NextMove: Task Coordination in a Distributed Agile Development Environment

by

David K. M. Mak

B.A.Sc., University of British Columbia, 2004

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate Studies

(Electrical and Computer Engineering)

The University of British Columbia

September 2007

© David K. M. Mak, 2007
ABSTRACT

While task coordination has always been a challenge for software project managers, the emerging trend of distributed development adds a new dimension to the conundrum. This thesis describes NextMove, a tool which relieves the project manager from the mundane work of day-to-day task assignment to team members, as well as the heuristic processes behind NextMove. NextMove leverages common practices in software engineering and multi-criteria decision-making processes in order to continuously evaluate the pertinence of tasks that are currently available in the project and advise the team on what to do next as the project progresses. Two methods were used to validate the evaluation approaches behind NextMove: project simulations and experiment with human subjects. Simulation results show that teams that make use of NextMove’s heuristic process consistently post shorter project completion times than at least 80% of randomized executions of the same project. The experiment yielded less optimistic results, with conflicting results between different experiment sessions. However, important lessons were learned during the experiment and they open avenues for further research and improvement on NextMove.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
</tr>
<tr>
<td>GLOSSARY</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
</tr>
<tr>
<td>DEDICATION</td>
</tr>
<tr>
<td>CHAPTER 1 INTRODUCTION</td>
</tr>
<tr>
<td>1.1 SIGNIFICANCE</td>
</tr>
<tr>
<td>1.2 RESEARCH GOALS</td>
</tr>
<tr>
<td>1.3 ORGANIZATION OF THIS THESIS</td>
</tr>
<tr>
<td>1.4 CONTRIBUTIONS OF THIS THESIS</td>
</tr>
<tr>
<td>CHAPTER 2 CONTEXT AND PROBLEM STATEMENT</td>
</tr>
<tr>
<td>2.1 CHARACTERISTICS OF DISTRIBUTED AGILE DEVELOPMENT</td>
</tr>
<tr>
<td>2.1.1 Distributed Development</td>
</tr>
<tr>
<td>2.1.2 Agile Methods</td>
</tr>
<tr>
<td>2.1.3 Process Models</td>
</tr>
<tr>
<td>2.1.4 Team Structure</td>
</tr>
<tr>
<td>2.1.5 Difficulties in Agile Distributed Development</td>
</tr>
<tr>
<td>2.2 CONTEXTUAL MODEL AND PROBLEM DEFINITION</td>
</tr>
<tr>
<td>2.2.1 Scenario</td>
</tr>
<tr>
<td>2.2.2 Feature Model</td>
</tr>
<tr>
<td>2.2.3 Process Model</td>
</tr>
<tr>
<td>2.2.4 Project Model</td>
</tr>
<tr>
<td>2.2.5 Problem Definition</td>
</tr>
<tr>
<td>CHAPTER 3 REVIEW OF CURRENT STATE OF THE ART</td>
</tr>
<tr>
<td>3.1 CURRENT STATE OF THE ART IN COORDINATION METHODOLOGIES</td>
</tr>
<tr>
<td>3.1.1 Policy-based Task-Assignment Strategy</td>
</tr>
</tbody>
</table>

iii
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.2</td>
<td>Multi-Criteria Task Allocation</td>
<td>19</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Optimizing Task Allocation using Genetic Algorithms</td>
<td>20</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Summary</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>DESIGN CONSIDERATION</td>
<td>21</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Activity Theory</td>
<td>21</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Continuous Collaboration</td>
<td>22</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Group Awareness in Distributed Environment</td>
<td>23</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Summary</td>
<td>24</td>
</tr>
<tr>
<td>3.3</td>
<td>PROJECT MANAGEMENT TOOLS</td>
<td>24</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Microsoft Office Project</td>
<td>25</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Open Workbench</td>
<td>25</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Rally Agile Life Cycle Management</td>
<td>27</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Xplanner</td>
<td>27</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Summary</td>
<td>28</td>
</tr>
<tr>
<td>4</td>
<td>TASK COORDINATION STRATEGY</td>
<td>29</td>
</tr>
<tr>
<td>4.1</td>
<td>TASK PRIORITIZATION</td>
<td>29</td>
</tr>
<tr>
<td>4.2</td>
<td>TASK ALLOCATION</td>
<td>31</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Recommended Task List</td>
<td>36</td>
</tr>
<tr>
<td>4.3</td>
<td>BENEFITS OF METHODOLOGY</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>IMPLEMENTATION</td>
<td>37</td>
</tr>
<tr>
<td>5.1</td>
<td>WHY ECLIPSE?</td>
<td>37</td>
</tr>
<tr>
<td>5.2</td>
<td>NEXTMOVE ARCHITECTURE</td>
<td>38</td>
</tr>
<tr>
<td>5.3</td>
<td>STATISTICS AND EXTERNAL LIBRARIES</td>
<td>40</td>
</tr>
<tr>
<td>5.4</td>
<td>SERVER</td>
<td>40</td>
</tr>
<tr>
<td>5.5</td>
<td>USER INTERFACE</td>
<td>41</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Team Backlog View</td>
<td>41</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Personal Backlog View</td>
<td>44</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Team Activities View</td>
<td>45</td>
</tr>
</tbody>
</table>
## 5.5.4 Parking Lot View

- Page 46

## 5.5.5 Artifacts View

- Page 47

## 5.5.6 Log View

- Page 48

## 5.6 Design Discussion

- Page 49

### CHAPTER 6 SIMULATION

- Page 50

#### 6.1 Simulation Data

- Page 50

- 6.1.1 Project Data
  - Page 50

- 6.1.2 Team Member Data
  - Page 54

#### 6.2 Simulation Model

- Page 55

#### 6.3 Data Analysis

- Page 58

#### 6.4 Results

- Page 59

- 6.4.1 Scenario Ax
  - Page 59

- 6.4.2 Scenario Ay
  - Page 61

- 6.4.3 Scenario Bx
  - Page 63

- 6.4.4 Scenario By
  - Page 65

- 6.4.5 Scenario Cx
  - Page 67

- 6.4.6 Scenario Cy
  - Page 69

#### 6.5 Discussion

- Page 71

- 6.5.1 Observation from the Results
  - Page 71

- 6.5.2 Simulation Model Limitations
  - Page 72

### CHAPTER 7 EXPERIMENT

- Page 74

#### 7.1 Experimental Planning

- Page 74

- 7.1.1 Goals
  - Page 74

- 7.1.2 Experimental Units
  - Page 74

- 7.1.3 Experimental Material
  - Page 75

- 7.1.4 Tasks
  - Page 76

- 7.1.5 Hypothesis
  - Page 77

- 7.1.6 Procedure
  - Page 78
LIST OF TABLES

TABLE 1. RELATIVE IMPORTANCE OF TASKS .......................................................... 34
TABLE 2. NORMALIZED VALUES OF Table 1 .......................................................... 34
TABLE 3. TEAM MEMBERS’ RELATIVE JAVA PROGRAMMING ABILITIES ............... 34
TABLE 4. NORMALIZED RELATIVE COMPETENCIES FOR EACH TEAM MEMBER .... 35
TABLE 5. RANKING THE SKILLS-BASED SUITABILITY RATINGS OF THE TEAM MEMBERS ................................................................. 35
TABLE 6. FEATURE LIST FOR PROJECT A ......................................................... 51
TABLE 7. FEATURE LIST FOR PROJECT B ......................................................... 53
TABLE 8. FEATURE LIST FOR PROJECT B ......................................................... 54
TABLE 9. ORDER OF ASSIGNMENT OF TREATMENTS ....................................... 76
TABLE 10. SUMMARY OF EXPERIMENTAL RESULTS ....................................... 85
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Matrix depicting relationship between activities, features and tasks</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>Matrix depicting relationship between work products, features and tasks</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>UML Class Diagram of the Project Model</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Open Workbench General User Interface</td>
<td>26</td>
</tr>
<tr>
<td>5</td>
<td>Open Workbench Task Assignment Dialog</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>Rally Dashboards: Team Status</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>Xplanner Story View</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>Layer Diagram of NextMove</td>
<td>39</td>
</tr>
<tr>
<td>9</td>
<td>NextMove Team Backlog View with tasks coloured by their activities</td>
<td>41</td>
</tr>
<tr>
<td>10</td>
<td>NextMove Team Backlog View with tasks coloured by their features</td>
<td>42</td>
</tr>
<tr>
<td>11</td>
<td>NextMove Task Information Dialog</td>
<td>43</td>
</tr>
<tr>
<td>12</td>
<td>NextMove Manual Task Allocation Dialog</td>
<td>44</td>
</tr>
<tr>
<td>13</td>
<td>NextMove Personal Backlog View with the Context Menu</td>
<td>45</td>
</tr>
<tr>
<td>14</td>
<td>NextMove Team Activities View</td>
<td>46</td>
</tr>
<tr>
<td>15</td>
<td>NextMove Parking Lot View</td>
<td>47</td>
</tr>
<tr>
<td>16</td>
<td>NextMove Artifacts View</td>
<td>48</td>
</tr>
<tr>
<td>17</td>
<td>NextMove Log View</td>
<td>48</td>
</tr>
<tr>
<td>18</td>
<td>Process Flow Diagram of Project A</td>
<td>52</td>
</tr>
<tr>
<td>19</td>
<td>Process Flow Diagram of Project B</td>
<td>53</td>
</tr>
<tr>
<td>20</td>
<td>Process Flow Diagram of Project C</td>
<td>54</td>
</tr>
<tr>
<td>21</td>
<td>General Execution Flow of Each Simulated Team Member</td>
<td>56</td>
</tr>
<tr>
<td>22</td>
<td>Detailed Execution Flow within the “Find and Work on Task” Activity</td>
<td>56</td>
</tr>
<tr>
<td>23</td>
<td>Comparisons of Project Completion Times in Scenario Ax</td>
<td>59</td>
</tr>
<tr>
<td>24</td>
<td>Comparisons of Mean Feature Completion Times in Scenario Ax</td>
<td>60</td>
</tr>
<tr>
<td>25</td>
<td>Comparisons of Weighted-Mean Feature Completion Times in Scenario Ax</td>
<td>61</td>
</tr>
<tr>
<td>26</td>
<td>Comparisons of Project Completion Times in Scenario Ay</td>
<td>62</td>
</tr>
</tbody>
</table>
Activity: a guideline for a certain type of task. An activity modifies certain work products and may require roles with certain expertise to perform it.

Artifact: a piece of project-related document, source code or file which are produced or modified by a task. The scope and format of an artifact are defined by its parent work product.

Feature: a client-defined unit of functionality which adds value to the software system. Features are also known as user stories.

Process: an abstract model which defines the relationships between different activities and relationships between different work products.

Role: a set of responsibilities within the project. A team member may take on different roles as the project progresses.

Task: a unit of work which can be allocated to individual team members. The scope and actions of a task are defined by its parent activity.

Work Product: a template which defines the scope and format of a physical artifact.
ACKNOWLEDGEMENTS

As an alleged novel writer, perhaps I am expected to write something insightful or even witty in this acknowledgement section. However, I truly believe that nothing is more sincere than a few simple word of thanks to those who helped me along throughout my graduate school years.

To my supervisor Philippe, I thank you for giving me the opportunity to explore the world of research. Thank you for your generous support, without which my graduate school experience would not have been nearly as fulfilling and memorable. And I thank you for your patience and guidance throughout this long journey.

To all my friends in SEAL, Yvonne, Eve, Mandana, Larix, Steve, Agung, Davide, Jaana, thank you all for the great times. I will always remember our lab as a place full of happy memories and fun.

Big thanks to my friends Wilson, Sunny, Joyce and Pamela for your company and invaluable support since my undergraduate years.
Dedicated with Love.....

My parents, my sister and
my grandmother
Coordinating and allocating tasks are important elements of project management. Traditionally, the responsibility of managing resources falls to the project manager. However, as the project becomes more sophisticated, the project manager faces the problem of having to collect, analyze and disseminate a large amount of information during the course of the resource allocation process. Solutions to this problem often result in one of the two extremes: coordination according to heavy up-front planning and hierarchical team structure, or ad-hoc self-coordination. Neither type of solutions is optimal for a volatile environment of software projects. Up-front planning is too rigid and inflexible to respond to the constant and inevitable changes in a software project. Self-coordination, which leaves the individual team members to their own devices, may descend into chaos if it is not properly monitored, since the software project team needs to work as a tight unit at all times to ensure that various components are continuously and smoothly integrated.

Advances in electronic communications and the development of distributed collaborative tools make distributed software development possible. The term "working together" no longer applies solely to co-located teams. A modern project team may be distributed over a number of rooms in the same building, or over different buildings in the same city, or even over different cities around the globe.
In recent years, the emergence of agile development has attracted much attention and interest from the software development community. The Agile Movement helps reduce the reliance on heavy up-front planning and written artifacts. However, because many Agile practices depend on direct, face-to-face, informal communications, managing the dynamic and tactical activities of discovering, scheduling and allocation of tasks is difficult in a distributed environment. Agile teams need to find a way to dynamically manage a project backlog, which is a list of tasks to be done in a project, and attain a high level of collaboration while working in a distributed fashion.

