Context-Aware Task Allocation for Distributed Agile Team

Jun Lin
School of Computer Engineering
Nanyang Technological University, Singapore 639798
Email: jlin7@e.ntu.edu.sg

Chunyan Miao (Advisor)
School of Computer Engineering
Nanyang Technological University, Singapore 639798
Email: ascymiao@ntu.edu.sg

***Abstract*—The philosophy of Agile software development advocates the spirit of open discussion and coordination among team members to adapt to incremental changes encountered during the process. Based on our observations from 20 agile student development teams over an 8-week study in Beihang University, China, we found that the task allocation strategy as a result of following the Agile process heavily depends on the experience of the users, and cannot be guaranteed to result in efficient utilization of team resources. In this research, we propose a context-aware task allocation decision support system that balances the considerations for quality and timeliness to improve the overall utility derived from an agile software development project. We formulate the agile process as a distributed constraint optimization problem, and propose a technology framework that assesses individual developers' situations based on data collected from a Scrum-based agile process, and helps individual developers make situation-aware decisions on which tasks from the backlog to select in real-time. Preliminary analysis and simulation results show that it can achieve close to optimally efficient utilization of the developers' collective capacity. We plan to build the framework into a computer-supported collaborative development platform and refine the method through more realistic projects.**

***Index Terms*-distributed agile, task allocation, project management**

I. Introduction

Agile software development (ASD) is a relatively new software development paradigm that has gained popularity over the last decade. The typical process of assigning tasks in Agile is as follows: with the agile principle that teams are self-led, during the iteration each team member pulls out tasks they feel comfortable handling and assigns it to themselves. This pull model emphasizes on self-motivation of the team members compared to the push model of traditional software development processes, especially for a collocated agile team.

However, current task allocation approaches for agile are mostly empirical, depending on the developers intuitions and confidence. Recent studies [1], [2] have pointed out important limitations of ASD reported by practitioners in the field. The human and social factors that give small software development teams adaptability through ASD often hinder their performance in large software projects. In addition, practitioners have found it to be difficult for teams distributed over large geographic areas [2] to effectively adopt ASD. The discussions and coordination necessary for the team to agree on task allocations often need to be carried out in a face-to-face manner in order to be effective.

The idea behind daily stand-up meeting is to build common understanding of the situation and progress among team members. For example, when a new task arrives, the core questions of task allocation include whose skills are most suited to completing this task with high quality? And what effect will assigning this task to any given person have on the overall timeliness of the project? In essence, a good task allocation strategy should minimize both the risks of producing low quality work as well as failure to meet the deadlines at the same time. It is possible for effective solutions to be worked out when team members work together and are familiar with each other. However, for distributed or novice agile teams, it is difficult to make a timely decision that balance these considerations well.

II. Related Work

During the past decade, a number of solutions to the task-coordination and allocation problem have been proposed. Shim et al. [3] proposed a model-based methodology that allows the modeling of task-assignment policies. Their methodology makes use of UML-based meta-models to describe the four crucial elements of task assignment: organization, process, work products, and project. Shen et al. [4] proposed a multi-criteria task assignment model within workflow management systems. Their model describes qualitative measures, such as individual capabilities, using linguistic scales that are quantified by fuzzy numbers. Duggan et al. [5] proposed an optimization method for task allocation during the construction phase of software development. They make use of multi objective evolutionary algorithms (MOEA) and genetic algorithms (GA). Mak and Kruchten [6] proposed a two-step approach to solving the task-coordination and allocation problem in agile distributed software development environment. Their approach makes use of analytic hierarchy process (AHP) methodology. Yilmaz and O'Connor [7] introduced a market based mechanism to overcome task allocation issues in a software development process. They proposed a mechanism with a prescribed set of rules, where valuation is based on the behaviors of stakeholders (such as bidding for a task).

The problem and related works demonstrate the need for a method to facilitate task coordination and allocation decisions.

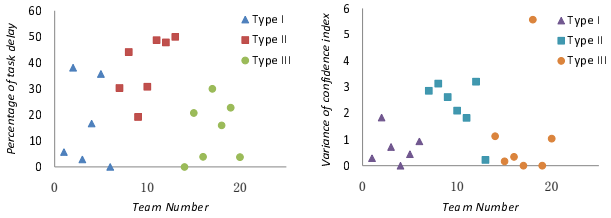
A well-conceived task allocation strategy for distributed or novice agile teams should utilize the team members' skills and capacity under changing environment and poor communication for producing high quality software on time. This requires some degree of automated system to collect and analyze the dynamic situations to assist developers' decision making. In order to avoid loss of agility of the process, the system also needs to produce decision support in real-time and minimize the overhead for developers to use it. However, existing methods have not fully addressed these requirements.

