaires were examined, and the IBM usability questionnaire [132] was selected for this study as it required few changes to be suitable. Items 9-13 are unchanged, items 1-8 were adapted to measure usability on a component basis, examining the skills model component, the recommendation component, and the context component, as suggested by [29]. As to the reliability of the questionnaire, the items use Likert scales, which are more reliable than single item scales [159]. Moreover, Cronbach’s alpha was calculated using SPSS to be 0.84, indicating that the questionnaire has a satisfactory level of reliability. With regard to the balance between open and closed questions, in addition to the Likert scale, each item has a section for comments so that both types of answers may be acquired.

6.2.4 Variables/Measures

The same measures used in Study 1 were used in this study to measure the effectiveness of the MCR recommendation strategy in producing the conditions of flow. The conditions of flow were measured for a task as a whole, and precision was measured as defined in Section 6.1.4.
6.2.5 Procedure

In the first session, the students are each given a username and a password. The use of Inka and the meaning of the variables are explained, and as an additional aid, this information can be consulted throughout the session by clicking Help on the user interface (see Figure 6.4). Next, each student initialises his skills model, by clicking Skills Model, clicking into each skill, and rating his perception of confidence that he can do each of a set of tasks. This done, each student clicks into Switch Task, chooses a task from the recommendation list (see Figure 6.4), goes to the task screen (see Figure 6.4), and set to work on the task.

As before, each time a student finds one or more of the key conditions of flow are absent, he creates a snapshot. These pop up on the snapshot queue on the teaching assistant’s PDA. The teaching assistant goes over to the next student on the snapshot queue to rectify the situation by providing feedback and supplying resources, as detailed in Section 5.1.5. Once the teaching assistant has attended to the student, he creates a context snapshot, which removes the student’s entry from the snapshot queue, and updates the student’s certified feedback (that is, the percentage of the task the teaching assistant certifies has been completed). When a task has been certified to be completed, the student creates a final snapshot, after which the task is removed from his working on list.

The students are not idle while waiting for the teaching assistant, as it is possible to be working on several tasks, simply switching to a different task when waiting for the teaching assistant. After two hours, the session ends and the students can take up where they left off at the start of the next session. During the last session, the subject fills in the usability questionnaire described in the materials section above (Section 6.1.3).
6.2.6 Findings

Effectiveness of the MCR Strategy

Table 6.2 shows how the precision is calculated. 177 tasks were predicted to be highly rated (that is, the task has a task score of 80 or more). Of these, 156 were highly rated, giving a precision of 88.1%. The table shows only whether the task score was less than 80 or not; Figure 6.5 details the breakdown of the scores. As in Study1, we calculate the 95% confidence interval of the mean task score; the result in this case is [80.8, 86.3]. These results offer clear evidence of the effectiveness of the MCR strategy. Note also that the tasks classified as negative, which were actually negative, point to a lack in the repository - where for particular states of the skill model, there are no highly rated tasks.
Figure 6.5: The frequencies of the task scores.
Effectiveness of the Recommendation Improvement Strategy

With regard to clear goal, the average rating for all tasks was 4.37, and the minimum average for an individual task was 3.86. As the minimum standard required is 3, no tasks with unclear goals were identified. With relation to feedback, the average rating for all tasks was 3.91, and the minimum average for an individual task was 3.25. Once again, as the minimum standard required is 3, no tasks with subpar feedback were identified. Had any tasks been identified as subpar, they would have been flagged for a content developer to examine and improve upon. With regard to the third criteria, confidence level, no task was rated enough for the confidence component of its recommendation score to be identified as poor.

Usability

Figure 6.6 illustrates the results of the usability questionnaire. The items were scored using the scale on the questionnaire in reverse, so that strongly agree was awarded 7 points, and strongly disagree awarded 1 point. This means that the average scores, given in Table 6.3, have a maximum value of 7, and the lower the score, the greater the indication of a usability issue with that item. The first two items, concerning usability of the skills model, were without issue.

The next three items (3-5) concerned the recommendation component. An issue arose here concerning the number of tasks in the working on list. It was decided to impose a limit on the number of tasks the students could work on at any one time, so that they would generally be working on highly rated tasks – if they could add as many tasks as they wanted to the working on list, they would soon have added all the highly rated tasks, and could end up working on low rated, unsuitable tasks. The comments on this item were generally about increasing the limit, for example, “Sometimes it would be nice to have a few more [tasks] available”. In one case, a particularly hard-working student expressed his frustration “can have many tasks waiting to be marked . . . and
<table>
<thead>
<tr>
<th>Item no.</th>
<th>Questionnaire item</th>
<th>Average score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Skills model easy to update</td>
<td>6.33</td>
</tr>
<tr>
<td>2</td>
<td>Skills model quick to update</td>
<td>6.50</td>
</tr>
<tr>
<td>3</td>
<td>Quick to get recommended task</td>
<td>5.58</td>
</tr>
<tr>
<td>4</td>
<td>Easy to get recommended task</td>
<td>5.67</td>
</tr>
<tr>
<td>5</td>
<td>Satisfied with number of tasks</td>
<td>4.42</td>
</tr>
<tr>
<td>6</td>
<td>Context snapshot quick to create</td>
<td>6.33</td>
</tr>
<tr>
<td>7</td>
<td>Context snapshot easy to create</td>
<td>6.25</td>
</tr>
<tr>
<td>8</td>
<td>Context snapshot not distracting</td>
<td>5.92</td>
</tr>
<tr>
<td>9</td>
<td>Overall System easy to use</td>
<td>5.92</td>
</tr>
<tr>
<td>10</td>
<td>Feel comfortable using system</td>
<td>5.33</td>
</tr>
<tr>
<td>11</td>
<td>Easy to learn system</td>
<td>6.25</td>
</tr>
<tr>
<td>12</td>
<td>System has all expected functions</td>
<td>6.00</td>
</tr>
<tr>
<td>13</td>
<td>Overall satisfied with system</td>
<td>6.00</td>
</tr>
</tbody>
</table>

Table 6.3: The average scores of the questionnaire items.

![Diagram of questionnaire item responses]

Figure 6.6: The responses to the questionnaire items.
I am unable to start anything new”. A solution to this issue could be to give certain students an increased limit.

The next three items (6-8) concerned creating context snapshots, and no issues were identified there. The final five items concerned the system overall, and the response was largely positive. Some easily remedied issues were mentioned. To improve ease of use: “when making a final snapshot, [it] would be better if the format for how hard you found the task were reversed”. Another student mentioned that he would like to be “able to see the queue during class”; this would enable students to gauge how soon the teaching assistant would be over. Another student suggested that it could have “milestones to be achieved by students at a [certain] point in time”. This could be readily achieved by creating milestones composed of certain subsets of the skills model, and a student’s current progress towards the milestone could be automatically generated from his/her skills model.

In summary, the same result as in Study 1 was observed – a number of issues related to usability were identified, but none of them were serious. That is, in each case, a solution for overcoming the issue readily suggested itself.

6.3 Study 3

The aim of this study was to show the effectiveness of the strategy for improving poor recommendations described in Section 4.3. The study focuses on poor recommendations caused by an inaccurate skills index, as this is the only one that can be improved computationally. Poor recommendations caused by other two causes (unclear goal and poor feedback) areflagged so that content developers can manually improve them.

The purpose of the strategy is to take tasks whose skills index has been identified as being the cause of poor recommendations, and to determine a better index. The ideal approach to demonstrate the effectiveness of the strategy is by means of a real
field experiment, such as Study 2, in which a flow application containing the strategy is deployed in an authentic environment.

However, a large number of users is required for this – far more than were accessible. The large number of users is needed because the strategy requires a certain amount of data before it can make the decision to switch to a different index. Users in general do only a fraction of the tasks in the repository, hence only a fraction of the users will do, and hence rate, a given poorly indexed task. Moreover, once a poorly indexed task has been identified and its index replaced, it remains to determine how effective the new index is. This requires many more users to happen to get the task recommended to them, and then rate the task.

The main alternatives to real field experiments are to use an existing dataset and to use a simulated dataset [68]. While a number of existing datasets are available, none of them are suitable. As for simulations, they avoid some constraints of real field experiments [68]. Here, for example, the challenge of obtaining long-term access to a large number of interested users is sidestepped. The main limitation of simulations is that they might not accurately model real user behaviour. Nevertheless, simulations are useful in indicating some degree of effectiveness before running field experiments, which are costly and cannot easily be repeated or altered midstream [68].

It was decided therefore to use a simulation. This section describes the variables chosen to measure the effects of the MCR recommendation strategy in improving recommendations, the procedure used, and finally, the findings of the study.

6.3.1 Variables/Measures

In order to assess the effectiveness of this strategy, the percentage accuracy is calculated by taking the ratio of good calls made by the strategy against the total number of calls made. A good call occurs when the strategy elects to replace the current skills index \( w \) with one it has calculated \( w' \), and for the next \( N \) ratings, \( w' \) proves to be better
than the skills index \( w \), where \( N \) must be sufficiently large to provide adequate proof that \( w' \) is indeed better than \( w \); here \( N = 50 \) was chosen.

Determining whether one skill index is better than another is done in the following way. Recall that the confidence component of a task’s recommendation score is defined to be successful if Equation 4.6 (on page 99) holds. Moreover, the inequality in Equation 4.6 exists for each user who did the task, and, as described in Section 4.3.2, for each task \( t \), these inequalities form a system of homogeneous linear inequalities. \( J(w) \) denotes the number of inequalities in this system satisfied by the skills index \( w \). A skills index \( w' \) is defined to be better than the skills index \( w \) if it solves more of the inequalities in the system, that is, if \( J(w') > J(w) \) holds.

### 6.3.2 Procedure

The procedure is to set up a simulation that models the real system as closely as possible. It consists of the following steps:

1. Assume we have a task \( t \) whose confidence level component will be identified as the cause of poor recommendations. This implies that the domain expert’s estimate for the index of this task \( t \) is incorrect, and we can therefore represent it by a random vector, \( w_f \). Assume that \( w_0 \), an optimal index for the task, is known.

2. Create a random skills model; this represents a random user, \( u \).

3. Given the task \( t \), predict the confidence component of the final snapshot for the user \( u \). To do this, calculate \( C_{t,u} \), an estimate of the confidence level the user \( u \) has in doing a task \( t \), using the known optimal index vector \( w_0 \), and choose the most likely one of \( c_h, c_r, c_e \) (see Equation 4.5). Use this to compute the linear inequality given in Equation 4.6

4. Add this linear inequality to the system, \( Cw > 0 \), and solve for an optimal solution \( w \) (that maximises the number of inequalities in the system are satisfied).
5. If \( w \) satisfies the selection criteria, that is, the criteria determining whether the index found should replace the current index – this is described in Section 4.3.2 and at least \( \text{least}_n \) ratings have been taken, set \( w' = w \).

6. If \( w \) does not satisfy the selection criteria, check if \( N \) iterations of this process, that is \( N \) user ratings, have been completed. If so, then no change to \( w \) is made, as no better index could be found despite a sizeable number of ratings. If \( N \) iterations have not been completed, return to step 2.

7. If a solution, \( w' \), has been found, the question that must be answered is: is \( w' \) better than \( w_0 \)? This is determined to be the case if \( J(w') > J(w) \) after each new rating is added, for the next \( N \) ratings.

In order that the results obtained represent what occurs on average, 100 trials of the above process are carried out. Furthermore, 100 trials are done for each value of \( \text{least}_n \) in the range 5 to \( N \), with the aim of informing the choice of \( \text{least}_n \) for future non-simulated studies.

6.3.3 Findings

The results of the simulation are depicted in Figure 6.7. The x-axis of the graph represents \( \text{least}_n \), that is the least number of ratings that have to be made before the strategy can change the current index. It also shows \% accuracy, which determines the percentage of good calls made by the strategy. It is important for the percentage accuracy to be as close to 100\% as possible, because otherwise there is a risk that the strategy will replace an index with an inferior index.

It was expected that if only a few ratings were available, the percentage accuracy of the strategy would be low. This proved to be the case, for example, when \( \text{least}_n \) was set to 5, that is, five users have rated the task, the simulation showed that the
**Figure 6.7**: The % *accuracy* demonstrated in the simulation. The x-axis of the graph represents the least number of ratings required before the strategy is permitted to change the current index.

*Percentage accuracy* was just 60%. However, for a strategy to be successful, it should demonstrate a high percentage accuracy once it has sufficient data.

The strategy under investigation demonstrated this. By repeatedly simulating the same situation but each time incrementing *least_n*, we found that *percentage accuracy* increases to 100%. Moreover, the simulation suggested a value of the least number of ratings it should have before making a decision. In order for *percentage accuracy* to be close to 100%, the simulation indicated that the strategy should have at least 26 ratings for a task before a changing its index.

A limitation of this evaluation is that it uses simulated users, whose behaviour may differ from real users, thus impinging on the result. However, the assumption made in step 2 of the procedure – that the predicted snapshot is the most likely one, is a reasonable assumption to make. For the result to be considerably different, a significant proportion of the real users would have to make snapshots that are not the most likely
one of them to make. This would suggest that a significant proportion of the users have inaccurate skill models. This is certainly a possible issue, and it is considered in further detail in Section 8.3.2.

In summary, the simulation demonstrated the effectiveness of the strategy in producing better indexes for tasks, and moreover indicated a value for the choice of least $n$ for future non-simulated studies.

6.4 Conclusion

This thesis investigates the question of how tasks can be recommended so that the key conditions of flow are produced. This chapter focused on evaluating the effectiveness of the MCR strategy and on the strategy of recommendation improvement, both of which were described in Chapter 4. This was achieved by means of three studies, which addressed a number of shortcomings of the pilot studies.

The aims of the first study were to evaluate the effectiveness of the MCR strategy and to identify any usability problems with the flow application used in the study. This study addressed a number of shortcomings of the pilot studies: it had a larger sample size than the pilot studies; it used a different measure (measuring flow of a task as a whole) so that subjects’ omission to create snapshots was no longer an issue; and it evaluated usability – this is an important aspect of the research question since producing effective recommendations will be of limited value if the user is not satisfied with how the recommendations are produced.

