TASK COORDINATION IN AN AGILE DISTRIBUTED SOFTWARE DEVELOPMENT ENVIRONMENT

David K.M. Mak
University of British Columbia
email: kmak@ece.ubc.ca

Philippe B. Kruchten
University of British Columbia
email: pbk@ece.ubc.ca

Abstract
As both distributed software development (DSD) and agile development practices become more popular, the problem of task coordination in an agile DSD environment becomes more pertinent. Even though task allocation has been a subject of study for many years, the team dynamics in an agile DSD environment makes the nature of task coordination distinctly different from that in other disciplines.

This paper proposes a solution to the problem of remote task allocation and coordination in an agile DSD environment. It combines current practices in software project management, such as object-oriented process modeling and critical-path analysis, and methodologies from other fields, such as workflow management and management science. It also describes NextMove, a Java/Eclipse-based distributed tool that would assist project managers in making day-to-day task allocation decisions, increasing transparency throughout the project, as well as complementing other modes of communication in a DSD environment.

Keywords: Project Management, Task Allocation, Agile Development, Distributed Software Development

1. Introduction

Task coordination and allocation in an agile DSD environment is a difficult problem for any project team. Agile development practices require frequent, informal communications, such as daily stand-up meetings, in order to manage project backlogs and achieve the effect of a scrum. Such informal communications are difficult to conduct when team members are geographically dispersed. In addition, the team dynamics in an agile DSD team makes the nature of task coordination distinctly different from that in other disciplines such as manufacturing.

This paper discusses issues affecting task coordination and allocation in an agile DSD environment and proposes a solution to the problem. The characteristics of the problem described are outlined first and prior work reviewed. Then the approach of the proposed solution is introduced and NextMove, an implementation of the solution is introduced. Finally, possible extensions to the project are discussed.

2. Background

Although the problem of effective coordination exists within projects in other engineering fields, a number of critical differences between software projects and other traditional engineering projects make the issue of coordination in agile distributed software projects unique.

In the last few years, agile software development methods [1] have sprouted and gained popularity. Examples include eXtreme Programming [2], Lean [3], Scrum [4], Feature-Driven Development [5], and Crystal [6]. The common goals of these methods are to lessen the reliance on heavy upfront planning and written artifacts and to increase coordination between parties by direct communication. They emphasize teamwork over atomization of work and pipelining of activities according to a rigid, predefined process.

However, when teams have more than 12 to 15 members, or when members are geographically dispersed, problems with managing the dynamic and tactical activities of discovering, scheduling and allocation of tasks emerge. Agile techniques such as features or story cards tend not to adapt very well. For example, agile teams often use daily stand-up meetings (also known as scrum meetings) to report on current activities, identify issues, and tactically plan the next activities. As an aid, they often use wall displays populated with cards or Post-it® notes. However, such interactions cannot be emulated in a distributed environment, since audio/video technologies do not adequately replace face-to-face interaction [7].

Agile software development teams tend to be less hierarchical than other types of project teams due to a philosophy that favours generalists over specialists. Hence, each team member is able to take on different roles. In such a structure, the team would have to rely on the project manager to make task coordination decisions and allocate roles (and in turn tasks) to various team members.

The combination of the above factors poses a challenge for the project manager: s/he is forced to retrieve all relevant information due to the lack of means for informal communication. The project manager may also represent an information bottleneck because all team members would have to look to the project manager for information.

3. Prior Work

A number of solutions to the task-coordination and allocation problem have been proposed. This section provides a summary of these prior works.
3.1. Task-Assignment Policy Modeling using UML

Shim, Lee, et al. proposed a model-based methodology that allows the modeling of task-assignment policies [8]. Their methodology makes use of UML-based meta-models to describe the four crucial elements of task assignment: organization, process, work products, and project. The models contain information on roles, work products, and their relationships, as well as the workload information for each team member. The models combine to define the task assignment policies and constraints within a project.

Since the proposed methodology consisted of only a simple set of basic elements, it is adaptable to different development environments. However, the models do not take into account the relationships between work products; instead, they treat each task as a standalone instance.

3.2. Multi-Criteria Task Allocation

Shen, Tzeng, et al. proposed a multi-criteria task assignment model within workflow management systems [9]. The model describes qualitative measures, such as individual capabilities, using linguistic scales that are quantified by fuzzy numbers. The suitability of a team member for a task is evaluated using four criteria: capability, workload, similarities of tasks, and relationship between team members. For each task, a suitability score for each team member is derived by aggregating pairwise comparison of results between all eligible team members. Social-relationship scores are also derived in cases where the task must be performed by a team.