III. Preliminary Investigation

We have conducted an eight week group-based software engineering project as part of an undergraduate course work in Beihang University from 01/04/2013 to 31/05/2013. In this round of data collection activity, we have:

- 122 undergraduate students from Beihang are divided into 20 teams with an average team size of 6 persons. During the course work project, they have no common venue to work together on a daily basis. Thus, all teams are distributed. Team members possess similar skill levels and backgrounds.
- All teams adopt the ASD process to perform programming tasks. There are a total of 726 tasks during whole period. Tasks are divided among the team members at the beginning of each week based on internal discussions among each team.
- Each student may be assigned 0, 1, or multiple tasks during each iteration meeting. Each student reports his/her estimations of the difficulty of the task(s) assigned to or picked up by him/her and his/her confidence index of completing the task(s) with satisfactory quality in 10-point Likert scales, and the expected number of days needed to complete the task(s). Finally the team will use Planning Poker [8] technique to reach the consensus for task difficulty, confidence index and expected days.
- In the following iteration meeting, the timeliness of their completion relative to the expected time is evaluated by the members of each team together.
- At the end of the project, the performance of each team is graded by the course instructors with two scores representing the functional features of the software system developed and the time management of the team.

Based on the empirical data collected, it has been found that the task allocation strategies adopted by the teams can be grouped into three types: Type I - *equality* based group, their numbers of tasks per capita are almost same (variance less than 1); Type II - *mixed strategy* group, their variance of number of tasks per capita is between 2 to 5; Type III - *competence* based group, their numbers of tasks per capita very much depends on the agreed competence of the team members (variance greater than 5). Fig. 1(a) shows the percentage of task delay for each team. From the data we can observe that the strategy an agile team uses to allocate tasks has significant implications for its task completion rate, accordingly results in significant turbulence for the team confidence index shown in Fig. 1(b). This study has empirically confirmed one of the most significant limitations of the Agile software engineering process many researchers have pointed out: the effectiveness of open discussion based task allocation depends on how well the team members know each other and how experienced they are. The resulting strategy adopted may not necessarily be suitable for the situation facing the team.



(a) Distribution of the percentage of tasks delayed

(b) Distribution of variance of confidence index

Fig. 1. Analysis of Field Study Results

IV. Problem Formulation

Task allocation and workload balancing not only rely on information about internal properties of the tasks (such as priority, utility, difficulty and effort level required etc.), but also on human factors not depicted in product backlog (such as developers' workload, competency, confidence and even psychology stress, etc.). To efficiently utilize the capacity embedded in an agile team, task allocation plans should consider the trade-off between maintaining software quality (which tends to result in more tasks being allocated to highly competent team members) and delivering the software product on time (which needs tasks to be allocated to team members with less workload at the moment).

To achieve this goal, we first formalize the agile team management problem into a distributed constraint optimization problem (DCOP) [9]. The overall objective of ASD is to develop high quality software product according to the customer's requirement within the stipulated deadline, which can be expressed as maximizing a *quality-minus-delay* expression for all team members.

The quality aspect depends on a wide range of factors innate to each team member. In our theory analysis and simulation, we abstract them into key variables represented by the 3-tuple $\langle M_i(t), C_i^\tau(t), E_i^{max} \rangle$. $M_i(t)$ denotes a team member i's morale status (e.g. confidence of completing a task) at time t, $C_i^\tau(t)$ represents i's competence in performing task of type τ (e.g., C# programming), and E_i^{max} is the average total effort i can spend on performing tasks per unit time.

To minimize the *delay*, two conditions must be satisfied: 1) i's pending task queue $Q_i^\tau(t)$ should not be allowed to grow indefinitely for all τ and t, and 2) $Q_i^\tau(t)$ should not be too short (i.e., i is idling). Based on i's internal context, we can derive its target workload for each type of task q_i^τ which positively correlates to $\langle M_i(t), C_i^\tau(t), E_i^{max} \rangle$. Thus, the objective of minimizing delay is equivalent to minimizing the *drift* of the collective workload in an agile team from its collective target workload. Therefore, the objective function

now becomes maximizing the *quality-minus-drift* expression for all members.