The precision of the MCR strategy was calculated as 100%, and the mean task score calculated, with 95% confidence, to be in the range $[91.5, 95.25]$, well above the minimum of 80 (the minimum requirement for the three key conditions of flow to be present). These results certainly supply evidence of the effectiveness of the MCR recommendation strategy. However, it is important to observe that these results are
limited, since despite the time-consuming nature of the gathering of this data, it is still a relatively small dataset (just 25 tasks were rated). In order to obtain a stronger result, more data is essential. With regard usability, a number of issues were identified, but none of the serious nature. That is, in each case, a solution for overcoming the issue readily suggested itself.

The second study had three aims. The first two aims of the study were identical to those of Study 1, that is, to measure the effectiveness of the MCR task recommendation strategy in producing the conditions of flow, and to identify any usability problems with the flow application used. The third aim was to evaluate the strategy for identifying and improving poor recommendations. This study addressed two shortcomings of Study 1. Firstly, rather than getting students to take part in sessions individually, an entire class group (comprising 12 students) took part in sessions together – the teaching scenario for which the flow application was designed. Secondly, the study enabled considerably more data to be gathered since it took place in classes over 10 weeks of a single semester course, compared with the two 90 minute sessions used in Study 1.

The precision of the MCR strategy was calculated as 88.1%, and the 95% confidence interval of the mean task score as [80.8, 86.3]. These results offer clear evidence of the effectiveness of the MCR strategy, and although not as high as in Study 1, they are stronger results since they are based on a larger dataset (189 tasks). With regard to usability a number of issues related were identified, but none of them were serious. That is, in each case, a solution for overcoming the issue readily suggested itself.

With regard to identifying and improving poor recommendations, no tasks were identified as having insufficiently clear goals or inadequate feedback. The minimum rating for each of these conditions of flow to be present is 3. The average clear goal rating for all tasks was 4.37, and the minimum average for an individual task was 3.86. The average feedback rating for all tasks was 3.91, and the minimum average for an individual task was 3.25. Had any tasks been identified as subpar, they would have
been flagged for a content developer to examine and improve upon. With regard to the third criteria, confidence level, no task was rated enough for the confidence component of its recommendation score to be identified as poor.

The aim of the third study was to evaluate the strategy for improving poor recommendations caused by an inaccurate skills index – this is the only one of the three causes that can be improved computationally. Poor recommendations caused by other two causes (unclear goal and poor feedback) are flagged so that content developers can manually improve them. The ideal approach to demonstrate the effectiveness of the strategy is by means of a real field experiment, such as Study 2, in which a flow application containing the strategy is deployed in an authentic environment. However, this was impossible as it would have required substantially more users than were available. This problem was sidestepped in the third study by simulating the users.

It was expected that if only a few ratings were available, the percentage accuracy of the strategy would be low. This proved to be the case, for example, when five users have rated the task, the simulation showed that the percentage accuracy was just 60%. However, for a strategy to be successful, it should demonstrate high percentage accuracy once it has sufficient data, and this also proved to be the case. By repeatedly simulating the same situation but each time incrementing the least number of ratings required, we found that percentage accuracy increases to 100%. Moreover, the simulation suggested that the value of the least number of ratings the strategy should have before making a decision in order for percentage accuracy to be close to 100%, should be at least 26 ratings for a given task.

The main limitation of simulations is that they might not accurately model real user behaviour. Nevertheless, simulations are useful in indicating some degree of effectiveness before running field experiments, which are costly and cannot easily be repeated or altered midstream [68]. Moreover, field experiments can be used to inform future simulations, making them increasingly accurate.
Chapter 7

A Framework for Flow Applications

A software framework is defined as “the skeleton of an application that can be customised by an application developer” [107]. It is also commonly defined as “a set of cooperating classes that make up a reusable design for a specific class of software” [88]. Frameworks are needed because they can reduce the cost of developing applications from a specific class of software by “an order of magnitude” by enabling both design and code to be reused [181], they can increase modularity, enhance reusability and extensibility, and improve the quality of software [79].

In order to enable developers of flow applications to readily use the results of this thesis, a framework for flow applications was developed. The key features of a framework for flow applications are that it can recommend suitable tasks, improve recommendations, provide feedback, and measure flow. No framework for developing flow applications exists at present. The closest efforts are not at all close. For example, there are several recommendation frameworks that recommend learning objects, such as [130] but they do not have the aim of creating the conditions of flow. Recommendation frameworks that recommend generic items are not relevant since, as has been argued in this thesis, recommending tasks is quite different. Work that provides — to some degree — some of the features, such as the Personalised Recommender System
and ELM-ART are not frameworks, and thus do not confer the benefits, in particular the rapid development of flow applications.

7.1 Overview of the Framework

This section gives an overview of the framework. It begins describing the approach taken to develop a framework, and defines the framework requirements. This is followed by an overview of the framework components. Next, the task recommendation, improving recommendations, feedback, and measuring flow features available in the framework are described along with a brief description of their implementation. Following this, the elements of the activity model used in the framework are specified, followed by a description of how context is acquired, modelled, and managed. Finally, a means of creating a flow application using the framework is given, along with a description of each of the properties that can be set in the framework, and an account of persistence in the framework.

7.1.1 Approach

A software framework is designed to help develop applications from a particular class of applications. The software framework described in this thesis was developed using the “Three Examples” approach to framework development [181]. In this approach, three examples from this class of applications are developed, and the framework develops from the abstractions that are reused by the applications. A framework arrived at using this approach is called a Harvested Framework [4]. The alternative is a Foundation Framework, which is built directly, that is, without first developing any applications. While the Foundation Framework approach seems more efficient, it almost always fails in practice [4, 181]. It is only by building applications that the abstractions that are actually being reused can be determined [181]. It often transpires that the capabilities
of a Foundation Framework are not what that the applications actually require [4]. This is due to the extreme difficulty in generalising without any concrete examples.

Once the first application has been built, code is available for reuse. When developing the second application, any code that replicates code from the first application is factored out into a common area: the framework area. Each subsequent application that is developed refines the framework area. The more applications that are built, the more general the framework becomes [181]. However, developing applications is a lengthy process. Therefore, in order to produce a framework in a reasonable amount of time, a limit must be imposed on the number of applications developed; the “Three Examples” approach maintains that this limit should be three.

7.1.2 Framework Requirements

The requirements for the application framework for flow applications were obtained by generalizing the requirements of three prototype flow applications: Inka (see Section 5.1.3), Musika (see Section 5.2.1), and Inka 2 (see Section 6.1.3). In this way they are no longer specific to a particular activity, but appropriate for any activity. The application framework:

- must influence (that is, support the creation of) the three key conditions of flow. This requirement is composed of two sub requirements. The application framework:
  - must recommend tasks since to go into flow one needs, at any moment, to be engaged in a task. Moreover, these tasks must have clear goals, and must be chosen so that the challenges of the task (as the user perceives them) are balanced with the user’s skills (as he perceives them).
  - must supply or enhance feedback from the recommended tasks, since receiving feedback is another of the conditions of flow.
• must measure whether the user is in flow, so that:
  – when a user is not in flow, action can be taken to aid its return.
  – subsequent users in similar situations should be more likely to experience flow.

• should increase its ability to assist its users to experience flow as it is used.

• must be mobile since it must be possible to use the application in the environment in which the activity takes place. If, for example, the activity is rock climbing, it must be possible to use the application at a climbing wall or a rock formation.

• must be reusable and extensible so that application developers can use its approaches to facilitate the development of applications for a diverse range of activities.

7.1.3 Framework Components

The application framework comprises 44 classes and a property file. This section supplies a high-level overview of the implementation of the application framework by outlining the structure of the framework and the responsibilities of each of its components. Figure 7.1 depicts the components of the framework, along with the core classes that make up these components. The framework is composed of nine components:

1. Recommendation component. This is responsible for recommending suitable tasks to a user (that is, tasks whose challenges match the perceived skills of the user, whose goals are clear and which provide the user with feedback). The framework provides three strategies for recommending suitable tasks: the Stereotype strategy (this behaviour is contained in two classes: StereotypeStrategy and MasteryCriteria), the Multi-Attribute Utility Theory strategy (this behaviour
Figure 7.1: A class diagram depicting a high-level overview of the application framework
is contained in **MAUTStrategy**), and the Multi-Criteria Recommender strategy (this behaviour is contained in **MCRStrategy**).

The recommendation component uses the Strategy pattern [88], in which a family of related algorithms or strategies is defined in separate classes, and then selected by the client at runtime. The Strategy pattern facilitates this selection without requiring any changes to the client (in this case, **RecommendationManager**). It also enables any number of strategies to be defined. The only condition that must be satisfied in order for a concrete strategy to be used by the client is that it must implement the behaviour defined in the strategy interface (in this case **RecommendationStrategy**).

The recommendation component is also responsible for improving recommendations by identifying poor recommendations and taking action so that the recommendation strategy will provide better recommendations in future. The behaviour that achieves this for the MCR strategy is contained in the **ImproveMCRStrategy**, **ImproveMCRStrategyResult**, and **HomogeneousLinearInequalities** classes.

2. **Activity component.** This component has three main responsibilities:

   (a) To provide a representation of an activity. The components of this model of an activity are represented as the classes: **Task**, **Artefact**, **Resource**, **Skill**, **Session**, **Project**, and **Unit**.

   (b) To enable tasks, resources, projects, skills, and relationships between them to be added, removed, or edited. This is handled by the **ContentManager**.

   (c) To provide a way of accessing and editing a user’s current session. This behaviour is contained in **SessionManager**.

3. **Context component.** This component has three chief responsibilities:
(a) To acquire context, which is represented as subclasses of the abstract class Context. The framework provides several such subclasses: Flow3K, SkillConfidence, PercentComplete, and Time. The Context component uses the Composite pattern to hold the context objects. Using this design pattern, context objects are stored in a tree structure, and individual objects and compositions of objects are treated uniformly. In addition, a type of context can be updated manually if the class representing it implements the ManualUpdateContext and UpdateContextForm interfaces.

(b) To store current context and to allow clients to obtain context either synchronously or asynchronously. Context can be requested directly from the Context component via the ContextManager class, which uses the Visitor pattern, implemented with the GetContextVisitor, to visit each context object. The Visitor pattern allows an operation to be carried out on all elements of a structure (in this case the composite of context objects); an advantage of this design pattern is its flexibility: new operations may be added without requiring changes to the classes comprising the structure (in this case, the context objects).

Another way clients may access context is by subscribing to a particular context type. The Context component uses the Observer pattern to accomplish this. The Observer pattern defines a relationship between a subject object and a set of objects that observe the subject, so that when the subject changes state, its observer objects are automatically notified and updated. An advantage of using this pattern is that it allows related objects to be kept up to date, without making the classes tightly coupled. In the current implementation, the subjects are concrete subclasses of Context, and the observers, which must implement the interface ContextChangedEventListener, are any classes interested in the context, for example, classes
representing feedback (which are contained in the Feedback component, described below).

(c) To allow clients to search context history (that is, previous states of particular contexts). There are many purposes for analysing and reasoning over context history. This behaviour is contained in ContextManager.

4. Feedback component. This component is responsible for providing feedback, letting users know how well they are doing with the task at hand. Feedback comes in a wide range of varieties, and since it is information, it will usually be in visual, auditory, or haptic form. Each different type of feedback is represented as a different subclass of FeedbackOutput; the types of feedback available in the framework are BarGraph, FlowView, Tone and Text.

Each feedback type is associated with a set of context types, and is updated when one of the set of context objects is updated, using the Observer pattern. The Bridge pattern [88] is used to enable the feedback objects to be used with different output systems (for example, for a visual feedback Sun’s Abstract Window Toolkit (AWT) and Eclipse’s Standard Widget Toolkit (SWT) [74] are two examples of output systems). The purpose of this pattern is to enable an abstraction to change independently of its implementation. An example of this in the Feedback component is that by choosing different implementations of the interface BarGraphImplementor, a bar graph can be displayed in SWT or AWT. Furthermore, the Bridge pattern makes it easy to add an output system which is not already included in the framework.

5. Repository component. This component is responsible for storing and retrieving objects. Examples of objects that need persistence in the framework include tasks, resources, and context. Any implementation of the Repository interface can be used; the framework comes with an SQL implementation, SQLReposi-
6. **User Management component.** This component is responsible for adding and editing users. This behaviour is contained in the `UserManager` class. This component also enables a user to edit his skill model, which is represented by the `SkillModel` class.

7. **User Interface component.** This is responsible for providing a basic means of doing the following standard tasks: viewing or editing a session, viewing a task in a session, viewing a resource, and creating a context snapshot. This behaviour is contained in the `UI` class, and the concrete subclasses of `Screen`.

8. **Communication component.** This component is responsible for communicating messages between components on different devices. This behaviour is largely contained in the `CommunicationManager` class. It is also responsible for communication via the serial port, which is one way of connecting sensors to a device. This behaviour is implemented by the `SerialPort` class.

9. **Utilities component.** This component provides utilities required by the other components of the framework. These utilities include performing t-tests, cloning objects, computing unit vectors, starting processes in different operating systems, and various string manipulations.

### 7.1.4 Task Recommendation

The task recommendation problem may be summarised as follows: recommend tasks that have clear goals, whose challenges (as the user perceives them) are balanced with the user’s skills (as he perceives them), and which supply or enhance feedback. The task recommendation process that supplies the most suitable tasks to a user is invoked when a session is being designed. Figure 7.2 depicts a sequence diagram showing a
Figure 7.2: A high level view of the recommendation process

high level view of the recommendation process. Regardless of the strategy chosen, the
process begins with the initialisation of the RecommendationManager and ends by
returning a set of suitable tasks.