1.1 Significance
Over the years, numerous tools and management systems have been created to help teams manage tasks in distributed projects. Some of these tools and systems leave the task coordination problem to the project manager, who has to manually assign tasks. Others try to automatically determine the “most optimal” task allocation sequence for each team member and force the team to follow those pre-determined schedules. Chapter 3 reviews the aforementioned types of tools and systems. They tend to focus on the procedural aspects of a project and therefore impose numerous restrictions on the team. Inevitably, users would try to circumvent the system as they see little benefit in locking themselves in a rigid and seemingly draconian system which offers little assistance when changes in the project occur.

Chapter 3 also reviews a number of proposed methodologies in modeling and decision-making to help facilitate task coordination. These methods fail to consider all of the different aspects in software development, such as the need for quick reactions to changes in circumstances, which is the key concept of agile development.
The work described in this thesis integrates common practices in software project management in order to devise an approach that would help the project manager handle the chores of day-to-day task coordination. The proposed approach aims to combine the quantitative and qualitative aspects of a software project in order to discover tasks that are most pertinent to the project’s success, as well as to rate the suitability of each team member to perform those tasks. To demonstrate the effectiveness of the proposed strategy, NextMove was developed as a proof-of-concept. The tool’s design makes use of concepts from computer-supported collaborative work (CSCW) in order to maximize benefit for the end-user while minimizing the amount of user-intervention.

1.2 Research Goals
The objective of this thesis is to assist the project manager in coordinating a distributed software development project which would result in improvements in the efficiency and job satisfaction of the development team. In summary, the goals of this thesis are:

- identify the advantages and disadvantages of current practices
- create a tool to assist the project manager in performing task coordination activities
- validate proposed approaches and processes with three metrics: total project completion time (TPCT), mean feature completion time (MFCT), weighted-mean feature completion time (WFCT)

1.3 Organization of this Thesis
This thesis consists of five major sections. Chapter 2 describes the background behind the task coordination problem. Relevant literature and the current state of project management tools are reviewed in Chapter 3. Chapter 4 presents the approach of evaluating task coordination
problems. Chapter 5 describes the architecture and user interface of NextMove, an implementation of the proposed methodology. Chapter 6 summarizes the results of simulations, which validates the effectiveness of the heuristic process as described in Chapter 4. Chapter 7 presents the results of the experiments in which the usefulness of NextMove was evaluated using human subjects. Lastly, conclusions and suggestions for future work are outlined in Chapter 8.

1.4 Contributions of this Thesis

My contributions in this thesis are:

- proposed a heuristics-based process to determine the most important tasks in a project at various points in time, as well as determine the most suitable person to perform those tasks
- implemented NextMove, which helps teams coordinate their tasks and increase project transparency
- validated the approach through simulations of real-life project situations
- evaluated the effectiveness of the prototype tool NextMove through testing with human subjects
- parts of the work of this thesis was published in three conferences ([1], [2], [3])
CHAPTER 2
CONTEXT AND PROBLEM STATEMENT

A software project poses as much of a managerial challenge as a technical challenge. While careful planning is essential to risk management and budgeting, it is equally important for a team to effectively monitor and respond to emergent changes quickly and effectively. Agile development methods reflect the recognition of the need to break away from the reliance on heavy, documentation-intensive, rigid up-front planning. Following the trend of globalization, software developers start to experiment with distributed development in the hopes of increasing competitiveness by reducing cost and leveraging the talents of software practitioners around the world. Unsurprisingly, the new world of distributed agile development adds a new dimension of complexity to software project coordination and poses a number of new problems. This chapter first discusses the various facets of distributed agile development, and how its task coordination problem is different. Then, the problem statement and model will be defined within the context discussed.

2.1 Characteristics of Distributed Agile Development
This thesis focuses on software development teams that are individually and geographically distributed (instead of teams that are composed by distributed sub-teams, each with its own team structure) and make use of agile development methodologies, more specifically, Scrum [4], Feature-Driven Development [5], eXtreme Programming [6], Lean [7] and Crystal [8]. Direct communications, which play an integral part in agile methodologies, are often made difficult by
the inadequacies of remote communication technologies, such as video conferencing. The realities of distributed development are preventing teams from taking full advantage of the strengths of agile methodologies, as discussed in the following sections.

2.1.1 Distributed Development

In recent years, distribution of software development teams has become an increasingly common practice in the software development industry. The motives behind such moves include [9]: taking full advantage of the global resource pool, localizing products so that they are better suited to different markets, reducing development time by “around-the-clock” development and because of the multinational nature of the organization. However, distributed software development teams often run into difficulties in coordinating their work, especially when they are using agile methodologies.

2.1.2 Agile Methods

Proponents of lightweight agile methodologies contend that agile methods are better suited to the fast pace of contemporary software projects than traditional plan-based development methods [10]. Agile methods put a much stronger emphasis on informal communication over formal written artifacts [11]. While direct, informal communications provide flexibility to the project, they are not suitable for information retention, which is sometimes necessary in order to exchange information within a distributed group.

One example of an agile methodology is the Scrum process, as coined by Ken Schwaber [4]. In the Scrum process, a team would keep a list of identified tasks, commonly known as a backlog. Each task is a short-term activity which should take no longer than several days to complete. The
backlog is continuously updated with new tasks added or irrelevant tasks removed as changes in the project occur. Tasks are prioritized and each team member would commit to a number of tasks. Short daily meetings, known as the daily Scrum or daily stand-up meeting, would be held throughout the course of the project in order for the project manager to collect feedback from team members. Scrum meetings serve two additional functions: team members get to know one another’s progress and gather information on one another’s work. Hence, they will know the needs for each team member to get his/her job done. After the Scrum, the project manager would try to resolve issues that were raised during the Scrum by allocating resources to perform the tasks on hand.

In the realm of feature-driven development, a feature is defined as a client-valued functionality that can be implemented within a specific (and usually short) timeframe [12]. Each feature is treated as an incremental unit of the system. Feature-driven development methodology focuses on features instead of tasks in planning because features are what customers understand [13]. After each milestone, the priorities of the planned features can be revisited by the customer and adjusted accordingly. The advantage of feature-driven development is that the project is divided into incremental and executable parts, hence enabling the organization to have early, partial releases.

2.1.3 Process Models

While an overly-detailed and rigid process model may prove to be detrimental to the agility (i.e. the ability to respond to changes) of the team, an abstract software development process model is useful as a communication tool to convey the general direction of the project. A process model describes the various activities in a project and their inter-dependencies, as well as any
documents or artifacts which are related to the activities. There are a number of languages and meta-models, ranging from data flow diagrams to Petri nets [14]. The work in this thesis focuses on development processes which are defined by the Software Engineering Process Meta-model (SPEM) [15].

2.1.4 Team Structure

Compared to project teams in traditional software development, agile software development teams are less hierarchical in nature. An agile team usually consists of a project manager (or project leader) and a number of team members of equal standing within the organization's structure. Agile software projects often employ the concept of roles. A role describes a set of activities performed and a set of artifacts owned [16] [17]. When a project team member fulfills a role, he or she is responsible for performing the activities and maintaining the artifacts that are included in the role. Each team member may take on different roles at various times or multiple roles simultaneously over the course of the project. For example, at the beginning of the project, a team member may take on the role of a requirements analyst; later on, that member would fulfill the role of implementer and developer of the user interface; towards the end of the project, the team member would become a tester and ensure certain functions are working properly. Hence, Agile project teams prefer generalists who are versatile over specialists who can perform only one type of task extremely well.

2.1.5 Difficulties in Agile Distributed Development

While agile methodologies and distributed development have their advantages, a number of issues are encountered where such problems do not exist in a traditional, co-located
environment. These issues are the root causes of the task coordination problem on which this thesis is focused.

Studies such as one by Herbsleb and Mockus [18] have shown that distributed development may indirectly introduce delay, as compared to same-site development. These delays may be attributed to the lower frequency of communications, difficulty in exchanging information with other team members, and lack of understanding of the work that distant colleagues are performing. These problems are not confined to teams that are globally distributed. Teams that are separated merely by walls within the same building may face similar communication problems.

Agile approaches such as Scrum processes often experience difficulties in distributed development environments [19]. For example, it is difficult to replicate the daily Scrum when a team is scattered across the globe. Logistics concerns such as finding a suitable time for meeting and limitations in technologies are the main drivers of such a conundrum. Hollan and Stornetta contend that video/audio conferencing technologies have not been able to emulate the effects of social presence in a face-to-face communication environment [20]. Current technologies in video conferencing and instant messaging do not provide the necessary level of interaction to facilitate face-to-face, informal communications, which in turn create and maintain working relationships. This is because these technologies do not fully conform to established social mechanisms, such as eye contact and tone of voice.
A flat hierarchical team structure that is often found in software development teams poses a number of issues in communication and information seeking. With the absence of informal face-to-face communication between team members, the project team is left with two options: to have the project manager collect all the information and allocate tasks to team members, or have the team members coordinate amongst themselves. Each approach has its own disadvantages.

Effective, timely assignment of tasks requires an experienced and skilled project manager. Before deciding on the allocation of tasks, the project manager would need to objectively assess each team member's skills and experiences, make sure that tasks will be completed on time, and prioritize the tasks. While an experienced manager can often make correct decisions using intuition and experience, such a task may prove to be overwhelming for new, inexperienced project managers. In addition, in the event that the project manager has to take a leave from the project, contextual information on allocation decisions would be lost.

When team members self-organize, they tend to choose tasks that they like or tasks that they know how to perform because they understand the context. However, the tasks that are chosen purely on personal preferences may not be the tasks that are most critical to the project’s success. In addition, since the chain of command in a software project is not always clear, team members may have difficulties in deciding whom they should talk to in order to communicate any problems that they encounter (e.g., one team member cannot continue his work because a modification is needed in another module for which another team member is responsible). Similarly, team members may not know whom to ask for information on a particular part of the system because each team member’s responsibilities are not always visible to all members. This
perceived lack of transparency in the project would lead to communication overhead and replication of work.

2.2 Contextual Model and Problem Definition

This section describes a scenario and the contextual model of the task coordination problem, to both of which the work of this thesis can be applied. The scenario and contextual model are based on the issues described in the previous sub-section.

2.2.1 Scenario

Consider a group of three students, Joe, Bob and David (no relation with the author of this thesis) who are developing a web application for travel agencies as a summer project. Prior to the start of the project, they decided to adopt some agile practices, such as feature-driven development, in order to beat their competitors to the market. Joe is known as a software development guru who is good at most areas (e.g., documentation, Java programming, testing). Bob has some experience in testing, but not much experience in Java. David has worked on several Java school projects before but has not taken the course on testing yet.

Throughout the summer, the team cannot meet in person because of a number of reasons: Joe is working as a co-op in Japan; Bob went home to Taipei for the summer; and David is staying in Vancouver. Originally, they designated Joe as the project manager. However, one month into the project, Joe soon found himself overworked because he needed to keep track of the progress of himself, Bob and David. Since Bob and David did not have much opportunity to communicate directly because of time differences (both Bob and David sleep and work during normal hours,
unlike graduate students), they rely on Joe to determine what to do next and allocate tasks to them. When Joe is busy with his job, sometimes he would post the pending tasks on the team’s private message board and let Bob and David work out who is doing what. It turns out that David and Bob tend to volunteer for tasks that they like better. Some tasks were left untouched for weeks, and others were done unnecessarily quickly – both Bob and David, albeit unknowingly, were in a race to perform same tasks that they liked in order to force the other person to perform the undesirable tasks. Hence, the team needs to improve its coordination: to find out which tasks are pertinent at the current time, and to find the most suitable person to perform those tasks.

2.2.2 Feature Model

At the planning phase of the project, a number of features would be collaboratively defined by the customer and the project team. A feature consists of two major attributes:

- priority: the importance of the feature, which is also related to the order in which the features should be completed
- size: the size of features in terms of function points [21]

2.2.3 Process Model

In the context of this thesis, the process model serves as the template for the tasks and artifacts in a project. In a process model, there are a number of inter-related activities. Each activity represents a step towards attaining the overall goal of the process. An activity may be associated with one or more work products as inputs or outputs. An activity has a number of attributes:
• workflow relationships: how the activity is related to other activities in the workflow of the process
• time required: the time required to perform this activity, expressed as a percentage of the total execution time of the process
• required skills: the skills required to perform the activity and their relative importance

In addition, a number of work products are defined within a process model to serve as guidelines to the types of artifacts to be created in a project. In each work product model, the work product's relationships with one or more activities and dependencies on other work products are defined.