This task assignment model is flexible and easy to understand. However, as with the meta-model-based approach, this model assumes that tasks are independent of one another.

3.3. Task-Allocation Optimizer

Duggan, Byrne, and Lyons proposed an optimization method for task allocation during the construction phase of software development [10]. 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. These goals 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 individual 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. Task-coordination and allocation problem are treated as static optimization problems, and emergent changes are not taken into account. Moreover, 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.

4. Problem Definition

The issues and prior work described in Section 3 demonstrate the need for a method to facilitate task coordination and allocation decisions. A model, which takes into account all aforementioned factors, must be defined in order to describe the context the method applies to.

4.1. Process and Project

The method proposed by this paper applies to a distributed software development organization that follows a software development process, such as those defined by the Software Engineering Process Meta-model (SPEM) [11]. A number of inter-related activities exist in a process model. Each activity represents a step towards completion of the project. An activity may have one or more work products as inputs or outputs. A number of roles are defined within the process model. Each role is responsible for executing a set of activities, and maintains a set of work products.

A project is the realization of a process. A project consists of team members, tasks and artifacts corresponding to roles, activities, and work products in a process. For example, one or more team member(s) may take on a role and work on a task, which is spawned from an activity. A task may modify one or more artifacts, which are physical instances of work products.

This approach to modeling enables a distinct separation between the abstract process model and the physical project. According to van der Hoek et al., formal processes work best when they are used to provide high-level guidance and support [12] instead of imposing 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 below illustrates the relationship between process, activities, tasks, and artifacts.

![Figure 1. Relationships between abstract process elements and physical project elements](image-url)
4.2. Task Model

In the proposed task model, there are two types of tasks: tasks spawned from activities in a process model and ad-hoc tasks that are initiated by human stakeholders in reaction to events emerging during the course of the project. Ad-hoc tasks do not necessarily modify any artifact; nonetheless they are critical to the project’s success. An example of such a task is moving certain equipment from one part of an office to another.

At any instance during the execution of a project, there are a number of to-be-done tasks in the team backlog. Each task has a number of attributes:
- Category: e.g., creation, modification, review, administrative, verification
- Workflow relationship: which task(s) has to be done before the task can be executed; which task(s) depends on the task
- Effort/schedule estimation: the amount of time needed to perform the task
- Deadline: whether the task has to be done before a certain date
- Task initiator: the person or process who created the task, and
- Task performer: the person who is currently performing the task, if any
A task created according to a process has the following additional attributes:
- Parent Activity: which activity the task would (partially) satisfy
- Artifact relationship: input/output artifacts of the task

4.3. Team Member Model

A project team consists of two or more members. At various points of the project, a team member may take on different roles and perform different tasks. Each team member has different skill-sets and attributes including the following:
- Skills: skills and experience that enable an individual to perform the aforementioned types of tasks, such as programming skills, documentation skills
- Familiarity with artifact: as the project progresses, certain team members may become more familiar with certain set of artifacts
- Workload: the amount of work the team member has taken on at that time

4.4. Artifacts Model

Artifacts may be physical documents, such as design and architecture documentation, as well as software code. An artifact goes through different states as it is modified by different tasks. An artifact has the following attributes:
- Parent work product: the class of work product that the artifact belongs to
- Type: e.g. documentation, code
- State: current status of the artifact, e.g., needs to be modified, is finished
- Artifact dependencies: other artifacts that may impact the artifact when they are changed

4.5. Task Coordination and Allocation Problem

Given the aforementioned models, the project manager faces a two-part problem: prioritize the given set of tasks, and determine the best personnel to perform these. In essence, the project manager needs to arrange a set of tasks, \( T \), consisting of \( n \) tasks, into an n-tuple according to various factors listed above. Then, for each task, \( t_i \), each with a set of requirements, \( R_n \), the project manager needs to find one or more team members \( M \), each with a set of attributes \( A_{mn} \), who is/are most suitable to perform \( t_i \).

In order for a method to succeed in solving the task-coordination and allocation problem in the context outlined in the previous sections, it must 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

5. Approach

The authors of this paper propose a two-step approach to solving the task-coordination and allocation problem. The first step is to prioritize currently available tasks. Next, a suitability ranking of team members is produced for each available task. The two rankings are then combined to produce a recommended task list for each team member.