The three approaches to task recommendation described in the thesis (Stereotype,
MAUT, and MCR) are available in the framework. An application developer can choose
the most appropriate approach for the particular application being developed. This
decision can be made by considering the benefits and limitations of each of the three
approaches. For each approach, a brief overview of the approach is given, along with
its principal benefits and limitations (other benefits and limitations may be found in
Section 4.4); this is followed a brief description of its implementation.

Stereotype Strategy

In the Stereotype approach, the skills of each user are characterised by a stereotype.
Users switch stereotypes when they have mastered the skills associated with a particu-
lar stereotype. The Stereotype strategy is suitable for activities in which tasks can be readily ordered by difficulties, such as relaxation (see Section 7.2.1) or juggling [90]. Its principal benefits are that tasks can be readily added by a domain expert (as long as in the given activity it is straightforward to order tasks by difficulty), and that it facilitates an increase or decrease in difficulty of the task is recommended to the user. Its principal limitations are its limited set of possible tasks a user can choose (and the user cannot easily add to this set), and that a user may have to do many prerequisite tasks, to prove to the system that he has the skills he already knows he has.

The sequence diagram in Figure 7.3 illustrates the classes involved in the task recommendation process using the Stereotype strategy. Task recommendations are requested from the RecommendationManager using the method getRecommendedTasks. The key steps are: obtain the full list of requirements for the activity (getAvailableRequirements), acquire the prerequisites of each (getPrerequisites), examine the prerequisites to see if they have been mastered (isMastered); this acquires the mastery criteria, and determines if the user has mastered the requirement by considering his context history (usually whether he has completed certain tasks).

Note that StereotypeTask, an extension of the standard Task, is required, since information specific to the Stereotype strategy is required; for example, isDefault, which distinguishes default from non-default tasks. The other methods: getLastDoneTask, isTooEasy, isTaskSetAside, and isCompleted, examine a user’s context history to determine their result. For example, getLastDoneTask, examines a user’s context history to determine the last task of a requirement the user completed. The remainder of the method getRecommendedTasks combines the information gathered to select the most suitable task from each of these available requirements as set out in the activity diagram depicted in Figure 4.2 (on page 86).
Figure 7.3: A sequence diagram showing the key classes and methods involved in the Stereotype recommendation strategy.
MAUT Strategy

The MAUT approach differs considerably from the Stereotype approach. In the MAUT approach, a user rates relevant attributes of tasks, and the relative importance of these is used to calculate an overall utility value for each task; tasks with utility values above a certain threshold are likely to produce the conditions of flow. The MAUT approach is suitable for activities in which users spontaneously come up with ideas for tasks, such as thinking of a song they’d like to perform (see Section 5.2.1) or an office worker might add work tasks to do as they arise. The principal benefits of this approach are its potential to give the user much greater choice; this can be further increased by the user who can readily add new tasks. In addition, unlike the Stereotype approach, a user does not have to do any tasks to prove to the system that he has certain skills. The principal shortcomings of this approach are that as the user’s skills increase, he will have to re-rate the tasks, which would be an issue in systems with a large number of tasks, and that tasks that haven’t been rated by the user cannot be recommended.

Task recommendations are requested from the RecommendationManager using the method getRecommendedTasks, which invokes the method getRecommendedTasks of MAUTStrategy. This initialises the relative importance values acquired from the application properties file, obtains all the tasks in the repository, and obtains the flow context for each task (getLastContextSnapshot); this represents the user’s estimate of the state of the conditions of flow if he engaged in the given task. The flow context for each task is used to calculate the MAUT value using an implementation of the additive aggregation function given in Equation 4.1 on page 87. Each task’s suitability value is set as this MAUT value and the tasks are sorted in order of suitability using TaskComparator.

MCR Strategy

In the multi-criteria recommender (MCR) approach, each task is indexed by a set of
skills, and a rating for it is estimated using the user’s perception of his confidence doing these skills, along with ratings for clear goal and feedback from other users who did the task. The MCR approach is suitable for activities which have been or can be decomposed into a specific set of skills, such as yoga (see Section 7.2.2). The principal advantages of this approach are its potential for greater choice, that unseen tasks can be recommended to a user, and that difficulty can be increased or decreased by the user by reappraising his skills. The principal drawback of this approach is that indexing a task can be time-consuming, and not even a domain expert can guarantee its accuracy.

Figure 7.4 shows a sequence diagram that illustrates the principal classes involved in the task recommendation process using the MCR strategy. As with the other strategies, task recommendations are requested from the RecommendationManager using the method getRecommendedTasks, but this time the method getRecommendedTasks of MCRStrategy is invoked. All candidate tasks are obtained (getAllTasks), and an implementation of the collaborative filtering formula given in Equation 4.3 on page 92 is used to estimate the clear goal rating (getClearGoalComponent) and feedback rating (getFeedbackComponent) of the candidate tasks.

The current level of skills a user believes he has are stored in a skill model (the class SkillModel is an implementation of this). The skill model is created by the method getSkillModel, which obtains all the tasks used to estimate the skill using the method getTasksEstimatingSkill, acquires the user’s most recent confidence level for each task by consulting the ContextManager, and updates the SkillModel accordingly. The confidence component of the overall score is found by getConfidenceComponent; this uses the method $C_{t,u}$, which implements the formula for $C_{t,u}$ given in Equation 4.2 on page 91, and a measure of how well the challenges and skills are matched is obtained by calculating $P(C_{t,u} = 3$ or $4)$. Finally, the three components used to make up the suitability score are combined (aggregate) and the tasks are sorted in order of suitability using TaskComparator.
Figure 7.4: A sequence diagram showing the key classes and methods involved in the MCR recommendation strategy.
7.1.5 Improving the Recommendations

If users’ actions do not cause the system to change, then the same poor recommendations will be supplied to the subsequent users time and time again. As poor recommendations lead directly to the absence of the conditions of flow, addressed this problem is paramount. A poor recommendation can result from any of the three conditions of flow: unclear goal, inadequate feedback, or mismatching skills and challenges. An approach that can continuously improve recommendations by modifying items and/or items’ metadata based on items identified using user ratings was described in Section 4.3. Changing the items themselves is suitable for the clear goal and the feedback attributes, and changing the items’ metadata is suitable for matching skills and challenges. For each of these approaches, a brief overview of the approach is given, along with its principal benefits and limitations (other benefits and limitations may be found in Section 4.4); this is followed a brief description of its implementation.

Clear Goal and Feedback

Tasks that fall below certain criteria are identified (that is, tasks with unclear goal or substandard feedback) and flagged. The main benefit of the approach is that it gives content developers the opportunity to improve upon and re-release the items, rather than simply allowing items to fall out of use. The main drawback of the approach is that it requires user effort (users must provide an estimation of the attributes for each task they do). Identifying tasks with an unclear goal (isClearGoalSubpar) and substandard feedback (isFeedbackSubpar) is achieved by obtaining the estimation by each user of the clearness of the goal or quality of feedback of the given task using the ContextManager, and collecting this set of ratings into a sample. This sample, along with the chosen minimum standard for how clear the goal must be, and alpha, the level of significance, are taken by the method isSignificantRightTail of the class TTest, which calculates if the clearness of the goal or quality of feedback is below the
If it is the case that the task’s goal is not below standard, but the poor recommendation was the result of an inaccurate skill indexing (and thus the challenges of the task do not match the user’s perceived skills), then an index that produces a more accurate result is sought. The main advantage of this approach is that it is automatic, that is, it does not require the input of a domain expert but only user driven context snapshots. Its main limitation is the number of users who must do a task before it can produce a more accurate index with a low risk of error; this is unavoidable since sufficient data must be collected, and an alternative approach would necessarily have the same limitation.

Tasks whose challenges mismatched with the skills of the user are identified using \texttt{isConfidenceSubpar}, which acquires each user’s final context snapshot for a given task from the \texttt{ContextManager}, and the user’s actual confidence level is acquired from this, and compares it with the predicted the final snapshot (the most likely outcome using the methods \texttt{getTooHard()}, \texttt{getRight()}, and \texttt{getTooEasy()}). The difference between the predicted value and the actual value is then added to the sample, and the method \texttt{isSignificant} uses the t-distribution to determine whether the sample comes from a population with mean 0, and choosing a larger alpha make the acceptance level more lenient. If the method returns true, it is because there is a significant difference between the predicted confidence and actual confidence. That is, the task was responsible for many poor recommendations.

For each task identified, a candidate for the index is calculated as described in Section 4.3.2 and a decision is made about whether to use it in place of the existing index. The method \texttt{takeAction} creates the matrix $A$ by invoking the method \texttt{createA}, which obtains the skill model for each user who did the task at hand, and calculates $c(s_i, u)$ for each skill $s_i$ in the skill model by invoking the method $c$ of the \texttt{SkillModel}
class. The matrix $A$ is then used to instantiate $\text{HLI}$, which represents a system of homogeneous linear inequalities. The method $\text{solve}$ produces a solution to this system using the Warmack-Gonzalez algorithm. Finally, the method $\text{replace}$ is called and it evokes an implementation of the abstract method $\text{criteriaMet}$, an example of which is to use the Binomial distribution and place a restriction on $\text{lowest}_n$, as described in Section 4.3.2. If the criterion is met, the method $\text{replace}$ replaces the old $w$ with the $w$ just calculated.

### 7.1.6 Feedback

Some feedback is available in the environment for anyone to see. For example, anyone watching a game of tennis can see if the ball went in. Some feedback is available only to those with the required training. For example, a chess grandmaster can evaluate his position and thus determine how well he is doing, whereas someone with little or no training in chess will be able to make little sense of it. Some feedback is available to none but the most exceptional of people. For example, few people could quantify how relaxed they are to any degree of precision. It is the last two types of feedback that this framework focuses on.

Feedback is information, and thus it is usually in visual, auditory, or haptic form. An example of visual feedback is an object changing colour, such as an ambient orb [1]. An example of auditory feedback is a tone delivered in a specific rhythm. An example of haptic feedback is force feedback; this can be delivered to a user via a device such as the CyberGrasp™ system [3], which supplies force feedback to a user’s hand and fingers.

Feedback does not have a fixed meaning. For instance, a tone delivered in a specific rhythm could be used in an application to represent how close the user is to something, where the rhythm gets faster the closer the user gets to the goal object. In another application, it could represent the beating of the user’s heart. This property of feedback
Figure 7.5: A class diagram showing the framework classes involved in supplying feedback.

makes it ideal for reuse.

Supplying feedback is equivalent to representing a context type visually, auditorially, or haptically. Figure 7.5 contains a class diagram depicting the framework classes involved in supplying feedback. Feedback is represented in the framework as a class that implements the interface of FeedbackOutput, and each FeedbackOutput object observes a collection of one or more Context objects, and is updated accordingly. For example, in Musika (see Section 5.2.1), the FeedbackOutput object FrequencyComparisonView provides a graphical representation of how well the user is doing by observing two context objects (required frequency and actual frequency).

The FeedbackManager class initialises, stores, and enables access to current feedback objects. FeedbackOutput objects are designed to be as flexible as possible, and
thus they have many properties that can be set when they are initialised. These are obtained from the application properties file and include `feedback.type` (the type of the feedback, for example, a bar graph) and `feedback.contextType` (the type of context that the `FeedbackOutput` object observes). A complete list of properties can be found in Table 7.4 on page 197.

### 7.1.7 Measuring Flow

It is important for a flow application to measure flow so that it can take action when it becomes aware that a user is not in flow. The Experience Sampling Method (ESM), described in Section 2.1.4, is the most suitable candidate for measuring flow since it allows flow to be measured as an activity is happening. The framework takes an approach based on the ESM. Modifications were made to the ESM approach in order to reduce the impact of its limitations.

Firstly, only two of the 30 measurement points of the ESF (the form that needs to be filled in as part of the ESM), are necessary for measuring flow. By discarding the others, measurement of flow takes considerably less time. Secondly, the method of measuring skills and challenges in the ESM is ambiguous and may provide unreliable measurements [77]. To avoid these problems, the challenge/skill ratio is measured indirectly, by measuring a person’s confidence level of completing the task at hand. Thirdly, the ESM uses only one of the three key conditions of flow to predict flow experience. Measuring all three of the key conditions provides stronger evidence of flow. More importantly, this information can be used by a flow application to improve the clearness of goals and the feedback for the current user or subsequent users. For these reasons, the variables confidence level, clear goal, and feedback were included in the form.

In order to measure whether a user is in flow at a particular moment in time, the user must make a context snapshot by filling in a form similar to the one shown in Figure
5.8 (on page 121), except with just the first three items (clear goal, confidence, and feedback). Six conditions were originally included because it wasn’t clear in the earlier publications on the flow model (such as [38]) which elements of flow are characteristics of flow, and which are conditions of flow. This was clarified in later publications (such as [36]), in which the elements were explicitly divided into conditions and characteristics. Thus, in the framework, the extraneous conditions are not measured; only the three key conditions of flow are measured.

The options available on the form are explained in Table 7.1. The `isInFlow` method in the `Flow` class calculates whether the user is in flow, by determining if the conditions are present. Specifically, a user is deemed to be in flow if the variable clear goal has a value greater than 2 (representing that the goal is at the least somewhat clear), the variable feedback has a value greater than 2 (representing that the user has at least some idea of how he is doing), and the confidence level variable has a value of 3 or 4 (representing the situation in which the user thinks he stands a chance of succeeding – that is, that the task is not too easy nor too hard).

Figure 7.6 depicts a class diagram showing the framework classes involved in measuring flow. The abstract class `Flow` extends the abstract class `Context`. The concrete class `Flow3K` extends the abstract class `Flow`, and represents the model of flow used in the framework – the “3K” in the title refers to the three key conditions of flow. The class `Flow3K` implements `ManualUpdateContext`, which is required for flow to be updated manually, using the form represented by the class `UpdateFlow3KSWT`, which contains the behaviour to display the form using SWT, and to update the flow context from the form. The `ContextManager` class (shown in Figure 7.1) is responsible for storing and retrieving flow data.