2.2.4 Project Model

The feature model and the process model combine to create the project model. For each feature, an instance of a process is created. A process instance consists of a set of tasks (instances of activities) and a set of artifacts (instances of activities). The relationship between features, activities and tasks can be more clearly illustrated in the following matrix:

<table>
<thead>
<tr>
<th>Feature/Activity</th>
<th>Activity 1</th>
<th>Activity 2</th>
<th>Activity 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature A</td>
<td>Task 1A</td>
<td>Task 2A</td>
<td>Task 3A</td>
</tr>
<tr>
<td>Feature B</td>
<td>Task 1B</td>
<td>Task 2B</td>
<td>Task 3B</td>
</tr>
<tr>
<td>Feature C</td>
<td>Task 1C</td>
<td>Task 2C</td>
<td>Task 3C</td>
</tr>
</tbody>
</table>

Figure 1 – Matrix depicting Relationship between Activities, Features and Tasks
The relationship between work products, features and artifacts are similar:

<table>
<thead>
<tr>
<th>Feature/Work Product</th>
<th>Work Product 1</th>
<th>Work Product 2</th>
<th>Work Product 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature A</td>
<td>Artifact 1A</td>
<td>Artifact 2A</td>
<td>Artifact 3A</td>
</tr>
<tr>
<td>Feature B</td>
<td>Artifact 1B</td>
<td>Artifact 2B</td>
<td>Artifact 3B</td>
</tr>
<tr>
<td>Feature C</td>
<td>Artifact 1C</td>
<td>Artifact 2C</td>
<td>Artifact 3C</td>
</tr>
</tbody>
</table>

Figure 2 – Matrix depicting Relationship between Work Products, Features and Tasks

2.2.4.1 Task Model

The work in this thesis supports two types of tasks in a software project: process-based tasks, which are created from the combination of features and the process model; and ad-hoc tasks, which are tasks that are not regularly performed. Although ad-hoc tasks may not directly affect the final deliverables or intermediate artifacts, they are nonetheless critical to the completion of the project. An example of an ad-hoc task is the setup of a piece of test equipment. Both process-based and ad-hoc tasks have the following attributes:

- workflow relationship: the preceding tasks that need to be done before the task can be executed, as well as subsequent tasks that depend on the task. The workflow relationship of a process-based task is inherited from the activities relationships
- effort/time estimation: the amount of time required to perform the task; for process-based tasks, the effort estimation for process-based tasks are calculated as a function of feature size
- task readiness: whether the task is ready to be executed (e.g., the tasks on which it depends has been completed)
• required skills: skills required to perform the task, and their relative importance; for process-based tasks, this attribute will be inherited from the activity model. These required skills will be compared with team members' skills to determine the most suitable task performer.

A process-based task would have a number of additional attributes:

• activity: the activity (goal) that the task would (partially) satisfy

• feature: the feature that the task is related to

• artifact relationship: input/outputs of the task

2.2.4.2 Artifact Model

In the NextMove project model, artifacts refer to both textual and graphical documentation, such as use case, or software code. An artifact is modeled with the following attributes:

• work product: the class of work product that the artifact belongs to

• state: the current status of the artifact; an artifact's status is modified by different tasks

• artifact dependencies: other artifacts that may impact the artifact when they are changed

2.2.4.3 Team Member Model

A project team consists of one or more team members. Each team member has different sets of skills and attributes. These skills and attributes include:

• skills: skills and experience in performing the aforementioned types of tasks, such as programming skills, and documentation skills

• familiarity with artifact: as the project progresses, certain team members would be more familiar with certain sets of artifacts
• workload: the amount of work the team member has taken on at that time

This approach to modeling enables a distinct separation between the abstract process model and the physical project. According to van der Hoek, Redmiles et al. [22], formal processes work best when they are used to provide high-level guidance and support rather than to impose a strict sequence of activities on the team. The separation of process and project allows for more flexibility in the execution of the project.

The UML object diagram in Figure 3 illustrates the relationship between processes, activities, tasks, and artifacts.
2.2.5 Problem Definition

The establishment of the NextMove project model allows the task coordination problem to be described in a more formal manner. In essence, the project manager needs to arrange a set of tasks \( T \), which consists of \( n \) tasks, into an \( n \)-tuple that ranks the tasks from the most to least important at an instance of the project. The arrangement of the \( n \)-tuple is constrained by the attributes of the feature, task, and artifact models. Then, for each task \( t_n \), each with a set of requirements \( R \), from the task model, the project manager needs to arrange the team members \( M \) into another \( n \)-tuple. The ranking would be produced according to their attributes \( A_m \), from the team member model. At the end, the two \( n \)-tuples would be combined to determine the most pertinent task for each team member.

Furthermore, any method that successfully resolves this problem must also satisfy the following criteria:

- communicable: the task-allocation process should be clear, so that team members understand the rationale behind each decision. This is especially important for a distributed team, because the project manager may not be available at all times to respond to queries from team members
- non-intrusive: the method should not impose a strict course of action on each team member. Team members should be allowed to use their own judgment in organizing their tasks according to the current situation
- flexible: the method should be able to accommodate both process-based tasks and ad-hoc tasks
- allow feedback: allow team members to be involved in the task-coordination process
CHAPTER 3
REVIEW OF CURRENT STATE OF THE ART

This chapter summarizes and reviews the current state of the art of task coordination and is divided into three parts. First, research and methodologies on the task coordination problem will be reviewed. This is followed by a summary of literature that is relevant to the design of NextMove. Finally, since NextMove is billed as a project management tool, it is useful to review some of the currently available project management tools.

3.1 Current State of the Art in Coordination Methodologies

Although there are a number of studies on task delegation on multiple-member teams, there have been surprisingly few that focus specifically on software development or distributed projects. This section summarizes three types of methodologies that focus primarily on the subject of task allocation.

3.1.1 Policy-based Task-Assignment Strategy

Shim, Lee et al. [23] propose a model-based methodology that allows modeling of task assignment policies. The proposed methodology makes use of UML-based meta-models to describe the four elements that are crucial to task assignment: organization, process, work products, and project. These models hold information on team structure policies, task assignment, artifacts and project state, respectively. In order to test and demonstrate the usefulness of the proposed methodology, a prototype has been implemented. A study on the
effectiveness of the prototype showed a significant reduction in request queuing time (i.e. time between request initiation and request resolution). Shim, Lee et al. contend that their policy-based methodology is adaptable to different development environments, as the policy models are extensible. However, there are some omissions in the model, such as the definition of workflow and the relationships between tasks and artifacts. In addition, the model did not allow for changes in policies when emergent situations arise during the course of the project’s execution.

3.1.2 Multi-Criteria Task Allocation

Shen, Tzeng et al. [24] proposed a multi-criteria task assignment model within workflow management systems. The model describes qualitative measures, such as individual capabilities, using linguistic scales and those measures are quantified by fuzzy numbers. The evaluation procedure assesses the suitability of a worker for a task using four criteria: capability, work load, similarities of tasks (between the task being evaluated and tasks previously performed by worker), and relationship between team members (for tasks which require more than one performer). The ratings of all the evaluated criteria are summed to produce an overall score for each candidate task performer. The multi-criteria task allocation methodology offers a partial solution to the second part of the task coordination – how to find the suitable candidates for each task. However, the methodology assumes that the priority of the tasks has been pre-determined, or the workflow is sequential in a way that it is obvious what the next task that needs to be performed is. Zhang, Cao and Zhang [25] also proposed a similar approach to improve task distribution in workflow management systems.
3.1.3 Optimizing Task Allocation using Genetic Algorithms

Duggan, Byrne et al. [26] propose an optimization method for task allocation during the construction phase of software development. This method makes use of multi-objective evolutionary algorithms (MOEA) and genetic algorithms (GA). Duggan, Byrne, and Lyons contend that a software project has a number of goals and they sometimes conflict with one another (e.g., minimization of defects vs. minimization of time to release). MOEA helps the project manager by balancing various objectives and generating sets of optimized schedules for each team member. While this approach is effective for estimating costs and schedules, problems may arise when it is used to perform task allocation during the execution of the project. As task-coordination and allocation problem are treated as static optimization problems, emergent changes are not taken into account. Moreover, since the operations in task allocation using MOEA are complex and hard to understand, it is up to the project manager to convince team members that following a rigid, computer-generated schedule is in the best interest of the project. Chang and Christensen [27] also proposed a similar, genetic algorithms based model for managing task schedules.

3.1.4 Summary

Each of the aforementioned methodologies represents a type of task coordination strategy, each with its advantages and disadvantages. Shim, Lee et al. suggested that task allocation decisions should be based upon a set of predefined policies. While their solution is relatively easy to implement (compared to the other two methodologies), it is also inflexible in terms of dealing with changes during the course of the project, as the policies may not account for all possible scenarios that may occur in the project. On the other hand, Shen, Tzeng et al. proposed the use of
multi-criteria decision mechanisms to resolve the task allocation problem, but did not address the question of task prioritization. The task allocation optimizer as introduced by Duggan, Byrne et al. treats the task coordination problem as a static analysis problem, which may not be suitable in the highly dynamic world of software development. The work of this thesis attempts to draw from the strengths of the reviewed work and address the weaknesses of the prior arts.

3.2 Design Consideration
This sub-section reviews the concepts and framework that were taken into consideration as NextMove and its underlying heuristics process were designed.

3.2.1 Activity Theory
Barthelmess and Anderson [28] analyze the current state of the art in software development environments in three aspects of collaborative work: subjects (e.g., roles), objects (e.g., tasks), and community (e.g., teams). They classified collaborative environments into three levels: coordinated, cooperative and co-construction.

Collaboration on a coordinated level usually has team members follow a rigid and pre-defined sequence of work, and work is organized to lower the need for interpersonal collaboration. On the cooperative level, team members have to communicate with each other in order to accomplish tasks which are interdependent. Co-construction implies that work practices are repeated and ways of accomplishing a task (i.e., the process) are being constantly refined. Barthelmess and Anderson contend that the three levels constitute the instantaneous status of an activity, and the status can change dynamically during the course of the project.
Barthelmes and Anderson argue that a large part of the problems in contemporary software engineering is concerned with communications. The pace of change in the software development world does not allow the development of common grounds which allow practitioners to effectively communicate with each other. Hence, it is important to have other means of facilitating communication among team members. Barthelmes and Anderson contend that most process-centred management tools are focused solely on activities on the coordinated level and overlook the aspect of collaboration. In this way, they are merely tools that rigidly assign roles to team members.

Barthelmes and Anderson propose that tool developers need to incorporate communication support into management tools. Team members should be able to select a sequence of actions instead of being "told" what to do by the environment. In addition, the environment should allow team members to convey explanations or context about a task or an artifact.

3.2.2 Continuous Collaboration

Van der Hoek, Redmiles et al. [22] contend that existing software engineering tools, such as configuration management systems, put limitations on human activity and creativity, because of the tools' reliance on the formal process. A process-based tool that coordinates tasks according to predefined processes fails to support the informal or ad-hoc practices which are essential in software development. On the other hand, although CSCW tools help team members gain awareness of one another's activities and facilitate self-coordination, increase in the scale of the project often leads to information overload.
Therefore, van der Hoek, Redmiles, and Dourish attempt to combine the strengths of both process-based and CSCW approaches and propose a new approach called continuous collaboration. For example, mechanisms are present to inform team members of the others' activity that might be relevant to them in order to create a sense of awareness. Formal checkpoints and measures would be used to provide a high-level guideline on the direction of the project. Van der Hoek, Redmiles, and Dourish contend that current software engineering tools are not capable of effectively supporting continuous collaboration. Hence, developers have to construct tools such that they do not impose a certain mode of collaboration onto the users.

3.2.3 Group Awareness in Distributed Environment

Gutwin, Schneider and Penner [29] conducted a study on the dynamics of a distributed software development team, with focus on how team members maintain awareness of one another’s work over remote distances. According to Dourish and Bellotti [30], group awareness is defined as the understanding of the others’ activities and is essential for any co-located or distributed group to collaborate successfully.

Through interviews with developers involved in three distributed open source projects (NetBSD, Apache server, Subversion), they observe that the project teams relied on both formal (e-mails on the project mailing list) and informal written communications (IRC chat) to maintain awareness of what others are doing. The effectiveness of coordination and collaboration of the three projects are shown by the success of the resulting product. However, they note that the nature of open source projects may make the results of their study not directly applicable to all distributed projects. The projects had flexible schedules and did not have to face the pressure of releasing a product at a tight timeframe; because of the voluntary nature of open source projects,
members who are not able to maintain awareness using the teams’ methodology would drop out of the project.

Nonetheless, Gutwin, Penner, and Schneider conclude that some principles found in the study can be applied on a broader context. A distributed team should provide different venues for verbal communication, which is essential for maintaining group awareness. Also, the study found overhearing (i.e. observing discussions without actively participating in them) a useful means of gathering awareness information. Gutwin, Penner, and Schneider suggested that distributed collaboration tools, such as version control systems and messaging systems, should be designed to better capture group awareness information, reduce information seeking time for users, and increase the transparency of the project.