The properties of a task of type τ in ASD are represented by the 3-tuple $\langle p^\tau, u^\tau, e^\tau \rangle$. p^τ denotes the priority of this type of tasks and will affect their positions in the task backlog $Q(t)$. u^τ represents the utility that can be derived by the team from successfully completing a task of type τ on time. e^τ is the expected effort that needs to be expended to complete the task. Since tasks are often divided into small and more manageable pieces by the project manager in ASD, we can assume that u^τ and e^τ for all the tasks belonging to the same type τ are equal. The utility that can be expected from letting i perform task j can at time t, thus, can be expressed as:

$$utility(i, \tau, t) = u^\tau \cdot C_i^\tau(t) \cdot M_i^\tau(t) \tag{1}$$

The queuing dynamics for any pending task queue $Q_i^\tau(t)$ is:

$$Q_i^\tau(t+1) \leftarrow \max[Q_i^\tau(t) + \alpha_i^\tau(t) - \mu_i^\tau(t), 0] \tag{2}$$

where $\alpha_i^\tau(t)$ denotes the number of new tasks of type τ admitted into $Q_i^\tau(t)$ at time t, $\mu_i^\tau(t)$ is the number of tasks completed by i at time t. Therefore, the expected *quality* from a task acceptance decision can be expressed by:

$$quality(i, \tau, t) = \alpha_i^\tau(t) \cdot utility(i, \tau, t) \tag{3}$$

The *drift* is positively correlated to both $\alpha_i^\tau(t)$ and $\mu_i^\tau(t)$ (i.e., increase in either of them causes the *drift* to increase). Based on the principle of *Lyapunov drift* [10], we have:

$$drift(i, \tau, t) = \alpha_i^\tau(t) \cdot \mu_i^\tau(t) \tag{4}$$

Eq.(4) can be trivially minimized by assigning both $\alpha_i^\tau(t)$ and $\mu_i^\tau(t)$ to 0 for all i, τ and t (i.e., all team members stay idle all the time). However, this is not a valid solution in practice. Through the above analysis, we can formalize the objective of managing an agile team as:

Maximize:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{i,\tau} \Psi quality(i, \tau, t) - drift(i, \tau, t)$$

$$= \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{i,\tau} \alpha_i^\tau(t)[\Psi u^\tau C_i^\tau(t) M_i^\tau(t) - \mu_i^\tau(t)] \tag{5}$$

Subject to:
$$\sum_{\tau} (\alpha_i^\tau(t) \cdot e^\tau) \leq E_i^{max} \tag{6}$$

$$0 \leq \alpha_i^\tau(t) \leq \lambda^\tau(t) \tag{7}$$

where $\lambda^\tau(t)$ is the number of new tasks of type τ being added into the task backlog $Q(t)$ by the project manager at time t, and Ψ is a weight variable indicating the relative importance given to *quality* and *drift* while maximizing Eq.(5).

Algorithm 1 SMART

Require: $[\Psi u^\tau C_i^\tau(t) M_i^\tau(t) - \mu_i^\tau(t)]$ values for all τ for an agile team member i, the incoming tasks $\lambda^\tau(t)$ for all τ, and E_i^{max}.

$e_i(t) = E_i^{max}$
for each $Q_i^\tau(t)$ in i in descending order of its $[\Psi u^\tau C_i^\tau(t) M_i^\tau(t) - \mu_i^\tau(t)]$ **do**
  if $[\Psi u^\tau C_i^\tau(t) M_i^\tau(t) - \mu_i^\tau(t)] > 0$ **then**
    if $\lambda^\tau(t) \cdot e^\tau \leqslant e_i(t)$ **then**
      $\alpha_i^\tau(t) = \lambda^\tau(t)$
    else
      $\alpha_i^\tau(t) = \lfloor \frac{e_i(t)}{e^c} \rfloor$
    end if
    $e_i(t) \leftarrow e_i(t) - \alpha_i^\tau(t) \cdot e^\tau$
  else
    $\alpha_i^\tau(t) = 0$
  end if
end for
Return($\alpha_i^\tau(t)$ for each $Q_i^\tau(t)$ in i)

V. Proposed Approach

With this formalization, we have developed an algorithm called *SMART* to find solutions for all $\alpha_i^\tau(t)$ whenever new tasks are proposed so as to maximize Eq.(5) subject to Constraints (6) and (7). *SMART* proceeds as illustrated in Algorithm 1. During one sprint cycle, the algorithm makes use of current task related information generated during sprint assessment phase (or in sprint planning meeting) and task completion information generated during previous sprint review phase/meeting, and characteristics of team members to produce a task allocation plan for allocating the sprint backlog to aid the team's decision. Some characteristics such as personal skills and competence can be obtained by self-reported survey and past performance. Current morale status such as confidence or interest to each task needs to be provided by developer in every sprint assessment phase, and current workload can be calculated by system automatically. Generally speaking, this design does not require much extra effort from the developers.