The investigation made into measuring flow was done largely during the development of Inka, and more details of the approach for measuring flow in the framework can be found in Section 5.1.3.
Figure 7.6: A class diagram showing the framework classes involved in measuring flow.
<table>
<thead>
<tr>
<th>variable</th>
<th>question and possible answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear goal</td>
<td>How clear is the goal?</td>
</tr>
<tr>
<td></td>
<td>1. very unclear</td>
</tr>
<tr>
<td></td>
<td>2. unclear</td>
</tr>
<tr>
<td></td>
<td>3. somewhat clear</td>
</tr>
<tr>
<td></td>
<td>4. quite clear</td>
</tr>
<tr>
<td></td>
<td>5. very clear</td>
</tr>
<tr>
<td>confidence</td>
<td>How confident are you that you will succeed with the task (that is</td>
</tr>
<tr>
<td></td>
<td>by yourself without any help)?</td>
</tr>
<tr>
<td></td>
<td>1. definitely won’t succeed</td>
</tr>
<tr>
<td></td>
<td>2. more than likely won’t succeed</td>
</tr>
<tr>
<td></td>
<td>3. might succeed / stand a chance at succeeding</td>
</tr>
<tr>
<td></td>
<td>4. probably will succeed</td>
</tr>
<tr>
<td></td>
<td>5. definitely will succeed</td>
</tr>
<tr>
<td>feedback</td>
<td>Do you know how well you are doing right now with this task?</td>
</tr>
<tr>
<td></td>
<td>1. not at all</td>
</tr>
<tr>
<td></td>
<td>2. not really</td>
</tr>
<tr>
<td></td>
<td>3. some idea</td>
</tr>
<tr>
<td></td>
<td>4. good idea</td>
</tr>
<tr>
<td></td>
<td>5. know exactly how well I’m doing</td>
</tr>
</tbody>
</table>

Table 7.1: The options for the form to measure flow
### 7.1.8 Specification of Activity Elements

The elements of an activity (or, more precisely, the elements of the activity model used in this thesis) comprise: task, artefact, session, skill, project, and unit. Figure 7.7 depicts a class diagram showing the relationships between the activity elements and the key behaviour and key attributes of each activity element. Implementing the concrete classes that represent the elements of an activity requires explicitly stating both the attributes and behaviour of the elements.

All of the activity elements are subclasses of the abstract class `ActivityElement`. A user, at any moment, is engaged in a task, which is the principal activity element, and is represented by the class `Task`. A task can have any number of subtasks (each of which can also have any number of subtasks, since they are themselves tasks). The method `getSuitable` provides the behaviour that determines how suitable the task is for a user. The task can have a collection of artefacts, that is, items that can be used to help a user to do a task. Artefacts are represented by the `Artefact` class, and it has two subtypes, `Resource`, which represent any file that can be played or displayed, and `Tool`, which represents a physical item. A user’s current session (represented by the class `Session`) comprises a sequence of tasks containing the tasks he has done, his current task, and the tasks he plans to do in the current session.

Skills are represented by the `Skill` class. A user’s level of confidence that he can do a skill is of particular importance; it is modelled by a random variable and is represented by the class `SkillConfidence`. The method `getProbability(int x)` allows the probability that the random variable has the value `x` to be calculated. The confidence level of the skill is estimated from a set of related tasks, which can be acquired by the method `getUsedToEstimate`, and the relative importance of each task can be acquired by the method `getWeightsUsedToEstimate`. The complete set of skills required for the activity at hand and the relationships between them is encapsulated by the `SkillModel` class.
Figure 7.7: A class diagram showing the relationships between the activity elements and the key behaviour and key attributes of each activity element.
The length of time required to do a task is generally a number of minutes or hours. Larger goals can be encapsulated by a project (represented by the class `Project`), which contains a task tree, which is a sequence of tasks, and any task in the task tree can have a subtask. To complete a project, all the tasks in the task tree must be completed. The `addTask`, `removeTask`, and `moveTask` methods enable any desired change to the project. A unit is a specialised type of project, the goal of which is to master a specified set of skills. It contains a sequence of tasks, which the user can do in order to master the specified set of skills. It is represented by the `Unit` class, which contains the behaviour to navigate the structure of tasks in the unit, to obtain the units that must be mastered before the current unit should be attempted, and to test if the current unit has been mastered.

### 7.1.9 Context Acquisition, Modelling, and Management

In the framework, context is modelled using an object-oriented approach (the approach taken in the Java Context Awareness Framework (JCAF) [18] and the Hydrogen project [101]). Some context classes available in the framework are shown in Figure 7.8. Context is acquired using sensors, although as described in [15], sensors cover not just physical sensors, but also virtual sensors and logical sensors. Some authors, such as Lieberman and Selker [133], claim that if information is acquired explicitly, that is, through a user interface, then this information is not context. However, this definition of context is limiting.

While it is clearly preferable to acquire information automatically, it is not possible to acquire some kinds of information automatically, at least at present. It does not seem rational to classify information as context if no means to measure it automatically exists, but then to classify it as context as soon as a means becomes available. Manually updating context has been used in applications before, for example in GUIDE [46], which used it to update a user’s current location whenever it could not be updated.
Acquiring context manually can be considered to be a virtual sensor. The context can be inputted by the user who will benefit from it, or from a different user, such as in Inka, where the tutor supplies the task progress context, but it is the student who benefits from it. Of course, manually updating context is only feasible where the context does not need to be updated very frequently, since then the value the context-aware application provides would be outweighed by the inconvenience of constantly having to manually update the context.

In the framework, context can be extended so that it can be updated manually. If a context type can already be updated automatically, this extension means that it can be updated either manually or automatically. Context is manually updated by means of a form designed specially for that context type. The context type given as an example in Figure 7.9 is Flow3K, which represents flow measured by measuring the presence of the three key conditions of flow. Once the form has been filled in, the
updated context is stored by the **ContextManager**.

In addition to storing context, the **ContextManager** is also responsible for allowing clients to obtain context either synchronously or asynchronously. With synchronous access, the client polls the **ContextManager** and requests a particular type of context and the **ContextManager** returns it. With asynchronous access, the client subscribes to a particular context type and the **ContextManager** supplies that context every time the context changes. What constitutes a change can be defined so that small, inconsequential changes do not result in the client being sent the updated context. For example, in the application developed for the domain of yoga (see Section 7.2.2), it is unimportant if a person’s body position changes unless it is outside a predefined limit.
7.1.10 Skeleton Application and the Properties File

A software framework is defined as “the skeleton of an application that can be customised by an application developer” [107]. In order to use the framework described in this thesis, that is, to turn the “skeleton of an application” into an application, an application developer must proceed as follows. Firstly, he must develop any additional classes he requires, that is, classes which implement behaviour that the framework does not provide. Secondly, he must edit the properties file, in order to customise the application. Finally, he must invoke the method initialize from the Main class. This method takes one parameter – the name of the properties file, which allows the application developer to use a different properties file. The method initialize then initialises all the components of the framework, and thereby sets the new application in motion.

The properties file contains a number of different properties necessary for reuse and extension. Tables 7.2, 7.3, and 7.4 give the names of each property that can be set, a description of its purpose, the values it can have, and an example from one of the flow applications described in this thesis.

7.1.11 Persistence

Flow applications require persistence of many objects, including Context objects and objects relating to the activity (Task, Skill, Session, and Resource). Persistence is required for many purposes, such as task recommendation and allowing clients to search context history. The repository component is responsible for persistence and also for searching for and retrieving particular objects. The repository component contains an interface Repository, and implementation of which is provided in the framework. This implementation, SQLRepository, uses an SQL database to store the objects. There are many relational database management systems (RDBMS) that use SQL,
including several that allowed the database to be stored on the mobile device, such as SQLite [195]. Because the framework uses the Repository interface, it is possible for developers to provide other implementations of Repository that use a different means of storage (for example db4o [198] or XML).

### 7.2 Framework Reusability and Extensibility

A framework can reduce the cost of developing applications from a specific class of software [181]. To achieve this, framework components need to be designed so that they can be reused in many new applications [79]. In addition, because “applications seem infinitely variable”, a framework cannot contain all the required classes for any application in a specific class of software [107]. Therefore, it is more important for a framework to be easy to extend than for it to have all the features required for an application [107]. Hence, two important characteristics of a framework are its reusability and its extensibility.
<table>
<thead>
<tr>
<th>Property name</th>
<th>Property description</th>
<th>Value space</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>flow</td>
<td>The name of the class used to represent flow.</td>
<td>String</td>
<td>Flow3K</td>
</tr>
<tr>
<td>ClearGoal_standard</td>
<td>If the mean level of clear goal falls below the ClearGoal_standard, a poor recommendation results.</td>
<td>$0 \leq x \leq 5, x \in \mathbb{R}$</td>
<td>.25</td>
</tr>
<tr>
<td>ClearGoal_alpha</td>
<td>The significance level used to determine whether the ClearGoal_standard has been met.</td>
<td>$0 \leq x \leq 1, x \in \mathbb{R}$</td>
<td>.05</td>
</tr>
<tr>
<td>Feedback_standard</td>
<td>If the level of feedback falls below the Feedback_standard, a poor recommendation results.</td>
<td>$0 \leq x \leq 5, x \in \mathbb{R}$</td>
<td>.25</td>
</tr>
<tr>
<td>Feedback_alpha</td>
<td>The significance level used to determine whether the Feedback_standard has been met.</td>
<td>$0 \leq x \leq 1, x \in \mathbb{R}$</td>
<td>.05</td>
</tr>
<tr>
<td>FeedbackImplementor</td>
<td>The name of the class that implements the environment-specific component of the FeedbackOutput object.</td>
<td>String</td>
<td>BarGraphImplementorSWT</td>
</tr>
<tr>
<td>ImproveRStrategy</td>
<td>The name of the implementation of the strategy used for improving recommendations.</td>
<td>String</td>
<td>ImproveMCRStrategy</td>
</tr>
<tr>
<td>$t_0$</td>
<td>Used to define the success of a recommendation – the higher the value of $t_0$, the stricter the cutoff for success is.</td>
<td>$0 \leq x \leq 1, x \in \mathbb{R}$</td>
<td>.5</td>
</tr>
<tr>
<td>lowest_n</td>
<td>The least number of ratings before the improving recommendations strategy can change the current index.</td>
<td>$x \in \mathbb{R}$</td>
<td>26</td>
</tr>
</tbody>
</table>

**Table 7.3:** The complete list of properties of the application framework relating to improving recommendations that can be set, along with a description of their purposes, the values they can have, and an example of each.
<table>
<thead>
<tr>
<th>Property name</th>
<th>Property description</th>
<th>Value space</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>feedback.name</td>
<td>Unique name for an instance of feedback</td>
<td>String</td>
<td>percentcomplete</td>
</tr>
<tr>
<td>feedback.type</td>
<td>The type of the most recently named feedback.</td>
<td>String</td>
<td>BarGraph1</td>
</tr>
<tr>
<td>feedback.contextType</td>
<td>The context type the most recently named feedback observes.</td>
<td>String</td>
<td>RelaxationPercentComplete</td>
</tr>
<tr>
<td>feedback.contextTypes</td>
<td>Used if a Feedback-Output object observe the multiple context types ($n = 2, 3, \ldots$)</td>
<td>String</td>
<td>RelaxationPercentComplete</td>
</tr>
<tr>
<td>feedback.updateFrequency</td>
<td>The rate the Feedback-Output object should be updated from the context it observes.</td>
<td>$x \in \mathbb{N}$ in milliseconds. Default: blank, updates every time the context changes.</td>
<td>400</td>
</tr>
</tbody>
</table>

Table 7.4: The list of properties relating to feedback, along with a description of their purposes, the values they can have, and an example of each.

This section describes two flow applications developed using the framework, Relaks, for the activity of relaxation, and Joga, for the activity of yoga, and gives a discussion of reusability and extensibility in the framework. These serve to demonstrate how framework components can be reused and extended to develop new flow applications.

### 7.2.1 Relaks

Most flow applications cannot be implemented from the application framework without extending it. One way to extend an application framework is to define new concrete subclasses of framework classes, and use these to implement an application \[107\].

Relaks is a flow application for the activity of relaxation, and it illustrates how the application framework can be extended to provide applications with the following:

- A context type not available in the application framework.
- A new type of task.
- A new method of calculating how much of the task has been completed.
An extension to a context type allowing it to be used with an existing feedback output.

New mastery criteria for use with the Stereotype strategy.

Note that these extensions are independent and hence this case study demonstrates how applications requiring one or more of these extensions may be implemented.

A number of relaxation techniques exist, including progressive relaxation, hypnosis, and visualisation [188]. However, in order to stay in flow, it is essential that as skills increase, more challenging goals are taken on. This is not possible with most of the techniques, because the tasks involved in these techniques do not have specific goals, such as the precise degree of relaxation a person must reach, and hence it is impossible to determine if the user reached a goal. One notable exception is biofeedback, which measures specific information related to a person’s physiological processes, such as heart rate or muscle tension, and in real time communicates this information to the person, who can use it to learn to control the particular physiological process [177].

Several forms of biofeedback can be used to measure relaxation. These include the electroencephalograph (EEG), which measures the activity of brain waves, the electromyograph (EMG), which measures muscle tension, and galvanic skin response (GSR), also called the electrodemograph (EDG), which measures electrical resistance of the skin [177]. Using any of these forms of biofeedback, it is possible to precisely specify the goal of a relaxation task. For example, with galvanic skin response, the higher the resistance (usually measured in ohms), the more relaxed person is.

Relaks is required to recommend suitable relaxation tasks, monitor the user’s galvanic skin response, and use this to supply the user with real-time feedback about how well he is doing (how close he is to reaching the goal). In order to implement this application, the framework must be extended by adding two new types of context (galvanic skin response and also a type that represents the percentage complete of a relaxation task) and a new type of task (specific to this activity). It also illustrates
how context can be extended to use an existing type of feedback output. Finally, since relaxation tasks defined in terms of biofeedback are automatically ordered by difficulty, the Stereotype strategy is the most suitable recommendation approach to take, and in order for this strategy to be used, the framework must be extended to include new mastery criteria.