3.2.4 Summary

The aforementioned works on collaboration provided guidelines for NextMove’s general design. NextMove is designed to provide flexibility such that team members will be able to select a course of action instead of having a tool dictating their actions. The user interface will explain the reasons for certain tasks to be recommended. NextMove takes advantage of formally-defined process models to provide guidance on the project. Finally, measures are implemented to convey group awareness information (e.g., who is working on which task) to the team members in order to increase project transparency.
3.3 Project Management Tools

There are numerous project management tools currently on the market. This section reviews general-purpose project management tools (e.g., Microsoft Office Project) as well as software development project specific tools and focused on the aspect of task coordination and tracking.

3.3.1 Microsoft Office Project

Microsoft Office Project [31] is one of the better known general purpose project planning and management tools. Its latest version, Project 2007, emphasizes project scheduling, presentation of project information (through charts and diagrams), and provides guidance on setting up projects. In Project 2007, tasks are assigned by the project manager. Team members and the project manager exchange task information (e.g., current progress) through the page created for the task.

3.3.2 Open Workbench

Open Workbench [32] is seen as an open-source alternative to Microsoft Office Project. Its general view shows the list of tasks and their schedules as a Gantt chart, as well as a list of resources (e.g. labour, equipment, material) and their schedules in the corresponding times. Figure 4 below is an overview of Open Workbench's interface.
In Open Workbench, tasks are created, prioritized and linked (i.e. defining the relationships between tasks) manually. The tasks are then assigned to resources and the estimated time for the team member to complete the task is entered, as shown in Figure 5.

Figure 4 – Open Workbench General User Interface

Figure 5 – Open Workbench Task Assignment Dialog
Users are able to look at the performer of each task by viewing the individual task information dialog; however, there is no quick overview of each team members’ current work in progress or past work.

### 3.3.3 Rally Agile Life Cycle Management

Rally Agile Life Cycle Management suite [33] ("Rally") is tailored towards agile software development teams. It incorporates a number of agile practices, such as backlog and user stories. In Rally, tasks and user stories are created, prioritized, and assigned manually. In the team status view, users would be able to see the list of tasks with no owner, as well as the task lists of each individual team members (Figure 6). For each task listed as owned by a team member, there is a state indicator which displays the current state of the task (e.g., defined, in progress, completed).

![Figure 6 – Rally Dashboards: Team Status](image)

### 3.3.4 Xplanner

Xplanner [34] is an open source project planning tool for large eXtreme programming teams. The tool focuses on task tracking and increasing project transparency. Tasks which belong to the same user story are listed and their progress are indicated with coloured progress bars (Figure 7). The progress of user stories within an iteration are tracked and displayed in a similar way. Unfortunately, due to technical difficulties, the author of the thesis was unable to perform an in-
depth review in the task coordination and allocation aspect of the tool. In Xplanner’s documentation, there is no mention of guidance on task coordination.

![Xplanner Story View](image)

**Figure 7 – Xplanner Story View**

### 3.3.5 Summary

The aforementioned tools provide varying levels of progress and task tracking, with examples including the Open Workbench’s graphical project overview and the iteration and story view of Xplanner. However, they are lacking in support or guidance for task prioritization or allocation. NextMove does not intend to replace these tools, but to supplement them by filling in the missing area of support.
CHAPTER 4
TASK COORDINATION STRATEGY

This chapter is dedicated to describing the strategies behind the task recommendation process performed by NextMove. The process involves two steps. The first step is to prioritize currently available tasks. Then, a suitability ranking of team members is produced for each available task. This process is continuously executed to produce a personalized recommended task list for each team member as circumstances change over the course of the project.

4.1 Task Prioritization
Throughout the course of the project, the team backlog will contain a number of tasks to be performed. The task-prioritization process ranks them according to four criteria: schedule, artifact relationship, feature priority and workflow dependencies.

Given a task, \( t \), that would modify an artifact \( a \) into state \( s \) at the time of \( t \)'s completion, \( t \)'s priority score in terms of artifact relationship depends on the number of other tasks that require \( a \) to be at state \( s \) in order to start execution. In addition, when a team member is waiting to execute a task that requires \( a \), he or she can request \( a \) to be changed into state \( s \). Hence, the priority score (PS) of a task, in terms of artifact relationships, is derived by:

\[
PS_{\text{artifact}} = \beta_{\text{tasks}}R_{\text{tasks}} + \beta_{\text{team}}R_{\text{team}}
\]

where \( R_{\text{tasks}} \) is the number of tasks that require the artifacts produced by \( t \), and \( R_{\text{team}} \) is the number of requests to work on the artifact from the team. \( \beta_{\text{tasks}} \) and \( \beta_{\text{team}} \) are weight factors that
can be adjusted according to the needs of different environments. The artifact relationship factor gives team members a way to provide feedback on what needs to be done, from their point of view.

For a set of sequentially related tasks, critical-path analysis (CPA) [35] can be performed to estimate the minimum amount of time needed to complete the set of tasks, as well as to identify the tasks that are critical to the timely completion of all tasks. Before calculating the priority scores in terms of schedule for each task, CPA is performed on groups of tasks belonging to the same feature. The analysis determines the earliest and latest start time of the task, relative to the start time of the feature. In turn, the schedule priority score of each task is determined by:

\[
P_{schedule} = \frac{1}{1 + (LST - EST)} + \frac{T_{PD} - LST}{T_{PD}}
\]

where LST and EST denote the latest and earliest start time of the task, respectively, and T_{FD} is the total time required to complete the feature. The first term of the equation determines the degree of criticality of the task. The second part of the equation represents the degree of urgency of the task. Hence, a task that is on the critical path and has to be performed soon will be rated the highest.

Feature-driven development and frequent releases are two important concepts in an agile development environment. In an agile project, a number of features (also known as user stories) are defined. A feature often represents a functionality that is useful for the user. A team schedules the completion of a certain number of features in each release. Hence, a customer can frequently evaluate the progress of the project and make requests for changes early. Feature priority is an important consideration in the task-prioritization process. It is modeled as follows:
where $F_{task}$ denotes the relative priority of the feature for which the task is assigned. The value of $F_{task}$ is directly proportional to the importance of the feature.

The workflow dependencies category is added to make sure that tasks that are ready to be executed are recommended over tasks that are not yet ready. The formal expression for this category is

$$E_{task} \rightarrow (PS_{\text{readiness}} = PS_{\text{max}})$$

where $E_{task}$ is the Boolean expression for whether or not the task is ready, and $PS_{\text{max}}$ is the maximum rating possible in the task-prioritization process. Hence, when a task is ready to be executed, it is always rated higher than tasks that are not ready.

The priority scores of a task are aggregated using a weighted sum.

$$PS_{\text{final}} = \lambda_{\text{artifact}} PS_{\text{artifact}} + \lambda_{\text{feature}} PS_{\text{feature}} + \lambda_{\text{schedule}} PS_{\text{schedule}} + PS_{\text{readiness}}$$

where the $\lambda$ values are the weights of each prioritization category, and can be adjusted by the team as different situations call for different emphasis. For example, the feature priority category would be weighted higher than the others if a team wants to make sure that the important features are completed on time for the next release.

### 4.2 Task Allocation

As described by Duggan, Byrne and Lyons [26], task allocation is a multiple objective problem (please see Chapter 3). The goals of the task-allocation process include the following:
• locate the most skilled person for the task: in theory, a person whose strengths match those that are required by the task will perform the task most efficiently

• balance workload: in a large project, the team will work more efficiently if tasks are distributed among the team members instead of overburdening a minority of members who excel in many fields

• minimize overhead: for tasks that require a high degree of familiarity with the related artifacts, it is better to locate a team member who has previous experience with the artifact than to force another member to spend time familiarizing himself or herself with the artifact before performing the task

When allocating tasks, project managers need to strike a balance among these goals. Reaching a desirable solution to this problem requires years of experience, intuition and rational thinking. In addition, there may not be a team member whose skill set is perfectly matched to the requirements of the task. In this case, the team member who satisfies the most important requirements of the task would become the most desirable person to execute the task.

The analytic hierarchy process (AHP) [36] is used as part of a conflict-resolution strategy in the task-allocation process. AHP assists decision makers in modeling a multi-criteria evaluation problem in a structured way. The advantage of AHP is that it can evaluate qualitative factors (e.g., team member skills) alongside quantitative factors (e.g., team member workload) in the same process.
AHP can be used to determine which team member is most suitable for a task in terms of skills, workload and familiarity with certain parts of the system. In the AHP, the relative importance between a pair of attributes is expressed in a 9-point scale. The following example with fictional figures illustrates how AHP is used in the proposed approach.

Coming back to the scenario in section 0, Joe, Bob and David would like to know the most suitable person (and the next best candidates if the most suitable person is not free) for the task of architectural design for the file transfer functionality.

Table 1 illustrates the skills required to perform the task and each skill’s relative importance to the task’s successful completion. The number in each cell should be interpreted as “the importance of <attribute name of the row> as compared to the importance of <attribute name of the column>.” For example, the cell in row 1, column 2 of the table should be read as “the importance of Java programming skills as compared to the importance of documentation skills.” The values in Table 1 are normalized in Table 2 in order to obtain the weight of each requirement, $\alpha$.

Table 3 shows the team member’s relative competencies in Java programming. Those values are also normalized in a fashion similar to that in Table 2 in order to produce each team member’s overall competency in Java ($r$ in the following equation). In turn, the documentation and design skills of Joe, Bob and David are compared and normalized likewise. The result of the normalizations is shown in Table 4.

The skill-based suitability ratings of each of the team members are calculated as follows:

$$SR_n = \alpha_{Java} r_{n,Java} + \alpha_{Documentation} r_{n,Documentation} + \alpha_{Design} r_{n,Design}$$
where $r_{n,\text{skill}}$ is the normalized relative value of team member n’s ability in a particular skill. The skill-based ratings of the three team members are ranked in Table 5.

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>Documentation</th>
<th>Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>1.00</td>
<td>3.00</td>
<td>5.00</td>
</tr>
<tr>
<td>Documentation</td>
<td>0.33</td>
<td>1.00</td>
<td>3.00</td>
</tr>
<tr>
<td>Design</td>
<td>0.20</td>
<td>0.33</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 1. Relative importance of tasks

<table>
<thead>
<tr>
<th></th>
<th>Java</th>
<th>Documentation</th>
<th>Design</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>0.65</td>
<td>0.69</td>
<td>0.56</td>
<td>0.63</td>
</tr>
<tr>
<td>Documentation</td>
<td>0.22</td>
<td>0.23</td>
<td>0.33</td>
<td>0.26</td>
</tr>
<tr>
<td>Design</td>
<td>0.13</td>
<td>0.08</td>
<td>0.11</td>
<td>0.11</td>
</tr>
</tbody>
</table>

Table 2. Normalized values of Table 1

<table>
<thead>
<tr>
<th></th>
<th>David</th>
<th>Bob</th>
<th>Joe</th>
</tr>
</thead>
<tbody>
<tr>
<td>David</td>
<td>1.00</td>
<td>0.33</td>
<td>0.20</td>
</tr>
<tr>
<td>Bob</td>
<td>3.00</td>
<td>1.00</td>
<td>0.33</td>
</tr>
<tr>
<td>Joe</td>
<td>5.00</td>
<td>3.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 3. Team members’ relative Java programming abilities
In order to take into account workload, an additional factor must be included in the analysis. The workload factor from above can be interpreted as the inverse of how critical the task is. If a task is critical to the project and has a strict skill requirement (i.e., it cannot be performed without certain skills), the workload factor would have a lower rating relative to other skills.

Likewise, each team member’s familiarity with a particular part of the project is evaluated as one of the attributes in the analysis. Overhead due to context switching is incurred when the team member is unfamiliar with the sub-system that he/she is working on and needs time to acquaint him/herself with the artifacts. Hence, a team member who has previously worked on the same part of the system is rated more highly than a team member who has no previous experience with it.
4.2.1 Recommended Task List

The end results of the task-prioritization process and the task-allocation process are combined to produce a recommended task list for each team member. The recommended task list would be sorted by the overall recommendation rating, RR, for each task, calculated by:

\[ RR_{overall} = \alpha_{TP} RR_{TP} + \alpha_{TA} RR_{TA} \]

where RR_{TP} and RR_{TA} are the recommendation ratings produced from the task prioritization, and \( \alpha_{TP} \) and \( \alpha_{TA} \) are constants that determine the importance of task-specific priority ratings and team-member-specific suitability ratings.

4.3 Benefits of Strategy

The strategy proposed in this chapter assists project managers in coordinating tasks by aggregating and analyzing different types of input in an organized, rational fashion. The following features make the NextMove framework an attractive option for project managers:

- understandable: compared to optimization algorithms such as multi-objective evolutionary algorithms, AHP is much more transparent and understandable
- choice of action: team members are more inclined to make use of a system that allows them to choose from a list of recommended tasks based on their intuition and preference, rather than a system that imposes a strict course of action
- flexibility: weights and constants in the calculation can be adjusted to accommodate changes of team strategies
- allowance of feedback: criteria such as artifact dependencies allow team members to provide feedback and be involved in the task-coordination process
CHAPTER 5
IMPLEMENTATION

NextMove is an integral part of the work of this thesis. It is a collaborative work assistant that is designed to help distributed teams coordinate their tasks and provide team members with an extensive overview of the team’s activities. It makes use of the heuristics-based evaluation process described in Chapter 4 to provide guidance on which task is currently most pertinent to the project’s success. In addition, NextMove’s user interface is designed to augment the information finding abilities of the team; hence team members can more easily gain awareness of the rest of the team’s activities. This chapter documents the design decisions, features and functionalities of NextMove, as well as how its features will help distributed teams in coordinating their work more efficiently.