5.1. Task Prioritization

Tasks are prioritized according to three different criteria: task dependencies, schedules, and artifact relationship. For each task, a priority score in each criterion is assigned. The three scores are weighted and summed to produce a final priority score. In exceptional circumstances, the project manager may elevate the priority of tasks.

Artifact relationships are concerned with which artifacts the task would create, modify or review. Given a task, \( t \), that would perform operations on artifact \( a \) and change it 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 which require \( a \) to be at state \( s \) in order to start execution. In addition, in cases where the artifact-task relationship is not defined in the process model, team members can request a certain artifact \( a \) be changed into state \( s \). Hence, the priority
score (PS) of a task, in terms of artifact relationships, is derived by:

$$PS_{artifact} = \beta_{tasks} R_{tasks} + \beta_{team} R_{team}$$

where $R_{tasks}$ is the number of tasks that require the artifacts produced by $t$, and $R_{team}$ is the number of requests to work on the artifact from the team. $\beta_{tasks}$ and $\beta_{team}$ are weight factors.

The task-dependencies criterion deals with the number of tasks that are sequentially related to the task in question. Although task dependencies may be implied in terms of artifact relationships, some tasks may depend on other tasks which do not modify any artifact. The priority score of a task, in terms of task dependencies, is derived by:

$$PS_{td} = n_{tasks}$$

where $n_{tasks}$ is the number of tasks depending on $t$.

In an engineering project, a critical-path analysis (CPA) is often performed to estimate the minimum amount of time needed to complete the project and to identify the tasks that are critical to its timely completion. The proposed approach performs a CPA on the available tasks in the project and assigns a priority score for each task, determined by:

$$PS_{schedule} = \frac{1}{1 + \frac{LST - EST}{T_{PD}}}$$

where $LST$ and $EST$ denote the latest and earliest start time, respectively, and $T_{PD}$ is the total project duration. The 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 would be most highly rated.

The three priority scores are aggregated as a weighted sum to derive the final score:

$$PS_{final} = \lambda_{artifact} PS_{artifact} + \lambda_{td} PS_{td} + \lambda_{schedule} PS_{schedule}$$

where the $\lambda$ values are the weights of each prioritization criterion.

### 5.2. Task Allocation

The task-allocation process answers the question of “who should do what?” It has a number of objectives:

- Match task requirements to individual skills: locate the person with the best qualification to perform each task
- Balance workload: workload has to be taken into consideration because a purely skills-based assignment of tasks may result in most tasks being allocated to the few “gurus” who excel in most technical fields, causing an imbalance in workload among team members
- Reduce overhead: overhead due to context switching may occur when a team member has to move from working on one sub-system to another. The severity of this problem depends on the scope and complexity of the project

Since the aforementioned goals may conflict with one another, a way of resolving the problem in an objective manner and producing the optimal result is needed. The analytic hierarchy process (AHP) may be used for the purposes of quantifying qualitative properties and solving the multi-criteria decision problem.
Table 3. Team members’ relative Java programming abilities

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.00</td>
<td>0.33</td>
<td>0.20</td>
</tr>
<tr>
<td>B</td>
<td>3.00</td>
<td>1.00</td>
<td>0.33</td>
</tr>
<tr>
<td>C</td>
<td>5.00</td>
<td>3.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 4. Normalized relative competencies for each team member

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>0.11</td>
<td>0.26</td>
<td>0.63</td>
</tr>
<tr>
<td>Design</td>
<td>0.11</td>
<td>0.11</td>
<td>0.78</td>
</tr>
<tr>
<td>Documentation</td>
<td>0.75</td>
<td>0.18</td>
<td>0.07</td>
</tr>
</tbody>
</table>

Table 5. Ranking the skills-based suitability ratings of the team members

<table>
<thead>
<tr>
<th></th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0.611</td>
</tr>
<tr>
<td>A</td>
<td>0.423</td>
</tr>
<tr>
<td>B</td>
<td>0.262</td>
</tr>
</tbody>
</table>

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 sub-system 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.

5.3. Recommended Task List

In order to produce the final recommendation rating, the overall priority score of the task will be multiplied by the suitability score of the team member. The final output of the proposed approach is a personalized backlog for each team member, with tasks ranked from the “most recommended” to the “least recommended.”