**Implementation**

Figure 7.10 shows the classes added to the application framework to implement Relaks, along with their superclasses and the interfaces they implement. In order to distinguish the classes specific to Relaks from the framework classes, they are coloured yellow, while the framework classes are colourless.

**Adding a new type of context**

A type of context unavailable in the framework but needed for this application is galvanic skin response (GSR). The class GSR is responsible for acquiring and storing GSR, and also for sending GSR context events to feedback objects and other interested classes. GSR extends the abstract class Context, and acquires the current GSR value from a ThoughtStream™ (shown in Figure 7.11), which is a GSR device manufactured by Synetic Systems. A band containing two metal plates is connected to the ThoughtStream™. This band is placed on the hand, and by sending a small current through the hand, the ThoughtStream™ is capable of calculating the current GSR value two times every second, and can communicate this value to another device using a serial cable.

**Adding a new type of task**

A task in Relaks is different from the standard task available in the framework because the goal of a relaxation task is to become relaxed to a specific degree; this information
Figure 7.10: The extensions to the framework for Relaks.
Figure 7.11: The ThoughtStream™– a GSR device. The band with the metal plates on the left-hand side of the picture is placed on to the hand.
needs to be associated with the task. Specifically, as relaxation level is measured using galvanic skin response in Relaks, tasks in Relaks must be associated with a GSR object, which represents the goal of the task. This is achieved by extending Task with the subclass RelaksTask, which contains the method getGSRgoal.

Adding a new PercentComplete

A concrete subclass of the abstract class PercentComplete needs to be implemented for each new type of task developed. This concrete subclass is responsible for calculating the percentage of the task at hand that has been completed, for initialising the context required by this task, and for sending context events to the session manager, and other interested classes. In Relaks, this concrete subclass is RelaksPercentComplete. When RelaksPercentComplete is initialised, it acquires the GSR goal from the current task (the GSR goal is the GSR value that must be reached in order for the task to be complete). It also adds itself as an observer of the required context (GSR).

RelaksPercentComplete implements the two abstract methods of PercentComplete, onContextChangedEvent and update, which work as follows. When RelaksPercentComplete receives context events from GSR, the method onContextChangedEvent in RelaksPercentComplete is invoked, which updates the current GSR context and calls update, which uses the current context and the goal context associated with the task to update the current percentage of the task that has been completed. The session manager observes RelaksPercentComplete so that when a task is complete, it can select the next task.

Extending context to use a feedback output

To extend context so that it can be represented visually or auditorially, it must implement a FeedbackOutput interface. RelaksPercentComplete implements the BarGraph interface. This interface has one method, getX, where x represents the
Adding new mastery criteria

In contrast to Inka (Section 5.1.3), in which the criterion for mastery was that tasks had to be successfully completed once, in Relaks, the tasks must successfully be completed three times. In order to extend the Stereotype strategy so that these different mastery criteria could be used, RelaksCriteria, a different implementation of the MasteryCriteria was developed. The boolean method isMastered contains the behaviour that examines a user’s context history to determine if the task at hand has been completed three times. StereotypeStrategy instantiates the implementation of MasteryCriteria using reflection by reading the name of the class from the relaks.properties file.

7.2.2 Joga

Joga is a flow application for the activity of yoga, and it illustrates more complex cases of the extensions illustrated by Relaks, as well as some extensions that were not seen in Relaks, namely:

- Adding a new type of feedback.
- Extending task recommendation.

In contrast to relaxation, yoga tasks are complex – each can involve many different skills. A sequence of tasks ordered by difficulty does not easily emerge, as it did with relaxation. Consequently, for this application, the most suitable recommendation strategy is MCR, in which tasks are recommended based on a user’s confidence of the skills required for the tasks. An extension of the strategy is required to deal with the
vast number of yoga tasks in existence. Joga must also supply the user with a type of feedback unavailable in the framework, indicating whether his body is in the correct position, and signalling when a user has held a position for the required length of time.

Implementation

The classes added to the application framework to implement Joga, along with their superclasses and the interfaces they implement are shown in Figure 7.12. As was the case with the Relaks class diagram, the classes specific to Joga are coloured yellow to distinguish them from the framework classes, which are colourless.

Adding a new type of context

The key piece of context in this application is the position the user’s body is in. Although this context is more complex than the context added in Relaks, the application framework is nonetheless extended in the same manner. This context is modelled by the class BodyPosition, which extends Context, the class used in the framework to represent generic context. BodyPosition is responsible for acquiring and storing the position of a user’s body, and supplying context events to interested classes, such as YogaPercentComplete (the class that calculates how much of a task is done).

Accurately gauging body position requires a number of sensors. Fitzgerald et al. used 10 MTx sensors [208], each of which was placed on a different part of the body, and transmitted its current orientation [84]. The immense expense of these sensors precluded their use in Joga. Instead, 10 MTx sensors were simulated; each sensor is simulated by an instance of the Node class. The orientation of the MTx is modelled by Euler angles – three angles that describe the three rotations necessary to get from the initial orientation of the MTx to its current orientation. The Euler angles are represented in Node by the fields alpha, beta, and gamma. Each instance of BodyPosition is a composition of 10 instances of Joint, which represents the joints.
Figure 7.12: The extensions to the framework for Joga.
in the body, and is composed of two Node objects. For example, the elbow joint is represented by a node on the forearm and a node on the upper arm.

Adding a new type of task

Yoga tasks are more specific than the generic task supplied by the framework. A yoga task can be characterised by the position a person’s body must go into, and the length of time for which this position must be held. The class YogaTask extends Task and can supply the body position and time associated with the task using the methods getPositionGoal and getTime.

Adding a new PercentComplete

A concrete subclass of the abstract class PercentComplete is required. For this application, the class YogaPercentComplete fills this part. It is responsible for calculating the percentage of the task at hand that has been completed, initialising the context required by a task, and for sending context events to its observers, in particular, the session manager. During its initialisation, YogaPercentComplete acquires the goal body position and goal time from the task at hand, and uses these to initialise PositionMatch, a context type that represents how closely the current body position and the goal body position match. It also adds itself as an observer of the required contexts (PositionMatch and YogaTime, a context type representing the length of time a pose needs to be held for, and the length of time it has already been held for).

YogaPercentComplete fulfils its contract by implementing the two abstract methods of PercentComplete, onContextChangedEvent and update. When YogaPercentComplete receives a context event from PositionMatch, this signifies that the user’s current body position and the goal body position are considered to be equal. The method onContextChangedEvent in YogaPercentComplete is triggered, and this starts YogaTime.
When `YogaPercentComplete` receives a context event from `YogaTime`, it means that the user has held the pose for the required time, and `update` is called, which calls the `setValue` method of `PercentComplete` is called to declare the task done. The `SessionManager` was added as an observer of `YogaPercentComplete` in the constructor of `PercentComplete` using reflection, and thus the session manager is made aware when tasks are completed, so it can instruct `Session` to select the next task.

Adding a new type of feedback

When a user has managed to get into the required position, he receives feedback in the form of the sound of a drumbeat. This new type of feedback is unavailable in the framework. New types of feedback can be added by taking the following three steps. Firstly, an interface specifying the required behaviour of the feedback, and which extends the interface `FeedbackOutput`, is added. An implementation of this interface is also added. For this example (the drumbeat feedback), the interface added is `Sample` and the implementation of `Sample` is `BasicSample`. Secondly, the information about this type of feedback, in particular, its name and the context type that it requires must be included in the properties file, `yoga.properties`. When the next task of a session is selected, this information is used to set the feedback objects as observers of the context they require. In this example, `BasicSample` is initialised and set as an observer of `PositionMatch`, the context type that represents the closeness between the required body position and the current body position. Finally, to guarantee that the feedback object can use the context observes, the context type must implement the interface of the feedback in question. In this example, `PositionMatch` must implement the interface `Sample`. 
Extending context to use feedback output

Musika (Section 5.2.1) illustrated that supplying feedback is equivalent to representing a context type visually or auditorially, and to do this, the context type must implement the interface of the feedback it requires. PositionMatch implements Sample, and this allows a drumbeat to be sounded once the required position and the current position are the same. However, another requirement of Joga is that when the user has held the position for the required length of time, a gong sounds. In order to do this, the Time class would also have to implement Sample. However, since Time is a framework class, a subclass of it, YogaTime is created, and this class implements Sample. This achieves the same result as modifying the framework class, without modifying it.

Extending task recommendation

There are at least 1,300 yoga poses [148], and the number of poses with the right level of difficulty for a user could easily exceed a hundred. In an average session, a user is unlikely to do more than about 15 or 20 poses, and therefore there are many thousands of possible sessions. Most of these sessions would not satisfy a user’s purposes. Some users might want to reduce tension, others to build strength in certain muscles, others to increase flexibility of particular areas of the body, and still others might be doing yoga as part of a recovery programme for a particular ailment.

Therefore, some method of selecting the most relevant poses from those with the right level of difficulty is required. One way to do this is to tag each pose with keywords indicating purposes, such as “relaxation, neck” or “strength, shoulder”. Then, each user can compile a set of these tags, and give each a value between 1 and 100 denoting its importance; this can be used to compute modified values for each task, reflecting the relevance of the task to the user’s purposes.

This is achieved by extending the task recommendation strategy. In this applica-
tion, the multi-criteria recommender strategy, encapsulated by the `MCRSSStrategy` class is used. A class `YogaMCRSStrategy`, a subclass of `MCRSSStrategy` class is added. The method `getSuggestedTasks` in `YogaMCRSStrategy` first calls `getSuggestedTasks` in its parent class, which supplies the suitable tasks in order. Next, an instance of `YogaUser` representing the current user is retrieved from the user manager. `YogaUser` is an extension of `User` which contains the set of the tags discussed above, and allows the scores of the tasks to be adjusted according to relevance.

### 7.2.3 Discussion

In order for a framework to reduce the cost of developing applications, it must facilitate reuse. A standard metric for reuse level is defined as follows:

\[
\text{Reuse\%} = \frac{\text{RSI}}{\text{total statements}} \times 100\%
\]

where RSI is Reused Source Instruction, the total number of lines of code included in the source files of an application which had already been written; RSI covers only completely unmodified reused software units [175]. A high figure, such as 98.7% given by Relaks or 95.6% given by Joga indicates that only a small amount of new code was necessary to develop the application.

A framework cannot contain all the required classes for any application in a specific class of software, and consequently it is more important for a framework to be easy to extend [107]. A number of methods are used in the framework to facilitate reuse and extensibility without modifying framework source code: properties files, subclassing, abstract methods, reflection, and design patterns. The reusability and of extensibility of the framework components are discussed using examples from the flow applications described in this thesis.
Recommendation

The framework’s recommendation component can be reused and extended in a number of ways. The Stereotype approach can be reused for any suitable activity, but no customisation is possible without extension. This can be done by deciding on new mastery criteria, which can be implemented by creating a subclass of `MasteryCriteria`, and adding the name of the class to the application properties file. For example, in Relaks, the new mastery criteria are encapsulated in a new class `RelaksCriteria`.

In the MAUT approach, the relative importance of the attributes may be customised without modifying or adding source code but simply by means of editing the application properties file. This is one way of altering how the utility of a task is calculated. Another way is use a different aggregation function, such as a multiplicative aggregation function. This can be done by subclassing `MAUTStrategy` overriding the method `getMAUTValue`, and placing an implementation of the new outpatient function in the new method. Another extension of this approach is the addition of attributes. Since it has been acknowledged that the three key conditions of flow may not be the only conditions of flow [57], it is quite possible that additional attributes may need to be added in future. Or, additional factors, besides flow might be required to be taken into account. The MAUT approach allows for such a possibility, and furthermore, it caters for attributes that are composed of sub attributes. An example of an additional attribute is desire to play, allowing users of Musika (Section 5.2.1) to rate their level of desire to play particular piece. This can be done readily by adding the relative importance of the new attributes into the application properties file.

The MCR approach can be extended using a different aggregation function. For example, suppose an additional condition was added to the three key conditions of flow. This approach could be extended along another dimension, and giving four single criterion problems. The recommendation strategies for the individual components can be replaced or new strategies added. This can be done by subclassing the `MCRSStrat-`
egy class and overriding the method \texttt{getSuggestedTasks}. For example, in Joga, the subclass \texttt{YogaMCRSStrategy} is added to refine to the MCR Strategy. Finally, in addition to reusing and extending the task recommendation approaches supplied by the framework, it is possible for a developer to use an entirely different approach. This is done by adding a subclass of \texttt{RecommendationStrategy}, and adding the name of this subclass to the application properties file.

The strategy for improving recommendations is generic and can be used for any activity. Two customisations are possible. Firstly, the value of $t_0$, the threshold for defining the success of a recommendation may be freely chosen (within the range of $[0, 1]$). This enables an application developer to make the criteria for successful recommendation more lenient (by moving $t_0$ closer to 0), or more strict (by moving $t_0$ closer to 1). Secondly, both the standard of how unclear a goal must be before it is flagged and the standard of how poor feedback has to be before it is flagged, can be modified. This is likely to be used as a flow application evolves, and the size of the repository of tasks increases, whereupon application developers may wish to make the standards higher. The strategy can also be replaced by developers who wish to write their own strategies and use them in place of the one provided.

\textbf{Feedback}

Since feedback does not have a fixed meaning, it is ideal for reuse. The same \texttt{FeedbackOutput} objects can take different contexts as inputs in different applications.

For example, the \texttt{FeedbackOutput} representing the note a musician actually plays and the note he should have played (see Section\ref{sec:feedback}) could be used in another application, to represent the accuracy of a user performing a T’ai chi form (see \cite{47} for a description of an application which this feedback could enhance).