5.1 Why Eclipse?
According to Grudin [37], collaborative group applications often fail because of the perceived lack of benefit of using the tool in the light of the extra work required. Hence, NextMove aims to minimize the extra effort required in order to take advantage of the tool’s features. NextMove’s integration with a popular integrated development environment would help reduce overhead, since users will not need to leave their work environment in order to make use of the tool (i.e., having switched from their development environment and start an external tool).

The Eclipse Project is an open-source project which aims to provide a vendor-neutral open development platform and application frameworks for building software. A number of
programming languages, including Java, C++, and PHP are supported by Eclipse natively or by projects which extend the Eclipse Platform. The Eclipse Platform is written in the Java language, can be deployed in different types of development workstations such as Linux, Solaris, Mac OS X and Windows and is extensible through its extension architecture. Over the years, a community of developers created a large variety of extensions with applications ranging from code generation to project management. Eclipse was chosen as the first deployment platform for NextMove for the following reasons:

- widespread adoption: Eclipse is one of the most popular Java integrated development environments (IDE)
- integration with users’ workspace: as a plug-in, NextMove can integrate seamlessly into the Eclipse IDE, avoiding the overhead of requiring team members to open a separate application
- collaboration with other plug-ins: NextMove may work with other Eclipse plug-ins, such as version control systems and bug tracking systems to further automate the process of information gathering
- platform independent: the Eclipse Platform provides an abstraction layer between the extensions and different operating systems. Hence, NextMove may be used in different development environments

5.2 NextMove Architecture

NextMove is implemented as a distributed client-server system, with the client implemented as an Eclipse extension and the server as a Java application. The system consists of a database backend, a server, and one or more clients.
NextMove is written primarily in Java, and a MySQL database is used for persistent storage. The clients and the server communicate through the Java remote method invocation (RMI) protocol. The client, implemented as an Eclipse extension, serves as the user interface. As the user performs actions on the user interface, the server retrieves the relevant information from persistent storage and performs the necessary calculations (e.g., generation of the recommended tasks list). The server will then notify all clients of updates through RMI callbacks. The layer diagram below illustrates the relationship between the various high-level modules of the system:

![Layer Diagram of NextMove]

Figure 8 – Layer Diagram of NextMove
5.3 Statistics and External Libraries

The current version of NextMove consists of about 13,000 lines of code (excluding the external module for executing the experiments as described in Chapter 7). There are approximately 140 classes placed inside 31 packages. Approximately 950 methods were implemented.

Apart from the Standard Widget Toolkit (SWT) [38] libraries provided by the Eclipse Platform, NextMove also makes use of two external libraries:

- MySQL® Connector/J [39]: for connecting to the MySQL database backend
- JAMA : a Java Matrix Package [40]: for calculations in the analytic hierarchy process (please see section 4.2)

5.4 Server

The server is responsible for handling client requests, generating recommended task lists, and storing persistent project information. When the team initializes a project on the server, the server will load a SPEM-compliant XML file, which contains process information, and other project data files in order to generate the initial set of tasks and team member information. The logic component generates recommended task lists as the state of the project changes (e.g., a team member indicated that a task is completed, new tasks are added to the project), and the server notifies all active clients. The server only generates and sends information that the user is currently viewing in order to reduce processing load on the server and network traffic. In addition, the server will respond to team members’ requests for information (e.g., other team members’ backlogs, information on tasks or artifacts).
5.5 User Interface

In the Eclipse Platform, the workbench contains a collection of workbench windows. Each workbench window contains one or more pages, and each page contains a collection of editors and views. A view is typically used to navigate a hierarchy of information, open an editor, or display properties for the active editor. [41] A combination of five views forms the framework of NextMove’s user interface. The names of team members, tasks and artifacts mentioned in the following sub-sections are related to the scenario described in section 0.

5.5.1 Team Backlog View

The team backlog view displays the list of tasks that are currently available (i.e. neither completed nor adopted by any team member) in the project. The tasks on the list are ranked according to the heuristics-based approach as described in Chapter 4. Figure 9 below is a screenshot of the team backlog view.

![Team Backlog View](image)

*Figure 9 – NextMove Team Backlog View with Tasks Coloured by their Activities*
The left-most column contains the names of the tasks on the list, and can be coloured according to the task’s activity (Figure 9 above) or the features that the tasks belong to (Figure 10 below). The colours of each activity or feature can be viewed by bringing up a pop-up legend. The middle column indicates the status of the tasks. When a task’s status column is green, it indicates that all dependencies of that task are satisfied. Red in the status column indicates that another task has to be completed or started before the task is ready to be executed. The rating column explains the rationale behind the rankings. Each component in the horizontal stacked bar-graph indicates the task’s rating in each of the categories.

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototyping for Train Reservation</td>
<td></td>
</tr>
<tr>
<td>Gather Requirements for Ship Reservation</td>
<td></td>
</tr>
<tr>
<td>Test and Fix Bugs for Hotel Reservation</td>
<td></td>
</tr>
<tr>
<td>Implement Components for Flight Reservation</td>
<td></td>
</tr>
<tr>
<td>Test and Fix Bugs for Flight Reservation</td>
<td></td>
</tr>
<tr>
<td>Prototyping for Ship Reservation</td>
<td></td>
</tr>
<tr>
<td>Implement Components for Train Reservation</td>
<td></td>
</tr>
<tr>
<td>Implement Components for Bus Reservation</td>
<td></td>
</tr>
<tr>
<td>Test and Fix Bugs for Train Reservation</td>
<td></td>
</tr>
<tr>
<td>Test and Fix Bugs for Bus Reservation</td>
<td></td>
</tr>
<tr>
<td>Implement Components for Ship Reservation</td>
<td></td>
</tr>
<tr>
<td>Test and Fix Bugs for Ship Reservation</td>
<td></td>
</tr>
</tbody>
</table>

Figure 10 – NextMove Team Backlog View with Tasks Coloured by their Features

Right-clicking on one of the tasks listed will bring up a context menu with the following options: adopt the task, or to view/edit information related to the task (Figure 11 below). For example, if David right-clicks on the task “Gather Requirements for Ship Reservation” and selects “Adopt Task,” the task will be removed from the Team Backlog and added to David’s Personal Backlog View. This indicates to Joe and Bob that David is committed to gather requirements for ship reservation.
There are three additional functions available in the team backlog view: Adding a task, adding a feature, and manually assigning tasks. At times, process steps of a feature may need to be revisited. For example, Joe found inconsistencies in the requirements for hotel reservation when he was implementing the feature. He can create a new instance of "Gather Requirements for Hotel reservation" task and note that the requirements need to be clarified. When a process-based task is added, NextMove will automatically generate "downstream" tasks that are dependent on the added task. Hence, Joe will not need to manually add all subsequent tasks that need to be done after the new task is completed. The add feature function will add a new feature to the project, as well as automatically generate a set of tasks according to the process used in the project.

The manual task allocation feature (see Figure 12 below) is only available for the project manager. It allows the project manager to allocate tasks to team members and to remove tasks from the team member's backlog when necessary. For example, if Joe (the project manager)
thinks Bob has taken too many tasks that he will not finish on time, he can de-allocate some or all of the tasks and put them back into the team backlog.

![Task Allocation Dialog](image)

Figure 12 – NextMove Manual Task Allocation Dialog

### 5.5.2 Personal Backlog View

The Personal Backlog View (Figure 13) displays the lists of tasks on the team member's work list. The list in this view uses the same colouring scheme for task names, task status and recommendation ratings, as the team backlog view and the list are also sorted according to the heuristics from NextMove. For each tasks on his/her personal backlog, the user can choose to view information related to the task, mark the task as completed, or push the task back into the team backlog.
For example, if David decides to take a short holiday (it is summer after all), and there are tasks left in his backlog, he can choose to push them back to the team backlog so that others can work on them instead of holding up the progress of the team (or forcing Joe to take them away). On the other hand, when Bob has finished the prototyping stage for the bus reservation feature, he can right-click on the corresponding task in his Personal Backlog View and choose “Mark Task as Completed.” The task will be removed from Bob’s backlog and the system will update the status on the tasks that are dependent on the task “Prototyping for Bus Reservation.”

### 5.5.3 Team Activities View

The Team Activities View provides a by-member listing of all the current activities of the team. Expanding the tree branches for each team member will reveal the tasks that the team member has adopted. Just as in the Team Backlog View and Personal Backlog View, the task names under each team member’s name can be coloured by their activities or by their features. The green colour in the status bar indicates that the task is ready to be executed and red shows that the dependencies of the tasks are not yet satisfied.
Agile development teams often use Parking Lot Charts [42] as a progress and task tracking tool. On a Parking Lot Chart, tasks are written on pieces of post-it notes of different colours, denoting tasks for different features. Small stickers of different colours will be attached onto the post-it note to indicate the change in status of the task. Hence, the parking lot board provides an easy, at-a-glance overview of the project progress.

The Parking Lot View (Figure 15) in NextMove is based on the concepts of the Parking Lot Chart. In its initial state, the Parking Lot View lists the features currently in the project in the order of priority. The progress bar on the right indicates the amount of work that has been done or currently in progress for the corresponding feature. The blue component shows the amount of work that is completed, and the yellow component indicates the amount of work in progress. The length of the progress bars are calculated by accumulating the estimated duration of each completed or in-progress task.
When each of the features is expanded, the user will be able to see a detailed task-by-task view of the status of each task. The status column in the middle is similar to the ones in the team backlog and personal backlog views, with the additional colours of yellow to indicate that the task is current under execution and blue to indicate that the task has been completed. The right-side column of each task shows the names of team members who are working on the task or have worked on the task.

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Status</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hotel Reservation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gather Requirements for Hotel Res.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototyping for Hotel Res.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test and Fix Bugs for Hotel Res.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement Components for Hotel Res.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flight Reservation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gather Requirements for Flight Res.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prototyping for Flight Res.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test and Fix Bugs for Flight Res.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement Components for Flight Res.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bus Reservation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Train Reservation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ship Reservation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 15 – NextMove Parking Lot View**

### 5.5.5 Artifacts View

During the course of the project, team members may find themselves unable to proceed with their tasks because another task has not been completed. Hence, a mechanism is needed to enable team members to provide feedback as to what needs to be done. Through the Artifacts View (Figure 16), the user will be able to initiate requests for modification for a particular artifact. For example, Joe needs the requirements for hotel reservation before he can start implementing that feature, but Bob, who adopted the task of “Gathering Requirements for Flight Reservation” is working on other tasks and has left the requirements half-done. In this situation,
Joe can use the Artifacts View and request the requirements document for hotel reservation to be completed. When recommended task lists are generated, the server would take these requests into consideration (please see section 4.1) and raise the priority score of the tasks that relate to the artifacts that need to be modified.

![Figure 16 – NextMove Artifacts View](image)

### 5.5.6 Log View

The Log View provides a detailed listing of all actions performed on the NextMove system. Through this filterable view, the team will be able to track different types of activities by different team members.

![Figure 17 – NextMove Log View](image)
5.6 Design Discussion

NextMove is designed to help foster communications, increase the speed and ease of information retrieval and allocate tasks. By using heuristics to determine the most pertinent tasks to the team, the NextMove tool helps relieve the project manager of the mundane task of matching tasks with performers. In addition, team members need not to wait for instructions from the project manager in order to know which task to work on next. The concept of task adoption serves as a mental model for taking responsibility of performing a task, and allows team members to use their intuition and choose their courses of actions. The Team Activities View and the Parking Lot View help team members gain awareness of others' activities. Team members will find it easier to locate the right person of whom to ask questions by looking up the performers of other tasks.

In addition, NextMove has a number of features that increase the likelihood of user adoption. In the Team Backlog View, team members are given the freedom to choose tasks to adopt instead of having the system impose a strict sequence of action. Hence, team members are able to use their intuition and not feel restrained by a seemingly draconian system. NextMove makes use of process models to automatically generate tasks that relate to a feature, which decreases the amount of data entry (and cost to adapt to the tool) for the team. Integration with the Eclipse Platform helps reduce the context switching time required when the user needs to make use of NextMove while he or she is working.
CHAPTER 6
SIMULATION

As a preliminary step to validate the heuristics used in the task recommendation process described in chapter 4, a series of simulations was conducted. This chapter describes the simulation model and process, presents a summary of the results and discusses the results' significance and the limitations of the simulation model.