5.4. Benefits of Approach

The proposed approach is suitable for a distributed agile development environment for a number of reasons. First, compared to optimization algorithms such as genetic algorithms, the decision process using AHP is more transparent. Team members are able to understand the rationale behind the prioritization and allocation by examining the various factors that influenced the decision.

Second, by producing a personalized backlog for each team member instead of performing one-to-one mapping of tasks, team members may select a suitable task.

Third, weights and constants used in the rating calculations can be adjusted according to the needs of the organization and the current situation. For example, a team facing a hard deadline may weight the schedule factor more heavily. On the other hand, a team that produces mission-critical systems may adjust the weights to ensure that the best personnel perform the most important tasks.

In addition, by allowing direct input from team members into criteria such as artifact dependencies, the proposed approach enables team members to provide feedback and be involved in the task coordination and allocation process.

6. NextMove

As a proof-of-concept of the proposed method of solving the task allocation and coordination problem, a distributed project management application called NextMove is currently under development.

6.1. Design Goals

NextMove does not intend to completely replace all aspects of a daily stand-up meeting. As a groupware, the focus of NextMove is to support distributed, asynchronous task allocation from the perspectives of both project managers and team members. The authors of this paper have set the following design goals, based on literature research in the field of computer-supported collaborative work:

- NextMove shall not impose a strict sequence of actions on the user; instead it shall allow him/her to choose from a list of recommended actions
- NextMove shall facilitate communication between team members by allowing them to convey explanations or context about a task or an artifact [14]
- NextMove shall minimize the amount of user intervention in the system in order to minimize the amount of extra work required of the team in using the tool

6.2. Architecture

NextMove is implemented according to the client-server model. The client is implemented as an Eclipse plug-in that acts as the user interface. The server consists of the system logic (which performs calculations and produces rankings), as well as persistent storage. The client and server communicate to each other via the Java remote method invocation (RMI) mechanism. Persistent storage is implemented as a MySQL database.

6.3. Features

After a user starts the NextMove Eclipse plug-in and logs in, three Eclipse views are opened: the team backlog view, the personal backlog view, and the collaboration view.

The team backlog view consists of a list of tasks yet to be taken up by team members, ordered according to the recommendation of the system. The user can select a task and view more detailed information about it. If the user decides to take on a task by selecting the “adopt task” command, the
server will transfer the selected task from the team backlog to the personal backlog of the user.

The personal backlog view lists the tasks for which the user is responsible. The list is also ordered by the recommendation of the system. In the personal backlog view, the user has a number of options: the user can indicate which task s/he is currently working on, release a task (thereby putting it back into the team backlog), and indicate that a task has been completed.

The project manager has additional functionalities in the NextMove user interface. S/he can manually override the system and assign priorities to tasks, allocate tasks to team members, and delete tasks.

In the collaboration view, a user can observe other team members’ activities, such as recent commits. A user can also view other team members’ task lists, as well as the tasks that they have previously worked on.

7. Future Work

The proposed approach serves as a first step towards a solution that will assist project managers in making day-to-day task coordination and allocation decisions in an agile distributed development environment. There are several possible avenues for refining and extending the approach, discussed below.

The effectiveness of the approach can be evaluated through the use of NextMove in a real-life software development environment, where both quantitative and qualitative research can be conducted. Efficiency and quality of task allocation decisions from NextMove can be used to calibrate the constants used in calculation. Furthermore, users can offer feedback on the design of the tool’s interface and functionalities.

One important factor affecting the quality of decisions is the availability of accurate data, such as each team member’s abilities and the complexity of the different types of tasks. As an organization uses NextMove throughout the course of different projects, machine learning techniques or genetic algorithms can be used to generate more accurate data. For example, the system can be trained to learn the abilities of a team member by measuring how long it took the team member to perform certain tasks.

8. Conclusion

This paper outlines the issues in task coordination in an agile distributed software development environment and proposes an approach to assisting distributed teams in coordinating and allocating tasks. The approach makes use of current practices in software project management as well as methodologies, such as the analytic hierarchy process, from other fields of management. NextMove should help larger distributed teams to coordinate tasks without resorting to rigid planning and hierarchical team structures.

Acknowledgements

The authors would like to thank all colleague and friends who contributed to the paper by reviewing and giving feedback, as well as IBM and Eclipse for their support.

References