Consequently, \texttt{FeedbackOutput} objects are designed to be as flexible as possible. They have many properties that can be set. Some of these are general properties
that apply to all FeedbackOutput objects. For example, update frequency, that is, how often feedback is updated from the collection of context it observes – every x milliseconds seconds or every time there is a change in one or more pieces of context? It is possible that using different values of this property in different applications or for different users may produce better effects in terms of flow.

In addition to general properties, there are properties specific to a given type FeedbackOutput object. For example, the bar graph which changes colour at a particular threshold value can be given different colours, a different threshold value, and different maximum and minimum values. Or with the feedback object that outputs a tone at a certain rhythm, both the frequency and the rhythm can be customised. This enables FeedbackOutput objects to be used with a wide range of contexts and applications.

In cases where the framework does not have a FeedbackOutput object to satisfy an applications requirements, and a suitable FeedbackOutput cannot be obtained by customising or extending an existing FeedbackOutput object, application developers can create their own FeedbackOutput objects. This can be done by extending the interface FeedbackOutput, adding the new feedback implementation, and adding the name of the feedback and the required context type to the application properties file. The required context type must implement the new feedback interface. An example of this is found in Joga; Sample, a FeedbackOutput that plays a sample of music was added.

Measuring Flow

As was discussed in Section 2.1.2, the flow model currently has three key conditions of flow but it is possible in the future that other conditions will be found. Should this occur, it is possible to extend the mechanism for measuring flow to accommodate them. It is also possible to measure flow in different ways, such as by using combinations of
elements of flow (as described in Section 2.1.4) or even using EEG (if the research, described in Section 2.1.4, proves to be successful). These methods may be added by developers and used in place of the method provided by the framework by subclassing the class Flow and putting the name of the new subclass into the application properties file.

Context

As flow applications can be written for almost any activity, the context model must be able to incorporate a vast array of disparate context. To this end, the context model can usually be extended to include any new context type, simply by subclassing the abstract class Context. GSR in Relaks, and BodyPosition in Joga are two examples of this. A more complicated case is where a subclass requires certain behaviour. This is specified using the abstract methods which act as hooks making it clear to the developer precisely which methods need to be implemented. For example, adding a new type of PercentComplete (a subclass of Context) requires the developer to supply implementations of two abstract methods (onContextChangedEvent and update). In this way, the developer does not get mysterious errors if he doesn’t realise certain behaviour is required. Instead, he will get an explicit message at compile time informing him of the particular methods required.

In order to represent a context type visually or audiotorially, it must implement the relevant feedback interface. In Relaks, GSR implements the feedback interface BarGraph. In Joga, the context type (Time) is already available in the framework, and as it is the framework class it can not be modified. Nevertheless, extension is easily achieved by creating a subclass (YogaTime) of Time that implements the feedback interface (Sample). Extension is also eased using design patterns. For example, the Observer pattern makes it easy for a new class to monitor a particular context type. The new class only needs to implement the ContextEventListener interface, and
request the **ContextManager** to set it as an observer of the context, as illustrated by **YogaPercentComplete** in this application.

### 7.3 Conclusion

An iterative, application-led approach (the “Three Examples” approach [181]) was used to design the framework. This approach was favoured since the alternative, building a framework directly without first developing some applications, almost always fails in practice [4, 181]. The requirements for the application framework for flow applications were obtained by generalizing the requirements of three prototype flow applications: **Inka** (see Section 5.1.3), **Musika** (see Section 5.2.1), and **Inka 2** (see Section 6.1.3). The principal components of the framework are concerned with task recommendation, improving recommendations, providing feedback, and measuring flow.

The application framework provides three approaches to recommending tasks that are conducive to flow, which are suitable for different kinds of activities, enabling an application developer to choose the most appropriate approach for the application being developed. The first approach is the Stereotype approach. In this approach, a set of stereotypes is defined, each of which characterises the skills of a user. Users change from one stereotype to another when they have mastered the skills associated with the stereotype. The Stereotype strategy is suitable for activities in which tasks can be readily ordered by difficulty, such as relaxation or juggling. Its principal benefits are that tasks can be readily added by a domain expert (as long as in the given activity it is straightforward to order tasks by difficulty), and that it facilitates an increase or decrease in difficulty of the task is recommended to the user. Its principal limitations are its limited set of possible tasks a user can choose, and that a user may have to do many prerequisite tasks, to prove to the system that he has the skills he already knows he has.
The second approach to task recommendation is the Multi-Attribute Utility Theory (MAUT) approach, which is quite different from the Stereotype approach. In this approach, a user rates relevant attributes of tasks, and the relative importance of these attributes are quantified, resulting in an overall utility value for each task. The MAUT approach is suitable for activities in which users spontaneously come up with ideas for tasks, such as thinking of a song they’d like to perform (see Section 5.2.1). The principal benefits of this approach are its potential to give the user much greater choice; this can be further increased by the user who can readily add new tasks. In addition, unlike the Stereotype approach, a user does not have to do any tasks to prove to the system that he has certain skills. The principal shortcomings of this approach are that as the user’s skills increase, he will have to re-rate the tasks, which would be an issue in systems with a large number of tasks, and that tasks that haven’t been rated by the user cannot be recommended.

The third approach is the Multi-Criteria Recommender (MCR) approach in which each task is indexed by a set of skills, and a rating for it is estimated using the user’s perception of his confidence doing these skills, along with ratings for clear goal and feedback from other users who did the task. The MCR approach is suitable for activities which have been or can be decomposed into a specific set of skills, such as yoga (see Section 7.2.2). The principal advantages of this approach are its potential for greater choice, that unseen tasks can be recommended to a user, and that difficulty can be increased or decreased by the user by reappraising his skills. The principal drawback of this approach is that indexing a task can be time-consuming, and not even a domain expert can guarantee its accuracy.

The framework supplies an approach to identifying and improving poor recommendations so that subsequent users receive better recommendations. This approach analyses the ratings users gave to tasks recommended to them. Poor recommendations can be identified by unsuitable confidence level (that is, the task was either too easy or
Tasks that resulted in poor recommendations because of a substandard level of clear goal or of feedback are flagged, and the main benefit of this approach is that it gives content developers the opportunity to improve upon and re-release the items, rather than simply allowing items to fall out of use. **The main drawback of the approach is that it requires user effort** (users must provide an estimation of the attributes for each task they do).

Tasks that resulted in poor recommendations because of a misguided index are dealt with by calculating better item metadata (that is, a better skill index). The main advantage of this approach is that it is automatic, that is, it does not require the input of a domain expert but only user driven context snapshots. Its main limitation is the number of users who must do a task before it can produce a more accurate index with a low risk of error; this is unavoidable since sufficient data must be collected, and an alternative approach would necessarily have the same limitation.

The final two key features provided by the framework are supplying feedback and measuring flow. Feedback is information, and thus it is usually in visual, auditory, or haptic form. In the framework, feedback is a visual, auditory, or haptic representation of a collection of context objects. The mechanism for providing feedback enables any available context to be used as a source of feedback, and supplying it to the user by means of any available actuator, thus making a vast range of feedback available.

To measure flow, the framework considers flow as context, and determines whether a user is in flow by whether the three key conditions of flow are present. This enables the absence of flow to be detected, so that measures can be taken to aid its return. The means provided by the framework to measure flow (taking context snapshots) enables flow to be measured quickly and in a mobile environment, so that it can be used wherever the activity normally takes place.

The framework takes into account the limitations of mobile devices such as PDAs, ensuring that applications developed using the framework can be deployed and executed
In a mobile environment. In particular, the implementation of the framework is in written entirely in Java ME [144], which is designed specifically for mobile devices.

A framework can reduce the cost of developing applications from a specific class of software [181]. To achieve this, framework components need to be designed so that they can be reused in many new applications [79]. In addition, because “applications seem infinitely variable”, a framework cannot contain all the required classes for any application in a specific class of software [107]. Therefore, it is more important for a framework to be easy to extend than for it to have all the features required for an application [107]. Hence, two important characteristics of a framework are its reusability and its extensibility. Consequently, at the implementation level, the chief concern was that the framework should facilitate both reuse of framework behaviour and extension in many sections of the framework.

This chapter described two flow applications developed using the framework, Re-laks, for the activity of relaxation, and Joga, for the activity of yoga, and it gave a discussion of reusability and extensibility in the framework. These serve to demonstrate how framework components can be reused and extended to develop new flow applications. A number of methods are used in the framework to facilitate reuse and extensibility without modifying framework source code: properties files, subclassing, abstract methods, reflection, and design patterns. The reusability and of extensibility of the framework components were discussed using examples from the flow applications described in this thesis.
Chapter 8

Conclusions and Future Work

This chapter concludes the thesis by drawing the work together, examining its chief contributions, discussing the limitations of the thesis, and outlining some possible areas of future work.

8.1 Achievements

Flow is an immensely enjoyable mental state that is characterised by a “complete immersion in what one is doing” [61]. Indeed, it is so enjoyable that people invest considerable amounts of time and money “for the sheer sake of doing it” [59]. A model of flow, which has evolved over the last three decades, asserts that three key conditions must be present for a person to experience flow: a person must engage in a challenging task that requires skills and he must believe his skills match the challenges of the task; the task must have clear goals; and the task must provide immediate feedback.

To go from ordinary experience to flow experience, we must ensure the key conditions of flow are present. Unfortunately, this is not easy [59]. Flow applications, that is, applications that aim to assist their users to experience flow can be built for almost any activity, provided the activity supplies a set of challenges that require skills.

218
Requirements analysis of several flow applications for diverse domains produced a set of requirements necessary for any flow application. These comprise: the flow application must recommend tasks whose challenges the user believes match his skills, which have clear goals, and which provide or enhance feedback. It should also continuously improve on these task recommendations and it should measure flow. Central to flow applications is the recommendation of tasks likely to produce the key conditions of flow. This observation led to the main research question addressed by this thesis: how can tasks likely to produce the key conditions of flow be recommended?

A review of related work was conducted. First, systems that influence the conditions of flow, while the user is engaged in an activity in a computer mediated environment were reviewed. The systems met or partially met some of the requirements for a flow application, and each of the systems reviewed provides the user with tasks. However, in terms of producing the conditions of flow, task recommendation is of great importance, in particular balancing skills and challenges, but it receives little attention in any of the reviewed systems.

Next task recommendation was reviewed, including approaches to task recommendation, some representative systems, and approaches to improving recommendations. A crucial difference between recommending tasks to produce flow and recommending other items is the challenge of finding tasks whose challenges match user skills. This means that the challenges of tasks and the user’s current skills need to be taken into account. Furthermore, as a user learns, the level of his skills changes, and essentially, the user becomes a different person. This has been called the “stability versus plasticity” problem [41], and a consequence of it is that some recommendation algorithms are unsuitable for recommending tasks to produce flow. In particular, collaborative approaches are unsuitable for this reason, and this means a content-based approach is required, giving rise to a second challenge of task recommendation: accurately determining from tasks the information required by the recommendation algorithm.
Almost all systems involving task recommendation are in the domain of education; some representative examples of these systems were reviewed, and a number of limitations were identified. Firstly, the chief limitation of automatically updating the user model as done in many of the reviewed works, is that it is an objective measurement of mastery of skills, and there is no guarantee that this will coincide with a user’s perception of his mastery of the skills (required to meet one of the key conditions of flow). For instance, completing a task need not lead to an increase in perceived skill, or a skill may be forgotten or partially forgotten.

Secondly, all of the systems reviewed are limited by low precision, which is a concern for maintaining the “delicate balance”\textsuperscript{[54]} between challenges and skills required for flow. The approaches taken by the systems include stereotype models, scalar models, overlay models representing the user’s knowledge of a concept as a binary value (known/not known). Thirdly, tasks also need clear goals not only to meet the key conditions of flow, but also so that the task can be indexed properly (how can the required skills be determined if it is not clear what the goal is?). Some of the reviewed systems have clear goals and while the others potentially have clear goals, whether they actually do is not taken into account.

Recommender systems that do not improve over time will repeatedly produce the same poor recommendations, and in the case of task recommendation for flow, this will lead directly to the absence of the conditions of flow. This provides considerable motivation to continually improve recommendations. Approaches to improving recommendations include collaborative approaches, honing algorithms for a particular domain, hybrid approaches, multi-criteria approaches, and using context. All of these approaches have limitations, but the main limitation is that, while each of these approaches has the potential to improve recommendations over a system that doesn’t use the approach, only the collaborative approaches continuously improve the recommendations over time. Collaborative approaches, however, can’t be used for recommending
tasks for flow because collaborative approaches assume that users’ characteristics remain static, which is not the case when recommending tasks for flow.

An analysis of a set of recommendation strategies against the requirements for recommending tasks for flow identified the three most promising approaches: the Stereotype, MAUT, and MCR approaches. These were adapted and applied to the problem of recommending tasks to support the creation of flow, the first contribution of this thesis. The method for evaluating these was to develop flow applications containing the task recommendation strategies and to deploy them in authentic environments, and to measure the effect using relevant response variables. Authentic environments are necessary since assumptions about user behaviour and the effect the environment will have are often flawed [64].

In Pilot Study 1, the effectiveness of the Stereotype strategy was evaluated using Inka, a flow application containing the strategy for the activity of programming. The mean percentage of time subjects spent with the conditions of flow absent was found to be 19.3%. This result compares well with the ideal of 0% (that is, subjects spending the entire time with the conditions of flow present). However, failure to obtain permission to deploy Inka in the classroom led to a sample of six users, and consequently while the result does provide evidence of the effectiveness of the Stereotype strategy, the small sample size limits the strength of the result. An insight gained from this study was that the unexpected user behaviour (failure to create context snapshots) causes the measure of flow used to lack accuracy.