6.1 Simulation Data

Three sets of project data and two sets of team member data were used for simulation. Project data includes process information (activities, the skills that each activity requires, work products, and their relationships) as well as feature information (priority, size). Team member information includes member-by-member comparison of their competencies in various categories. This subsection describes the details of each set of data.

6.1.1 Project Data

The different project data used for the simulation are intended to reflect upon different types of software projects. Each has a different number of process steps (activities) and the three projects will hereinafter be referred to as project A, project B and project C.

Project A consists of ten activities and five features, with a mix of parallel and sequential activities. The names, priorities and size of each feature are as follows:
<table>
<thead>
<tr>
<th>Name</th>
<th>Priority</th>
<th>Size (in function points)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Game Client</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Online Chat</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Scoring System</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Options Menu</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Multiplayer Server</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6. Feature List for Project A

This data set serves as an example of a relatively large-scale project with longer development cycles. Teams that work in a relatively well-known domain or on projects that are heavily contract-based may opt for this development model. The process diagram in Figure 18 illustrates the process flow of the project. The percentage below each activity name is the estimated amount of time required to perform the activity in relation to the overall estimated time to execute the process. The letters “FS” on the arrow denotes a finish-start relationship (the activity which the arrow is pointing to cannot start executing before the originating activity is completed). The letters “SS” on the arrow denotes a start-start relationship (the activity to which the arrow is pointing to can start when the originating task has started).
The process for project B consists of four sequential activities. There are ten features. The names, priorities and sizes of each feature are as follows:

Figure 18 – Process Flow Diagram of Project A
<table>
<thead>
<tr>
<th>Name</th>
<th>Priority</th>
<th>Size (in function points)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaborative Editing</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Join Session</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Create Document</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Save Document</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Instant Chat</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Callback Functions</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Login</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Add Member</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Notify Change</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Server GUI</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 7. Feature List for Project B

This data set represents projects that have short development cycles and a large number of features. Examples of such a project include performing service pack updates after feedback is gathered from the field, or responding to a series of change requests in a web site. The process flow diagram below illustrates the process flow of project B.

![Process Flow Diagram of Project B](image)

Figure 19 – Process Flow Diagram of Project B

Project C’s process is also consisted of four activities, but they are arranged in a way that two activities can be executed concurrently. The names, priorities and sizes of each feature are as follows:
<table>
<thead>
<tr>
<th>Name</th>
<th>Priority</th>
<th>Size (in function points)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hotel Reservation</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Flight Reservation</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Bus Reservation</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Train Reservation</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Ship Reservation</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 8. Feature List for Project B

Project C is an example of projects where the team has to constantly revise different artifacts at the same time because it is unfamiliar with the domain. The process flow diagram below illustrates the process flow of project C.

![Process Flow Diagram of Project C](image)

6.1.2 Team Member Data

NextMove uses the team member data to determine the task recommendation ratings on the suitability category (see section 4.2). The team member simulation data rates each simulated team member in three skill categories: Java programming skills, documentation and testing. The
ratings are expressed in a relative form so that they can be used as the inputs of an analytic hierarchy process.

Team member data set \( x \) is made up of three simulated team members: Joe, Bob and David. Their skills are as described by the scenario in section 0: Joe is modeled as a comprehensive expert who is better than the other two team members in all skill areas. David and Bob have strengths in Java programming and testing, respectively.

Team member data set \( y \) is consisted of the members from data set \( x \) and two additional simulated team members: Mary and Jane. Both Mary and Jane have balanced competencies but are less skilled than Joe.

6.2 Simulation Model
The simulator intends to imitate a real-world project environment by simulating the behaviour of team members on a project team. At the initialization of the simulation, a number of tasks, as well as team member information, are generated. Task information is generated from a process model and preloaded features information.

After the simulation starts, it runs in a round-robin, iterative fashion, and terminates when all tasks in the project are labelled as “done.” Figure 21 illustrates the general execution flow of each simulated team member, and Figure 22 shows the details of the “find and work on task” activity.
Figure 21 – General Execution Flow of Each Simulated Team Member

Figure 22 – Detailed Execution Flow within the “Find and Work on Task” Activity
Each simulated team member had up to three tasks in their personal backlog. When a simulated team member needs to take on a task, it would pick a task and adopt it from the team backlog. The time it takes for a team member to complete a particular task is calculated by combining the estimated time of the task and the competencies (as derived from the team member information) of the team member. During each time period, when a team member works on a task, the time remaining for the task is decremented. When a task’s time remaining is zero and the task’s workflow conditions (e.g., task B cannot finish before task A is finished in a finish-finish relationship) are satisfied, the team member indicates that the task is completed.

Three sets of project data and two sets of team member data were created for simulation. In combination, the project data and the team member data create six scenarios. For each scenario, simulations are executed under two conditions: randomized condition and heuristics condition. The two conditions share the same basic simulation model and only differ in the way the tasks lists are generated when a simulated team member picks a task.

Under the randomized condition, the system would rank readily executable tasks at the top and the ones that are not immediately executable at the bottom. The ranking of the tasks within the two groups are randomized. The reason for the partial randomization of the task list (as opposed to total randomization in which both the readily executable and non-executable tasks are mixed together) is to avoid deadlocking situations in which all simulated team members’ personal backlogs are full but none of the tasks in those backlogs can be executed. One thousand simulations were executed for each scenario under the randomized condition in order to create baseline results, against which the results of the heuristic condition are compared.
Under the heuristics condition, the tasks list is generated according to the heuristics-based approach as outlined in Chapter 4. A simulated team member would pick the most recommended task (at a point in time) by the system.

6.3 Data Analysis

At the end of each iteration of simulations, three measures will be taken as metrics on the efficiency of the team:

- project completion time (TPCT): the total time it takes for the project to complete. This is used as a measure of the team's overall efficiency.
- mean feature completion time (MFCT): the average time it takes for a feature to be completed in one scenario. This serves as an indicator of whether most features are completed early in the project or completed late in the project.
- weighted-mean feature completion time (WFCT): before taking the mean of the feature completion times in one scenario, feature completion times are weighted according to the priority of the feature, with the higher priority features weighted more heavily. This measure indicates whether the team completes the more important features first.

Results from simulation under the heuristics condition were compared with a corresponding set of results under the randomized condition to see how they rank amongst the randomized results.
6.4 Results

The scenarios are named according to the project data and team member data which the scenarios are derived from. For example, Scenario Ax refers to a scenario with project data set A and team member data set x.

6.4.1 Scenario Ax

The bar chart below summarizes the results of the simulation for scenario Ax. The bar with a dotted pattern shows the result from the heuristics-based condition in comparison to the randomized results. Each solid bar represents one data point amongst the one thousand iterations of randomized simulation. For example, the 80th percentile project completion time is shorter than 80% of all project completion times from randomized simulations. The shortest case refers to the shortest time among the set of randomized simulations.

![Scenario Ax - Project Completion Time](image)

Figure 23 – Comparisons of Project Completion Times in Scenario Ax
The PCT under the heuristics-based condition, highlighted in yellow, lies somewhere in between the 85th and the 90th percentile. The longest PCT among the randomized results (not shown on the chart above), is 69 simulated hours.

![Scenario Ax - Mean Feature Completion Time](image)

Figure 24 – Comparisons of Mean Feature Completion Times in Scenario Ax

The heuristics-based condition produced the shortest MFCT among the samples, indicating that the simulated team took less time to produce deliverables (i.e., completed features) under the heuristics-based condition.
### Scenario Ax - Weighted-Mean Feature Completion Time

<table>
<thead>
<tr>
<th>Feature</th>
<th>Median</th>
<th>80th Percentile</th>
<th>85th Percentile</th>
<th>90th Percentile</th>
<th>95th Percentile</th>
<th>Shortest Case</th>
<th>Heuristics Based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>32.21</td>
<td>36.53</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Figure 25 – Comparisons of Weighted-Mean Feature Completion Times in Scenario Ax

The WFCT under the heuristics-based condition is lower than the short WFCT under the randomized condition by more than 10%. Since the feature priorities are weighted in the calculation, a lower WFCT indicates that the higher-prioritized features are completed before features of lower priorities.

### 6.4.2 Scenario Ay

The setting for scenario Ay is similar to scenario Ax, with the exception of two additional simulated team members. The PCT under the heuristics condition is slightly shorter than the 95th percentile time. This may indicate that the heuristics become more effective as the number of team members increases. The MFCT and WFCT results show a similar trend as Ax.
**Scenario Ay - Project Completion Time**

<table>
<thead>
<tr>
<th>Heuristics Based</th>
<th>Shortest Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>32.5</td>
</tr>
<tr>
<td>37.5</td>
<td>36</td>
</tr>
<tr>
<td>38.5</td>
<td>38</td>
</tr>
<tr>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>42</td>
<td>44</td>
</tr>
</tbody>
</table>

**Scenario Ay - Mean Feature Completion Time**

<table>
<thead>
<tr>
<th>Heuristics Based</th>
<th>Shortest Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.4</td>
<td>28.5</td>
</tr>
<tr>
<td>30.4</td>
<td>31</td>
</tr>
<tr>
<td>31.5</td>
<td>31.8</td>
</tr>
<tr>
<td>33.6</td>
<td>33.6</td>
</tr>
</tbody>
</table>

**Figure 26 – Comparisons of Project Completion Times in Scenario Ay**

**Figure 27 – Comparisons of Mean Feature Completion Times in Scenario Ay**
6.4.3 Scenario Bx

In scenario Bx, the simulated team took slightly longer than the 80th percentile time under the heuristics condition. However, the heuristics condition produced the shortest MFCT and WFCT. The significantly lower (by 20%) WFCT under the heuristics condition shows that although the PCT may be longer under the heuristics conditions, the simulated team was able to complete the more important features and deliver them at an earlier time.
Scenario Bx - Project Completion Time

<table>
<thead>
<tr>
<th>Heuristics Based</th>
<th>Median</th>
<th>80th Percentile</th>
<th>85th Percentile</th>
<th>90th Percentile</th>
<th>95th Percentile</th>
<th>Shortest Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulated Time (hours)</td>
<td>60.5</td>
<td>57.5</td>
<td>57</td>
<td>56</td>
<td>55</td>
<td>54</td>
</tr>
</tbody>
</table>

Figure 29 – Comparisons of Project Completion Times in Scenario Bx

Scenario Bx - Mean Feature Completion Time

<table>
<thead>
<tr>
<th>Heuristics Based</th>
<th>Median</th>
<th>80th Percentile</th>
<th>85th Percentile</th>
<th>90th Percentile</th>
<th>95th Percentile</th>
<th>Shortest Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulated Time (hours)</td>
<td>43.1</td>
<td>41.2</td>
<td>40.7</td>
<td>40.15</td>
<td>39.345</td>
<td>37.1</td>
</tr>
</tbody>
</table>

Figure 30 – Comparisons of Mean Feature Completion Times in Scenario Bx

64
6.4.4 Scenario Bx

The PCT under the heuristics condition is shorter than the 95th percentile time but is 3.5 hours longer than the shortest case, and continues the trend that heuristics-based results are ranked higher with more simulated team members. The increased number of team members may have helped narrow the gap between the WFCT from the randomized results versus the WFCT under the heuristics-based condition.
**Scenario By - Project Completion Time**

- **Median**: 43.5
- **80th Percentile**: 40.5
- **85th Percentile**: 40
- **90th Percentile**: 39.5
- **95th Percentile**: 38.5
- **Heuristics Based**: 37
- **Shortest Case**: 33.5

Simulated Time (hours)

**Figure 32 – Comparisons of Project Completion Times in Scenario By**

**Scenario By - Mean Feature Completion Time**

- **Median**: 30.1
- **80th Percentile**: 28.6
- **85th Percentile**: 28.3
- **90th Percentile**: 27.95
- **95th Percentile**: 27.4
- **Heuristics Based**: 25.9
- **Shortest Case**: 25.25

Simulated Time (hours)

**Figure 33 – Comparisons of Mean Feature Completion Times in Scenario By**
Simulation results from scenario Cx produced some interesting results, as the distribution of PCT is heavily slanted towards the shortest case time, 38.5 hours. This is probably due to the high number of tasks that can be executed in parallel. The PCT under the heuristics condition is the same as the shortest case. The MFCT and WFCT results show that the heuristics are still useful, helping the team to complete important features first.
Scenario Cx - Project Completion Time

- Median: 39 hours
- Heuristics Based: 38.5 hours
- Shortest Case: 38.5 hours
- 95th Percentile: 38.5 hours
- 90th Percentile: 38.5 hours
- 85th Percentile: 38.5 hours
- 80th Percentile: 38.5 hours

Simulated Time (hours)

Figure 35 – Comparisons of Project Completion Times in Scenario Cx

Scenario Cx - Mean Feature Completion Time

- Median: 33.4 hours
- 80th Percentile: 32 hours
- 85th Percentile: 31.7 hours
- 90th Percentile: 31.2 hours
- 95th Percentile: 30.7 hours
- Shortest Case: 29.1 hours
- Heuristics Based: 26.8 hours

Simulated Time (hours)

Figure 36 – Comparisons of Mean Feature Completion Times in Scenario Cx
6.4.6 Scenario Cy

The results from scenario Cy are somewhat the opposite of the trends that were shown in scenarios with project A and B. The PCT from the heuristics-based condition in Cy does not rank as high as the PCT in Cx under the heuristics-based condition. In addition, the MCFT under the heuristics-based conditions was longer than the 90th percentile, unlike results from all other scenarios.
**Scenario Cy - Project Completion Time**

<table>
<thead>
<tr>
<th>Heuristic Type</th>
<th>Median</th>
<th>80th Percentile</th>
<th>95th Percentile</th>
<th>90th Percentile</th>
<th>85th Percentile</th>
<th>80th Percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulated Time (hours)</td>
<td>23</td>
<td>22.5</td>
<td>22.5</td>
<td>22.5</td>
<td>22.5</td>
<td>22.5</td>
</tr>
</tbody>
</table>

**Scenario Cy - Mean Feature Completion Time**

<table>
<thead>
<tr>
<th>Heuristic Type</th>
<th>Median</th>
<th>80th Percentile</th>
<th>85th Percentile</th>
<th>Heuristics Based</th>
<th>90th Percentile</th>
<th>95th Percentile</th>
<th>Shortest Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulated Time (hours)</td>
<td>21.3</td>
<td>20.4</td>
<td>20.1</td>
<td>20.1</td>
<td>19.9</td>
<td>19.4</td>
<td>18.5</td>
</tr>
</tbody>
</table>

*Figure 38 – Comparisons of Project Completion Times in Scenario Cy*

*Figure 39 – Comparisons of Mean Feature Completion Times in Scenario Cy*
Figure 40 – Comparisons of Weighted-Mean Feature Completion Times in Scenario Cy

6.5 Discussion

The overall result of the simulations is positive, which indicates that the heuristics process has the potential to be a useful tool for making coordination decision. However, there are some limitations to the simulation model which may affect the results of the simulations. This section discusses some observation taken from the results, as well as limitations to the current simulation model.