In Pilot Study 2, Musika, a flow application that contained the strategy for the activity of practising or playing music was built. This activity was chosen primarily because playing music is a very different activity to computer programming, and this is conducive to generalising the task recommendation strategies. It was planned to measure the effectiveness of the MAUT strategy using the conditions of flow for a task as a whole. However, a problem with this was identified. The MAUT strategy produces
a recommendation score for each of the available tasks according to the user’s initial snapshot, but an accurate recommendation score depends on the user’s ability to correctly gauge tasks; the data gathered from the studies so far suggests that this requires a lot of practice. Determining a user’s ability to correctly gauge tasks would require a large-scale investigation, in which each user would rate a large set of tasks, do the tasks, and then re-rate the tasks. This is an extremely time-consuming process and the study could not be completed due to a lack of available resources. However, it is intended to study whether the difference between the user’s initial snapshot and final snapshot diminishes over time as part of a future longitudinal study.

The aims of Study 1 were to evaluate the effectiveness of the MCR strategy and to identify any usability problems with the flow application used in the study. This study addressed a number of shortcomings of the pilot studies: it had a larger sample size than the pilot studies; it used a different measure (measuring flow of a task as a whole) so that subjects’ omission to create snapshots was no longer an issue; and it evaluated usability – this is an important aspect of the research question since producing effective recommendations will be of limited value if the user is not satisfied with how the recommendations are produced.

In Study 1, the precision of the MCR strategy was calculated as 100%, and the mean task score calculated, with 95% confidence, to be in the range [91.5, 95.25], well above the minimum of 80 (the minimum requirement for the three key conditions of flow to be present). These results certainly supply evidence of the effectiveness of the MCR recommendation strategy. However, it is important to observe that these results are limited, since despite the time-consuming nature of the gathering of this data, it is still a relatively small dataset (just 25 tasks were rated). In order to obtain a stronger result, more data is essential. With regard usability, a number of issues were identified, but none of the serious nature. That is, in each case, a solution for overcoming the issue readily suggested itself.
Study 2 had three aims. The first two aims of the study were identical to those of Study 1; the third aim relates to the second contribution of the thesis and will be discussed shortly. This study addressed two shortcomings of Study 1. Firstly, rather than getting students to take part in sessions individually, an entire class group (comprising 12 students) took part in sessions together – the teaching scenario for which the flow application was originally designed. Secondly, the study enabled considerably more data to be gathered since it took place in classes over 10 weeks of a single semester course, compared with the two 90 minute sessions used in Study 1. The precision of the MCR strategy was calculated as 88.1%, and the 95% confidence interval of the mean task score as [80.8, 86.3]. These results offer clear evidence of the effectiveness of the MCR strategy, and although not as high as in Study 1, they are stronger results since they are based on a larger dataset (189 tasks). With regard to usability, no serious issues were identified.

The second contribution of this thesis is an approach to continuously improve recommendations for flow over time by modifying items and/or items’ metadata, based on items identified using user ratings (acquired from context snapshots). No recommender systems could be found that take this approach. The ideal approach to demonstrate the effectiveness of the strategy is by means of a real field experiment, in which a flow application containing the strategy is deployed in an authentic environment. The third aim of Study 2 was to do just this.

In Study 2, no tasks were identified as having insufficiently clear goals or inadequate feedback. The minimum rating for each of these conditions of flow to be present is 3. The average clear goal rating for all tasks was 4.37, and the minimum average for an individual task was 3.86. The average feedback rating for all tasks was 3.91, and the minimum average for an individual task was 3.25. Had any tasks been identified as subpar, they would have been flagged for a content developer to examine and improve upon. With regard to the third criteria, confidence level, no task was rated enough for
the confidence component of its recommendation score to be identified as poor.

In order for Study 2 to fully evaluate the approach, substantially more users than were available would be required. With the current class sizes in computer science, it is likely that it would take a number of years to gather the required data. Consequently, a third study was carried out, sidestepping the problem of insufficient users by simulating the users. The aim of the third study was to evaluate the strategy for improving poor recommendations caused by an inaccurate skills index – this is the only one of the three causes that can be improved computationally. (The other two causes, unclear goal and poor feedback, are flagged so that content developers can manually improve them).

It was expected that if only a few ratings were available, the percentage accuracy of the strategy would be low. This proved to be the case, for example, when five users have rated the task, the simulation showed that the percentage accuracy was just 60%. However, for a strategy to be successful, it should demonstrate high percentage accuracy once it has sufficient data, and this also proved to be the case. By repeatedly simulating the same situation but each time incrementing the least number of ratings required, we found that percentage accuracy increases to 100%. Moreover, the simulation suggested that the value of the least number of ratings the strategy should have before making a decision in order for percentage accuracy to be close to 100% should be at least 26 ratings for a given task. The main limitation of simulations is that they might not accurately model real user behaviour. Nevertheless, simulations are useful in indicating some degree of effectiveness before running field experiments, which are costly and cannot easily be repeated or altered midstream [68]. Moreover, field experiments can be used to inform future simulations, making them increasingly accurate.

Taking the results from all of the studies, what can be concluded? With regard to usability of the flow applications deployed, some small issues were identified that
can be readily rectified. With regard to the main question investigated in this thesis, we have gathered evidence of the effectiveness of the task recommendation strategies described in this thesis in producing the conditions of flow, though some limitations—chiefly the sample size—limit the strength of the results.

A secondary research question was also considered in the thesis: how can applications that assist its users to experience flow be developed for any activity? To this end, the thesis provides two contributions: a description of the iterative design and development of a flow application (Inka) for introductory computer programming, and a framework for developing flow applications. Inka was deployed in an authentic environment and built on the limitations of existing flow applications, including the methods of measuring flow, task recommendation, and improving recommendations. Each iteration of Inka built on previous one and the iterations are described in Pilot Study 1 (on page 107), Study 1 (on page 137), and Study 2 (on page 149).

The key features of the framework are concerned with task recommendation, improving recommendations, providing feedback, and measuring flow. The framework provides the implementations of each of the task recommendation approaches described in this thesis (Stereotype, MAUT, and MCR). It also provides an implementation of the approach for improving recommendations. In the framework, feedback is a visual, auditory, or haptic representation of a collection of context objects. The mechanism for providing feedback enables any available context to be used as a source of feedback, and supplying it to the user by means of any available actuator, thus making a vast range of feedback available. To measure flow, the framework considers flow as context, and determines whether a user is in flow by whether the three key conditions of flow are present. The means provided by the framework to measure flow (taking context snapshots) enables flow to be measured quickly and in a mobile environment, so that it can be used wherever the activity normally takes place.

A framework can reduce the cost of developing applications from a specific class
of software [181]. To achieve this, framework components need to be designed so that they can be reused in many new applications [79]. In addition, because “applications seem infinitely variable”, a framework cannot contain all the required classes for any application in a specific class of software [107]. Therefore, it is more important for a framework to be easy to extend than for it to have all the features required for an application [107]. Hence, two important characteristics of a framework are its reusability and its extensibility. Consequently, at the implementation level, the chief concern was that the framework should facilitate both reuse of framework behaviour and extension in many sections of the framework.

This thesis described two flow applications developed using the framework, Relaks, for the activity of relaxation, and Joga, for the activity of yoga, and it gave a discussion of reusability and extensibility in the framework. These serve to demonstrate how framework components can be reused and extended to develop new flow applications. A number of methods are used in the framework to facilitate reuse and extensibility without modifying framework source code: properties files, subclassing, abstract methods, reflection, and design patterns. The reusability and of extensibility of the framework components were discussed using examples from the flow applications described in this thesis.

8.2 Limitations of the Thesis

This thesis has a number of limitations. The small sample sizes of the studies limit the strength of the results of the effectiveness of the task recommendation strategies. Moving forward, the goal is to increase the reliability of the results. This is necessary so that as new task recommendation strategies are developed, their effectiveness can be compared against the existing ones. It can be achieved by conducting further similar studies with the same activity and also in different activities, and by gaining access to
larger samples. In addition, a lack of available resources prevented the MAUT strategy from being evaluated as desired. To this end, it is intended to study whether the difference between the user’s initial snapshot and final snapshot diminishes over time as part of a future longitudinal study.

Another limitation is that the framework was not evaluated. Also, because skills can change frequently, and users must ensure that they update their skills models, if a user neglects to update his model, his inaccurate skills model may result in him receiving poor recommendations. Finally, while the task recommendation strategies recommend individual tasks, they do not recommend sequences of tasks, which could enable longer, more seamless flow experiences. Approaches to how these limitations might be overcome are outlined in the next section, which discusses future work.

8.3 Future Work

8.3.1 Recommending Sequences of Tasks

Instead of recommending individual tasks, sequences of tasks could be recommended. Two particular uses of this are recommending a session, and transforming a desirable, but currently undoable, project into a doable project.

A session is a sequence of tasks, and at its simplest, recommending a session could involve taking the current highest recommended task and putting it as the first task of the session. Then, by assuming that the task will be completed successfully, the highest recommended task can be put as the next task in the session. Task selection can continue in this fashion until the session is ready for use. For example, one of the criteria for this could be that the estimated time to complete all the tasks in the session is within a specified range.

However, there may be approaches that yield much better results than this simple approach. For instance, a set of modifiable rules, aimed at maximising flow could be
used to design the session. Examples of possible rules include: ending the session on a ‘high’, to take advantage of the recency effect [207]; exploiting the advantages of grouping certain types of tasks (for example, some types of tasks may prove to be ideal warmup tasks to other candidate tasks); and optimal placement of paused tasks – that is tasks that were stopped in the middle of a previous session (for example, if task had been going well, it should be placed at the start of the session, to take advantage of a phenomenon identified by Kay [172]). Moreover, these rules would have to take clashes into account, that is, if two rules could be applied in a situation, should one or the other or both be applied?

Transforming a desired, but currently undoable, project into a doable project requires recommending a sequence of tasks that aims to improve the skills required for the project from the user’s current perceived levels to the levels necessary to do the project. This necessitates a knowledge of how doing a particular task is likely to influence a set of skills. The acquisition of such knowledge is challenging since the evolution of skills is a complex process involving a great deal of uncertainty. A possibility for modelling this process is to use a Hidden Markov Model [209].

In addition, the sequence of tasks must be monitored as the user is completing them, so that if the user’s behaviour departs sufficiently from the expected behaviour, the sequence will be modified, so that the user is likely to experience flow as he does the modified sequence of tasks.

8.3.2 Dealing With an Inaccurate Skills Model

Skills can change frequently, as a user learns, or indeed forgets, and users must ensure that they update their skills models. Of course, users may neglect to do this, causing them to have an inaccurate skills model, and consequently, unsuitable tasks will be recommended to the user. A solution that naturally suggests itself is to automatically update the skills model. However, in flow applications, the skills model represents a
user’s perception of his skills and so it cannot be automatically updated (no technology at present is capable of reading a user’s perceptions without input from the user).

There is, however, the possibility of using a mixed approach – that is, a mixture of the manual approach and an automatic approach. One way this might be achieved is to have two skill models for each user: the usual one, representing the user’s perception of his skills, and another representing the system’s estimate of the user’s skills based on evidence it has acquired (such as those used in [52] and [147]). The purpose of the second model is to compare it with the first and when the level of disagreement exceeds a specified threshold, the user can be prompted to update particular skills in the model, and he can look at the model and decide whether to change a skill or ignore the system.

A user’s confidence that he can do particular skills may change due to a host of more ephemeral factors, such as a user’s energy levels, alertness, or psychological factors, such as recent failures. Dealing with such factors seems to call for an ability to make changes, with a smaller scope than changes made to the skills model. For example, changes could be made while doing a task, which would have no impact on anything besides the task itself. In such an approach, the skills model is left untouched, but the tasks are made easier or harder, perhaps by means of a simple interface such as a slider or plus and minus buttons. Many factors can change the difficulty of a task. Probably the simplest is imposing or reducing a time limit. Another is altering the goal. Suppose, for example, that the goal concept was extended beyond a description that evokes an image in the user’s mind into something that can be parameterised. In this way, a constraint could be relaxed. For example, a parameter $n$, representing the number of bars a musician must play without error could be reduced from ten down to six, thus making the task easier.

A third factor that can change the difficulty of a task is supplying resources to the user. Resources are used in this thesis to mean anything that can assist a user
to complete the task. Examples of resources include tools and text, for example, in a programming task, a resource might describe how particular Java methods work (see Figure 5.7 on page 121). One challenge arising from supplying a user with resources is to ensure that, equipped with the resource, the task does not go directly from being too hard into being too easy, bypassing the suitable range.

Another challenge arising from supplying a user with resources is that some tasks have a high degree of variety [82], that is, there may be several ways of doing the task. Consequently, different ways of doing the task may be assisted by different resources. One approach to deal with this issue is to limit the ways in which the task can be done as part of the task’s goal. This approach was adopted in flow applications described in this thesis. Another possible approach is to use plan recognition [76], which could also use context to assist the plan recognition process [191].

There is no reason why the two approaches outlined (using two skills models and making changes to tasks as they are being done) cannot be used simultaneously. Furthermore, these are just two of the many possible approaches to dealing with an inaccurate skills model.

8.3.3 Evaluation of the Framework

Developing a framework that is highly reusable, extensible, and easy to use requires time and technical expertise; consequently it is typical for several versions of a framework to be released before a stable, reliable, a mature system is released [16]. Early versions support only core features, and later versions aim to be robust, flexible, easy to use, and complete (that is, have all the required features) [16].

Traditional software metrics such as size, complexity, and performance are insufficient for evaluating frameworks [16]. For example, one important characteristic is how easy it is to learn a framework; if it takes a long time to learn a framework, then it could easily exceed the time saved developing an application using the framework.
Another important characteristic is to measure the extent of change as a framework evolves, in particular structural characteristics and stability of architecture; this can be measured using a suite of object-oriented design metrics [17]. Furthermore, evaluating some characteristics of frameworks is possible only by building applications using the framework and evaluating those applications [75]. Evaluation of a framework may be facilitated by tools such as SEA [73] that automatically gather metric values for a framework and software developed using the framework, and enable visualisation of this data.

8.3.4 Unified Flow

At the conclusion of the book Flow [59], Csikszentmihalyi describes unified flow, which essentially involves transforming life into “a seamless flow experience” [59]. It is already difficult to transform one ordinary experience into a flow experience; to transform all experience into flow experience is a formidable challenge indeed.

While each flow application is designed to assist a user to experience flow in one particular activity, software designed to assist the user to experience flow in any activity could not be achieved in the same way – the goal is far too big to be achieved by a single application. A possible approach is to use Service-oriented architecture [78].

A feedback service could give users the option of activating one or more of the feedback devices available in the environment the user is in. For example, turning on smart juggling balls could enable them to supply the user with feedback in the form of changing colours. Or, turning on augmented reality glasses could enable feedback to be superimposed on the user’s environment.

A task recommendation service could give users the option of receiving, if desired, recommendations of tasks likely to create the conditions of flow. In order to do this for every activity, a complete model of all human skills would be required, which is needless to say, a vast undertaking. Alternatively, a task identification service could
identify the task of the user is doing and if the user desires, recommend changes that will increase the likelihood of flow.
Appendix A

Study Materials
A.1 Pilot Study 1 Reference Sheet

How to create a context snapshot
To create a context snapshot, simply click on the camera icon, and fill in the form that appears. You can do this by asking yourself the questions in Table 1 and choosing the most appropriate answer. For the best results, it is crucial that you answer completely truthfully, and that you create a context snapshot as soon as you notice your context has changed (especially when either your confidence or feedback are 2 or less).

<table>
<thead>
<tr>
<th>variable</th>
<th>question and possible answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear goal</td>
<td>How clear is the goal? 1. very unclear 2. somewhat unclear 3. somewhat clear 4. quite clear 5. very clear</td>
</tr>
<tr>
<td>confidence</td>
<td>How confident are you that you will succeed with the task (that is by yourself without any help)?  1. definitely won’t succeed 2. more than likely won’t succeed 3. might succeed / stand a chance at succeeding 4. probably will succeed 5. definitely will succeed)</td>
</tr>
<tr>
<td>feedback</td>
<td>Do you know how well you are doing right now with this task? 1. not at all 2. not really 3. some idea 4. good idea 5. know exactly how well I’m doing</td>
</tr>
<tr>
<td>meaningful</td>
<td>How important is this task in relation to your overall goals? (That is do you think the task will bring you closer to your goals or is it a waste of time?) 1. not at all 2. not important 3. somewhat 4. quite 5. very important</td>
</tr>
<tr>
<td>concentration</td>
<td>How well are you concentrating? 1. not at all well 2. not well 3. somewhat well 4. quite well 5. very well</td>
</tr>
<tr>
<td>sense of control</td>
<td>Do you feel in control? 1. not at all 2. no 3. somewhat 4. quite 5. very much</td>
</tr>
</tbody>
</table>

Table 1: The options for the form to measure flow
A.2 Study 1 Reference Sheet

Update your skills
The first thing to do is to update your skills. You do this by rating your perception of confidence that you can do specific concrete tasks. These are then generalised into your perceptions of your skills. You can update whenever you believe your skills have changed by clicking “edit skills model”.

Get suitable task
The system suggests a task suitable given your skills.

Doing the task
During the task, you will make several context snapshots. To make a context snapshot simply click on the camera item. You can do this by asking yourself the questions in Table 1 and choosing the most appropriate answers. For the best results, it is crucial that you answer completely truthfully.

<table>
<thead>
<tr>
<th>variable</th>
<th>question and possible answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>clear goal</td>
<td>How clear is the goal?</td>
</tr>
<tr>
<td></td>
<td>1. very unclear</td>
</tr>
<tr>
<td></td>
<td>2. unclear</td>
</tr>
<tr>
<td></td>
<td>3. somewhat clear</td>
</tr>
<tr>
<td></td>
<td>4. quite clear</td>
</tr>
<tr>
<td></td>
<td>5. very clear</td>
</tr>
<tr>
<td>confidence</td>
<td>How confident are you that you will succeed with the task (that is by yourself without any help)?</td>
</tr>
<tr>
<td></td>
<td>1. definitely won’t succeed</td>
</tr>
<tr>
<td></td>
<td>2. more than likely won’t succeed</td>
</tr>
<tr>
<td></td>
<td>3. might succeed / stand a chance at succeeding</td>
</tr>
<tr>
<td></td>
<td>4. probably will succeed</td>
</tr>
<tr>
<td></td>
<td>5. definitely will succeed</td>
</tr>
<tr>
<td>feedback</td>
<td>Do you know how well you are doing right now with this task?</td>
</tr>
<tr>
<td></td>
<td>1. no idea</td>
</tr>
<tr>
<td></td>
<td>2. v slight idea</td>
</tr>
<tr>
<td></td>
<td>3. some idea</td>
</tr>
<tr>
<td></td>
<td>4. good idea</td>
</tr>
<tr>
<td></td>
<td>5. know exactly how well I’m doing</td>
</tr>
</tbody>
</table>

Giving a value of 1 or 2 and the teaching assistant will come over and give you feedback, including an estimate percentage done of the task.
When to make context snapshots

(1) Initial snapshot
Once you are read the goal of the task you then must make an initial snapshot.

(2) Feedback
You can create a snapshot and use the feedback object to estimate your % complete. You can also get “certified feedback”. To increase this, you must convincingly demonstrate to the teaching assistant that you have done the stages you claim to have done, and the TA will increase your certified progress.

(3) Low confidence (2 or less)
(a) Get one of the pieces of material. Some tasks have a set of material associated with them. For example, a piece of material might tell you how Math.abs() works.
(b) Put a task aside. If you have seen all the material and still think you do not stand a chance of succeeding (confidence 2 or lower) you can put the task aside and get a new task. You can take it up again in some future session.

(4) Finished task
When a task is 100% complete (certified by the TA), or when you put it aside, you must make a final snapshot. This is slightly different from the usual snapshot (see Table 2).

<table>
<thead>
<tr>
<th>variable</th>
<th>question and possible answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear goal</td>
<td>How clear was the goal?</td>
</tr>
<tr>
<td></td>
<td>1. very unclear</td>
</tr>
<tr>
<td></td>
<td>2. unclear</td>
</tr>
<tr>
<td></td>
<td>3. somewhat clear</td>
</tr>
<tr>
<td></td>
<td>4. quite clear</td>
</tr>
<tr>
<td></td>
<td>5. very clear</td>
</tr>
<tr>
<td>Confidence / Balance of challenge and skills</td>
<td>How did you find the task?</td>
</tr>
<tr>
<td></td>
<td>1. very easy</td>
</tr>
<tr>
<td></td>
<td>2. easy</td>
</tr>
<tr>
<td></td>
<td>3. just right</td>
</tr>
<tr>
<td></td>
<td>4. hard</td>
</tr>
<tr>
<td></td>
<td>5. very hard</td>
</tr>
<tr>
<td>quality of feedback object</td>
<td>How would you rate the feedback object (in terms of telling you how well you’re doing with the task)?</td>
</tr>
<tr>
<td></td>
<td>1. very poor</td>
</tr>
<tr>
<td></td>
<td>2. poor</td>
</tr>
<tr>
<td></td>
<td>3. ok</td>
</tr>
<tr>
<td></td>
<td>4. good</td>
</tr>
<tr>
<td></td>
<td>5. very good</td>
</tr>
</tbody>
</table>

Table 2: Final snapshot

Finishing tasks
Because everyone isn’t working on the same task, it is not necessary to finish your current task in the current session; if you get a task partially completed, that’s fine- you can continue where you left off in the next session.
A.3 Study 1 Initial Questionnaire

Daire Ó Broin, Department of Computer Science, Trinity College Dublin

Name: ____________________________

1. Do you enjoy programming in Java?

☐ really don’t enjoy it  ☐ no  ☐ somewhat  ☐ yes  ☐ very much

2. How interested are you in Java?

☐ not interested – just want to pass the course  ☐ little interest  ☐ some interest  ☐ interested  ☐ very interested

3. In general, how difficult do you find programming in Java?

☐ very difficult  ☐ difficult  ☐ so-so  ☐ easy  ☐ very easy

4. In general, how would you rate your skills?

☐ very low  ☐ low  ☐ average  ☐ high  ☐ very high

5. How important is Java programming in relation to your future goals?

☐ really unimportant  ☐ unimportant  ☐ somewhat important  ☐ important  ☐ very important

6. Would you be interested in taking part in the evaluation of Inka, for a fee (€10/hr)?

☐ yes  ☐ no
A.4 Study 1 Final Questionnaire

Usability

How did you find making context snapshots (once you got used to it)?

☐ v inconvenient  ☐ inconvenient  ☐ neither  ☐ convenient  ☐ v convenient

How distracting did you find the tool?

☐ not distracting  ☐ somewhat distracting  ☐ distracting  ☐ v distracting

Confidence

If your confidence level is low (2 or 1), you can be supplied with one or more of the pieces of material attached to the task (for example a piece of material might tell you how Math.abs() works).

If you have seen all the material, and your confidence level is still low, would you have any difficulty setting aside the current task and choosing another?

☐ yes  ☐ no

Feedback

(a) Did you find the feedback objects useful as a means of calculating feedback during the session?

☐ yes  ☐ no

(b) In general, do you the feedback objects are:

☐ Completely unnecessary – I can come up with test cases myself
☐ Sometimes necessary – I sometimes need to have test cases provided.
☐ Necessary more often than not – I always need to have test cases provided most of the time
☐ Necessary most of the time – I always need to have test cases provided most of the time
☐ Always necessary – I always need to have test cases provided
(c) You can demonstrate your algorithm to the teaching assistant (TA) who updates your progress. Do you think this feedback is:

- Completely unnecessary – I can tell if the algorithm works myself
- Sometimes necessary – I sometimes don’t know if my algorithm works
- Necessary more often than not – more often than not, I don’t know if my algorithm works.
- Necessary most of the time – I don’t know if my algorithm works most of the time
- Always necessary – I never know if my algorithm works

(d) You can demonstrate some code to the TA who updates your progress and tells you how much is right. Do you think this feedback is:

- Completely unnecessary – I can tell if the tell if my code works myself
- Sometimes necessary – I sometimes don’t know if my code works
- Necessary more often than not – more often than not, I don’t know if my code works.
- Necessary most of the time – I don’t know if my code works most of the time
- Always necessary – I never know if my code works

**Effectiveness of the system**

Overall do you agree that the system is effective?
(That is: it provides tasks with clear goals, it matches your skills to the challenges of the task and it provides feedback to help you know where you are in relation to the goal of the task).

- strongly disagree
- disagree
- neither
- agree
- strongly agree

**Choosing your task**

Would you like more say in choosing your task?

- yes
- no
A.5 Study 1 Consent Form

Your participation in this experiment is voluntary. You may withdraw at any stage for any reason. However, if you do so you will not receive the agreed fees.

“I consent to the publication of the results of the study provided the data is confidential—that is no identification can be made. I understand that my responses are anonymous and will be identified by number only.”

Signed: _______________

Date: _______________
A.6 Study 2 Questionnaire

The purpose of this questionnaire is to evaluate the usability of Inka with a view to improving it. The responses you provide will help us to understand the aspects of the system you are concerned about, and those that you are satisfied with. Your responses are anonymous and will be identified by number only.

Please fill in the comments sections to elaborate upon your answers.

1. I am satisfied with how easy it is to update my skills model.

<table>
<thead>
<tr>
<th>STRONGLY AGREE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>STRONGLY DISAGREE</th>
</tr>
</thead>
</table>

COMMENTS

2. I am satisfied with how quickly I can update my skills model.

<table>
<thead>
<tr>
<th>STRONGLY AGREE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>STRONGLY DISAGREE</th>
</tr>
</thead>
</table>

COMMENTS

3. I am satisfied with how quickly I can get a recommended task.

<table>
<thead>
<tr>
<th>STRONGLY AGREE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>STRONGLY DISAGREE</th>
</tr>
</thead>
</table>

COMMENTS
4. I am satisfied with how easy it is to get a recommended task.

| STRONGLY AGREE | 1 | 2 | 3 | 4 | 5 | 6 | 7 | STRONGLY DISAGREE |

COMMENTS

5. I am satisfied with the number of tasks I can work on at a time.

| STRONGLY AGREE | 1 | 2 | 3 | 4 | 5 | 6 | 7 | STRONGLY DISAGREE |

COMMENTS

6. I am satisfied with how quickly I can create a context snapshot.

| STRONGLY AGREE | 1 | 2 | 3 | 4 | 5 | 6 | 7 | STRONGLY DISAGREE |

COMMENTS

7. I am satisfied with how easy it is to create a context snapshot.

| STRONGLY AGREE | 1 | 2 | 3 | 4 | 5 | 6 | 7 | STRONGLY DISAGREE |

COMMENTS
8. I do not find making context snapshots distracting.

<table>
<thead>
<tr>
<th>STRONGLY AGREE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>STRONGLY DISAGREE</th>
</tr>
</thead>
</table>

COMMENTS

9. Overall, I am satisfied with how easy it is to use this system.

<table>
<thead>
<tr>
<th>STRONGLY AGREE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>STRONGLY DISAGREE</th>
</tr>
</thead>
</table>

COMMENTS

10. I feel comfortable using this system.

<table>
<thead>
<tr>
<th>STRONGLY AGREE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>STRONGLY DISAGREE</th>
</tr>
</thead>
</table>

COMMENTS

11. It was easy to learn to use the system.

<table>
<thead>
<tr>
<th>STRONGLY AGREE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>STRONGLY DISAGREE</th>
</tr>
</thead>
</table>

COMMENTS
12. The system has all the functions and capabilities I expect it to have.

| STRONGLY AGREE | 1 | 2 | 3 | 4 | 5 | 6 | 7 | STRONGLY DISAGREE |

COMMENTS

13. Overall, I am satisfied with this system.

| STRONGLY AGREE | 1 | 2 | 3 | 4 | 5 | 6 | 7 | STRONGLY DISAGREE |

COMMENTS

14. Please give any other comments you have below.

COMMENTS
Bibliography


251


[88] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.


257


258