6.5.1 Observation from the Results

The heuristics-based conditions consistently post times equal or shorter than the 85th percentile of all randomized results in all categories (with the exception of the PCT in scenario Bx). The simulation results also show the flexibility of NextMove’s heuristics, as the heuristics can be
fine-tuned to adapt to different project situations. The heuristics conditions also required less switching between features for the simulated team members. Figure 41 and Figure 42 show the timeline of each team member during the course of the simulated project. Colours under the timeline represent different features, and each coloured block indicates that the simulated team member worked on the corresponding feature during that period. Figure 41 is the timeline of project A executed under the randomized condition, and the timeline in Figure 42 is the result under the heuristics-based condition of the same project. The two figures show that the simulated team members under the randomized condition often had to switch between features, whilst there were less switching between features under the heuristics-based condition.

6.5.2 Simulation Model Limitations

The simulation model represents a simplistic project model, where there is no exceptional case arising during the project's execution and there are no inherent logistical problems. A number of improvements can be added to the simulator to better reflect the realities of distributed software development.
Logistical issues such as time-zone differences can be built into the simulation model to better reflect the realities of distributed development, by placing restrictions on the work hours of each virtual team members.

During the course of a software development project, emergent events, such as changes in the feature list or changes in personnel, are fairly common. These events should be modeled in the simulation in order to test the NextMove framework’s exception-handling capabilities.

The simulation assumes that the team members have perfect communications (e.g. Bob is informed instantaneously when Joe has completed a task). This is an unrealistic assumption because communication overhead will always incur on a team with two members or more. Time elapsed due to communication overhead should be added to the refined model.

It is probably also unrealistic to assume that team members would always take the most highly recommended task from the team backlog. Hence, virtual team members should select tasks based on their preferences. The simulation’s accuracy can be further improved by including the effect of preferences on productivity (e.g., a team member may be less efficient when he or she is working on an undesirable task).
CHAPTER 7

EXPERIMENT

In addition to simulations, experiments with human subjects were conducted to further study the effectiveness of NextMove as a project coordination tool. While simulations have the advantage of being able to run repeatedly with relatively low cost in time and effort, they cannot evaluate the effects of user interaction on NextMove. As subjects have different perceptions of the tool as they start using it, their mode of usage may not be anticipated by the simulator nor the author of the thesis.

7.1 Experimental Planning

7.1.1 Goals

The following measures were recorded during the experiment and analyzed afterwards: total project completion time (TPCT), mean feature completion time (MFCT), weighted-mean feature completion time (WFCT). The three measures listed above correspond to the metrics which were used in the simulation. Please see the Analysis Procedure section for details.

7.1.2 Experimental Units

The population of subjects was drawn from the graduate student population, and the subjects had some experience in working in software projects. In each experiment session, three subjects formed a group that represented a distributed project team. The grouping of the subjects in each session was subjected to the schedules and availability of the subjects. All three treatments
(please see below) were assigned to each group. Subjects who had completed a session were asked not to inform other subjects on the results of their session. There were three sessions in total.

7.1.3 Experimental Material

The objects of the experiments were three projects that make use of different development processes. They were the same three projects which were used in the simulation. However, in order for project A and B to be finished within a reasonable time for the subjects, the number of features in project A was reduced to two and the number of features of project B was reduced to five.

Three types of treatments were assigned to the subjects. The treatments were labeled as: baseline, partial and full. Under the baseline treatment, the group had access only to instant messaging software as the communication tool while coordinating their work. Under the partial treatment, the group had access to NextMove as an aid, but the heuristics process on task recommendation was disabled. Instead, the recommended task lists were randomized lists of tasks. The group also had access to the same communication tool as in the baseline treatment. Under the full treatment, the group had access to the full-featured NextMove tool, as well as the communication tool in the baseline treatment.

The order of assignment of treatments and objects for each session was as follows:
Table 9. Order of Assignment of Treatments

<table>
<thead>
<tr>
<th>Order</th>
<th>Treatment</th>
<th>Project</th>
<th>Treatment</th>
<th>Project</th>
<th>Treatment</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>baseline</td>
<td>A</td>
<td>partial</td>
<td>A</td>
<td>full</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>partial</td>
<td>B</td>
<td>full</td>
<td>B</td>
<td>baseline</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>full</td>
<td>C</td>
<td>baseline</td>
<td>C</td>
<td>partial</td>
<td>C</td>
</tr>
</tbody>
</table>

The rotation of the treatments in different sessions was to mitigate the learning effects which may occur over the course of a session (e.g. the team members were more efficient in the third project than in the first project not because of difference in treatments, but because they got used to working with the experiment interface).

7.1.4 Tasks

In each session, the subjects worked in groups of three as a project team. Each subject took on the role of a team member, namely Bob, Joe and David. The subjects are informed of the pre-assigned attributes (in Java programming, documentation and testing) for each role.

The subjects sat in different parts of a computer laboratory in order to simulate a distributed environment. The participants worked on the three projects under different treatments, as specified by the above table.

During the experiment, a participant emulated their “work” on a task by selecting it in the Work View (see Figure 43 below) and clicking on the “work” button. If a task was dependent on another task, the participant was required to enter the keyword that was supplied from the other
task. The keywords needed for executing certain tasks were randomly generated before the start of each scenario.

![Figure 43 - The Work Simulator Interface](image)

After the work button was clicked by the subject, a timer would start and count down until time ran out or the participant interrupted. The timer length was dependent on the size of the feature, as well as the attributes assigned to the team member (i.e. since David was more competent than Bob in programming, the programming task timer length for David would be shorter than Bob’s).

The participant could interrupt the timer by clicking on either the “stop” or “cancel” button. When the “stop” button was pressed, the timer would stop and the progress on the task would be saved on the system. When the “cancel” button was pressed, the work on the task would not be recorded on the system. The “cancel” button was implemented in case the subject had selected a task which he or she did not mean to choose.

### 7.1.5 Hypothesis

Prior to the start of the experiment, the following hypotheses were made:
1. teams are most effective when they make use of the fully-featured NextMove tool
2. teams are more effective with the help of NextMove tool with the heuristics process turned off than when they have no assistance
3. teams benefit the most from NextMove in more complex (i.e. higher degree of interdependency) projects

7.1.6 Procedure

At the beginning of each session, the subjects were given a synopsis of the experimental procedures and a tutorial on the experiment tool. Questions regarding the tool and the experimental procedures were answered. In addition, 10-minute tutorials on the functionalities of NextMove were given to the subjects before they started using the tool. As aids, all subjects were given the following guides for each project: the process flowchart, the feature list, and the relative competencies of each team member.

At the beginning of the experiment, all project data was reset to its initial state. Then the subjects started working on the first project with the first type of treatment and performing the tasks as outlined in the previous section.

7.1.7 Analysis Procedure

The analysis procedure after the experiments was similar to the corresponding procedure for the simulation. From each project in the experiment, three measures were taken as metrics on the efficiency of the team:

- total project completion time (TPCT): the time between the first task being started and the completion of the last task
• mean feature completion time (MFCT): the average time it takes for a feature to be completed

• weighted-mean feature completion time (WFCT): similar to the MFCT, but with the average weighted according to the priorities of the features

7.2 Results

The results of the experiment are presented on a project-by-project basis. For each project, the TPCT, MFCT and WFCT of the three treatments are compared through horizontal bar charts. Each column of a chart is labeled as “treatment name, sy,” where y indicates the session (1, 2, or 3) that the data point is taken from.

7.2.1 Project A

The results from project A show that the project that ran under the full treatment (with the fully-featured NextMove tool) took the shortest time to complete. It also posted the shortest MFCT and WFCT. However, when the heuristics processes were disabled (in the partial treatment), the project took longer to complete than when the subject did not make use of NextMove at all (in the baseline treatment). The MFCT and WFCT for the partial treatment also significantly lagged behind (by about 10 minutes) the other two treatments, indicating that most features were not completed until the later stage of the project.
Project A - Total Project Completion Time

- partial, s2: 28.76 minutes
- baseline, s1: 22.78 minutes
- full, s3: 13.76 minutes

Figure 44 – Comparison of Total Project Completion Time in Project A

Project A - Mean Feature Completion Time

- partial, s2: 26.36 minutes
- baseline, s1: 17.46 minutes
- full, s3: 12.84 minutes

Figure 45 – Comparison of Mean Feature Completion Time in Project A
7.2.2 Project B

The results from project B are a complete reversal from the corresponding results from project A, as the TPCT, MFCT and WFCT from the partial treatment are the lowest among the three treatments. The project instance under the full treatment returned the worst results (all of TPCT, MFCT and WFCT from the full treatment are about twice as long as the corresponding times from the partial treatment). It is also worth noting that the group from session 2 posted the longest TPCT, MFCT and WFCT in both project A and B.
Project B - Total Project Completion Time

<table>
<thead>
<tr>
<th>Treatment</th>
<th>Actual Time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>full, s2</td>
<td>14.97</td>
</tr>
<tr>
<td>baseline, s3</td>
<td>10.12</td>
</tr>
<tr>
<td>partial, s1</td>
<td>6.91</td>
</tr>
</tbody>
</table>

Figure 47 – Comparison of Total Project Completion Time in Project B

Project B - Mean Feature Completion Time

<table>
<thead>
<tr>
<th>Treatment</th>
<th>Actual Time (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>full, s2</td>
<td>10.32</td>
</tr>
<tr>
<td>baseline, s3</td>
<td>8.61</td>
</tr>
<tr>
<td>partial, s1</td>
<td>4.99</td>
</tr>
</tbody>
</table>

Figure 48 – Comparison of Mean Feature Completion Time in Project B
7.2.3 Project C

The group that did not have access to NextMove (the baseline treatment) managed to return the best results in project C. The group under the partial treatment posted slightly slower MFCT and WFCT, while the full treatment has the worst time in all three categories.
Figure 50 – Comparison of Total Project Completion Time in Project C

Figure 51 – Comparison of Mean Feature Completion Time in Project C
7.3 Discussion
The results from the experiment are inconclusive at best, with all three treatments posting the best results in three different projects. The results fail to show that NextMove, with or without heuristics enabled, is capable of improving the efficiency of a distributed project team. This section summarizes the experimental results and offers some possible explanations.

7.3.1 Summary of Results
Since the orders of results in all categories for each of the three projects are the same, the results of the experiments can be summarized using the following table:

<table>
<thead>
<tr>
<th>Treatment</th>
<th>Project A</th>
<th>Project B</th>
<th>Project C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest Time</td>
<td>full</td>
<td>partial</td>
<td>baseline</td>
</tr>
<tr>
<td>Medium Time</td>
<td>baseline</td>
<td>baseline</td>
<td>partial</td>
</tr>
<tr>
<td>Longest Time</td>
<td>partial</td>
<td>full</td>
<td>full</td>
</tr>
</tbody>
</table>

Table 10. Summary of Experimental Results
Only the results in project A fulfilled hypothesis (1) (with the full treatment posting the best time) and only the results in project B fulfilled hypothesis (2) (as the partial treatment posted better time than the baseline treatment, but the full treatment posted the longest times). The full treatment posted the shortest time in project A, which has relatively complex interdependencies between tasks compared to project B and project C. Therefore, hypothesis (3) holds true in the context of the three projects involved.

7.3.2 Ineffectiveness of NextMove

While simulation results show that NextMove’s heuristics does improve a distributed team’s efficiency, differences between the simulation and experimental conditions may have caused the discrepancies in results.

As the task recommendation list is set up to rank readily executed tasks on the top of the list, it is possible that subjects who took on the role of the most competent team member would adopt tasks from a lower-priority feature, if the tasks from the higher-priority features were all adopted. When tasks from the higher-priority features become available, they may have been adopted by roles that were less competent (hence the tasks would take longer to complete).

In instances where there are more features than team members (such as project B and C), it may be possible that the subjects choose to divide up the features and work on them independently, hence eliminating the communication overhead. An example of this situation can be seen in the following Figure 53, which is the Gantt chart of the execution of project C under the baseline treatment. Each horizontal bar depicts a task, and the feature to which the task belongs is labeled on the y-axis. Each pattern represents a team member, with solid pattern associated with Bob,
vertical stripes associated with Joe, and brick pattern associated with David. The chart shows that each team member focused on one or two feature. Bob concentrated on Bus Reservation; Joe focused on Flight Reservation and Ship Reservation and David concentrated on Hotel Reservation and Train Reservation.

Figure 53 – Gantt Chart for Project C (Baseline Treatment)

Such situations may not be realistic in a real-world software project because of the interdependencies between features (which the experiment did not investigate). The example from above may also help explain the reason for the full treatment getting the best results in project A where there were only two features but the tasks are very much interrelated.

One of the advantages of NextMove's heuristics is that it is versatile in the way that it can be adjusted to accommodate different goals (e.g. minimize project completion time, or optimize the
project for a quicker first release) and situations. However, when the experiments were executed, the same heuristics were used for all three projects, in order to decrease the number of variables in the experiment. Project B and C under the full treatment may return better results if project-specific calibrated heuristics are used.

7.3.3 People Factor

As with any real-life software projects, group dynamics and individual behaviours play a big part in determining the outcome of the experiment. Understanding the effects of the people factor will help the design and execution of possible further studies.

Although the subjects were given a tutorial of NextMove, which introduces various components of the user interface and what they can do with them, the subjects were not explicitly told what to do with the tool. Hence, some subject may have used the tool in manners which were unexpected. For example, in the project depicted in the Gantt chart of Figure 54, subjects who played the role of David and Bob (brick pattern and solid pattern) took on a large number of tasks early on. As a result, Joe (who was the most competent of the three team members) was deprived of tasks and did not do much work. Therefore, time was wasted during the early stages of the project.
During the experiment, the manual override options (please see chapter 5) were not made available to any of the subject. Such mechanisms could have helped the team resolve the aforementioned priority inversion and task-hogging situations. One of the most important lessons learned from the experiments is that while a tool can be fully featured and designed to be as user-friendly as possible, it is still up to the team members to collaborate as a team in order for a project to be successful.

7.4 Validity

When evaluating the validity of the aforementioned experiments, there are some issues to be considered:
• sample size: due to time and resource constrains, only nine subjects have participated in the study. The small sample size means that any outliers in the data will have a significant impact on the overall results. For example, if the resulting times from the partial treatment and full treatment in session 2 are halved, the overall results and the conclusion drawn from the results will be completely different

• learning effect: the efficiency of the subjects may be affected by the increasing familiarity with the experimental tool. The design of the studies attempts to mitigate this effect by rotating the order of treatments. However, the order of the projects were not rotated; hence it is not known whether the results will be different if the subjects execute the projects in different orders

• skill differences: apart from the efficiency of coordination, metrics such as total project completion times are affected by the technical skill levels (e.g., proficiency in Java programming) of the team members involved. Hence, the experimental tool was designed such that the tasks are not dependent on the subject's technical skills. However, peripheral skills such as familiarity with instant messaging application (which was used in the experiment) still affect an individual's and the team's efficiency. It is difficult to account for such subtlety with only empirical data

• generalizability: whenever an experiment is designed to emulate a real-life environment, there is always the question of whether the results will be readily applicable to the real world. Although the design of the aforementioned experiment attempts to model a distributed environment as closely as possible, there are some shortcomings. Users develop usage patterns over long periods of time and it can be argued that the effectiveness (or the lack thereof) of NextMove cannot be seen until the users have
become fully accustomed to using it. In addition, as the tasks in the experiment are simply button pressing and waiting for the timer to expire, the effects of task preferences (as mentioned in section 2.1.5) were not present.
CHAPTER 8
CONCLUSIONS AND FUTURE WORK

The work of this thesis focused on the problem of task coordination in a distributed agile project. Its main body of work consists of a heuristics-based process to help answer the questions of "what to do next" and "who should do it" in a distributed software development project. Based on the heuristics-based process, NextMove was built as a project coordination tool. The validation process includes both automated simulations and experimentation with human subjects. The results provide some evidence that the heuristics-based process is effective in a bounded environment (i.e., the simulation), but NextMove's effectiveness as a project management tool in a real-world, less controlled environment remains to be further investigated. This chapter recaps the contributions made by this thesis to the field of software project management and outlines possible future work and studies.

8.1 Research Goals Summary

The goal of this thesis is to improve the efficiency of a distributed agile team. In a distributed environment, the project manager often becomes solely responsible for collecting all relevant information (e.g., current progress of each team member, issues that team members encounter during their work) and allocating tasks to team members. This is a particularly prevalent problem in a distributed environment because there is a significant amount of informal communication between the team members. The work of this thesis aims to relieve the project manager's burden by providing recommendations to individual team members on which task to work on next. The
recommendations are based on a heuristics-based process. The success of NextMove was measured by the following metrics:

- project completion time (PCT or TPCT): the time between the first task being started and the completion of the last task
- mean feature completion time (MFCT): the average time it takes for a feature to be completed
- weighted-mean feature completion time (WFCT): similar to the MFCT, but the average is weighted according to the priorities of the features

### 8.2 Contributions of This Work

The work of this thesis made the following contributions:

- reviewed the current state of project management tools and identified task allocation as a weak area of support
- developed a heuristics-based process for finding important tasks and allocating them to the best candidate. The process is based on a number of concepts that are commonly used by project managers, such as critical path analysis and feature-driven development. Hence, the reasoning behind the analysis can be easily understood without going through a steep learning curve
- implemented NextMove, a project management tool which is based on the aforementioned heuristics-based process. In addition, NextMove's user interface is designed to help team members gain awareness of their activities and facilitate communications between them
- validated the effectiveness of the heuristic process in a bounded and controlled simulation environment. The results indicate that the heuristics process outperforms the
80th percentile out of a set of partially randomized execution sequences in all but one of the eighteen metrics collected

- conducted a series of experiments to test the effectiveness of NextMove as a tool with human subjects. However, the results are not consistent with the simulation results. No treatments were able to show a clear advantage in all of the project scenarios. A number of possible reasons for these results are discussed in chapter 7

- parts of the work of this thesis have been published academically. [1] is a synopsis of the heuristics process; while [2] focuses on the user interface of NextMove, and [3] presents overviews of both the heuristic process and NextMove, as well as preliminary simulation results

8.3 Future Work

During the course of the work on this thesis, three areas of future work and studies have been identified. They are: i) improvements of the simulation model to reflect more complicated situations, ii) refinements of the experimental methods in order to produce more valid results and iii) long term improvements to NextMove in order to improve its effectiveness.

8.3.1 Simulation Model Improvements

Section 6.5.2 outlined the limitations of the current simulation model. In light of these limitations, this section proposes a number of improvements to the simulation model.

A limit on the periods of time that the simulated team member can work can be defined, and periods of work time for each team members can be shifted to reflect real-world time-zone differences. For example, if the simulated project timeline is divided into 24-hour cycles, and
each simulated team member works in an eight-hour period in each cycle, the work period of each team member can be assigned to different parts of the cycle, as illustrated by Figure 55:

![Figure 55 - Example of Time Zone Assignments](image)

The simulated team members will only be taking on new tasks and work on tasks in their own backlog in the shaded part of the timeline. Hence, each team member is working in three different time zones. The work periods can also be arranged in different ways to reflect the different locations of team members and thus the effects of different degrees of time differences can be investigated.

The simulator can be modified to allow events to be inserted at a specific time. For example, a new, high-priority feature may be added after the project has been executed for fifteen hours. The resulting response from the simulated team with and without the heuristics-based process can be compared in order to evaluate how well the heuristics-based process adapts to emergent situations.

In addition to the possible communication delays introduced by time-zone differences, the number of team members on a team also affects the communication efficiency of a team, especially with late additions of team members (i.e., the "tar-pit effect" [43]). The number of required communication channels in a team increases by n-1 (where n is the team size) with each
additional member. Therefore, the communication delay in the simulation model can be modeled as a polynomial function in relation with the team size.

An “efficiency factor” can be built into the simulation model in order to reflect the effects of team member preferences and inaccuracies in the team member skill evaluation process. The results of different efficiency factors can be compared to determine how accurate the input data has to be in order for the data to be considered useful.

8.3.2 Experimental Methods Refinements

- In order to increase the confidence level of the data, a larger pool of subjects should be selected. If the size of the subject pool permits, some subjects can be placed into groups with other subjects with whom they are acquainted, and others placed into groups with subjects with whom they do not have social relationships. This will help investigate how social relationships play a role in affecting the usage pattern of NextMove (e.g., whether they will depend more or less on the tool if the subjects know each other).

- One of the subjects can be assigned to be the project manager of the team and make use of the manual task allocation features (please see Chapter 5) in NextMove. Such mechanism could have helped the team in resolving the aforementioned priority inversion and task-hogging situations.

- The subjects should be given more time to develop familiarity with NextMove, so that less mistakes due to unfamiliarity with the tool (e.g., clicking the unintended button) will be made which may skew the results.
• Real projects that span over longer periods should be used instead of a "work simulator." In this way, investigators will be able to observe how usage patterns develop over time and whether opt-out behaviours will occur, in addition to the quantitative data gathered.

8.3.3 Long Term Development of NextMove

In addition to improvements to the simulation model and experimental methods, there are a number of other possible extensions to NextMove. These extensions would further augment the framework's adaptability to different situations and increase the quality of its coordination recommendations.

One of the major emphases of the design of NextMove is to minimize the amount of work that the users need to do in order to gain value from the tool. One way of achieving this goal is to interface NextMove with other project management tools and platforms. Tools such as version control systems (e.g. CVS, Subversion), bug and issue tracking systems (e.g., Bugzilla) and collaboration platforms such as Jazz contain valuable information which NextMove can make use of in order to generate project and team member information.

The task-allocation process can be further augmented to include personalities as factors in order to find the best fit between task requirements and team member capabilities. A paper by Acuña, Juristo and Moreno [44] describes how team member personality information is used to match team members to roles.

The current task model assumes that only one team member works on a particular task at one time. However, some agile practices may require some tasks to be performed by more than one
team member at the same time (e.g., pair programming in eXtreme Programming). Therefore, NextMove should be extended to support collaborative tasks.

Without accurate information on the characteristics of tasks and team members, NextMove's heuristics-based process would not be able to produce meaningful or useful results. Therefore, a project data collection methodology should be developed and used in conjunction with NextMove. One of the possible avenues in this area of development is the usage of machine learning techniques to adjust the ratings of each team member over time.

As described in Chapter 5, the current implementation of NextMove is based on a client-server model. While this approach is straightforward, it may lead to a system that suffers from reliability and performance issues as the number of users increases. Future work may explore alternative architectures, such as peer-to-peer based design, as proposed by Jun and Yun [45].

NextMove is not meant to be a catch-all solution to the task coordination and communication problems in a distributed software development environment. Rather, it aims to augment a team's ability to dynamically coordinate tasks in a distributed environment. Therefore, it is best to find ways to integrate NextMove with other project management/tracking tools and gather useful information from them, rather than reinvent the wheel by attempting to expand to other areas such as messaging and artifact repositories.

8.4 Conclusion
This thesis identifies task coordination in a distributed agile team as an important problem, and describes a heuristics-based process that helps teams in making decisions on task allocation. The
heuristics-based process is implemented as NextMove, a tool which aims to increase efficiency in task coordination, as well as project transparency. The heuristics process was tested under simulation and the results show that the process is potentially useful for improving the efficiency of the team. In addition, this thesis describes the experiments that were conducted to study the interaction between human subjects and NextMove. However, the results are not conclusive and further studies will be required to verify NextMove’s usefulness in a real-world project environment.

The work of this thesis serves as a first step towards a solution for increasing the quality of task coordination decisions in a distributed agile environment. The heuristics-based process balances the need for optimizing coordination from a global perspective and the need for team members to know and have control over their next moves in the project. Interactions between individuals and effective responses to changes are two of the main emphasis the Agile Manifesto [11], as well as two challenges faced by a distributed team. The work of this thesis helps teams answer these challenges.
REFERENCES