VI. Experimental Evaluation

As the problem in this study is relatively new, there is no existing dataset that can be used to comprehensively evaluate SMART. In addition, real project data are useful for designing a realistic experiment environment, but often lack the ground truth about the behavior patterns of the team members. In order to evaluate SMART under different circumstances, and to provide more flexible control of the team members' behavior, we implement it within a simulated multi-agent environment [11]. The task allocation strategies adopted by the agents in our simulations are designed based on the data collected our preliminary investigation in Section III.

A. Experiment Design

The member agent population in simulation consists of 20-160 agents exhibiting behavior patterns belonging to four categories according to their competence and high quality task result. By adjusting the number of different types of agents and tasks, we simulate agile teams with different developer compositions to perform 500-5000 tasks in 100 days.

Two simulated multi-agent systems are run in parallel. In one of them, agents adopt the general accept-when-requested approach for handling incoming task requests. In the other, agents adopt the proposed SMART approach for handling incoming task requests. The results from these two sets of experiments are labeled as AWR and SMART respectively in the following figures. Each simulation is repeated 10 times to reduce the effect of random variations in the system.

B. Results and Analysis

Fig. 2 compares the proportion of task allocated to agents for agile teams of various sizes and compositions over 100 time steps under both AWR and SMART. From it, we can see that under AWR, the agent's allocated task proportion fluctuates significantly. The sequence of event is: during the development process, agents with high competence will be allocated a large number of tasks. The influx of tasks resulted in long backlog in their task queues. On the other hand, many other agents are in idle status. The SMART approach avoids this issue and keep members' task queue in an average level while still assigning more tasks to competent agents without overworking them. Other simulations using various team sizes and compositions also produced similar results.

Fig. 3 shows the global utility achieved by given different agent populations under AWR and SMART. In it, symbols in the x-axis are interpreted as follows: *S*, *M* and *L* denotes small, medium and larges team sizes respectively; *hi*, *mc* and *hc* denotes highly incompetent, mediumly competent and highly competent teams respectively. By reducing the adverse effect of uneven task allocation and the corresponding inefficient utilization of team resources caused by the existing approach through efficient utilization of developer context-aware agents' capacities, the agile team equipped with the SMART approach consistently achieved significantly higher utility than the AWR approach for all team sizes and compositions. As during the process, SMART always adapts its plans to make sure that: 1) high utility tasks will be performed by high competent members; and, 2) tasks assigned to team members will not overwhelm them based on their past observed performance.

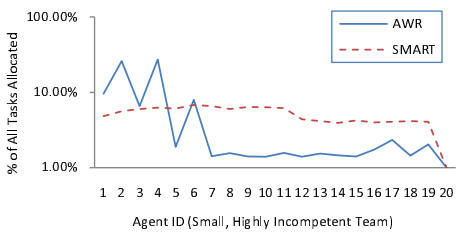


Fig. 2. Task allocation for Small-Highly Incompetent teams

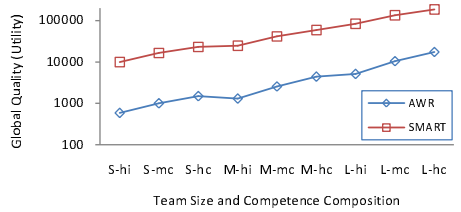


Fig. 3. Comparisons of global utility achieved by AWR and SMART

VII. Conclusions and Future Work

In this paper, we propose a context-aware task allocation approach to balance workload of team members in distributed agile team based on their individual situations. SMART does not intend to completely replace all aspects of task allocation in daily communications. As a component, the focus of SMART is to support context-aware, and distributed task allocation from the perspectives of both tasks and team members.

In subsequent research, we will build a multi-agent computer-supported collaborative development platform equipped with SMART to assist individual developers in distributed agile teams to make situation-aware decisions on which incoming tasks to handle so as to reduce the risk of low quality task results while maintaining the timeliness of the overall software project, and conducting some large-scale studies in university.

Acknowledgments

This research is supported in part by Interactive and Digital Media Programme Office, National Research Foundation hosted at Media Development Authority of Singapore (Grant No.: MDA/IDM/2012/8/8-2 VOL 01).

References

[1] F. Shull, T. Dyba, and T. Dingsoyr, "What do we know about agile software development?" *IEEE Software*, vol. 26, no. 5, pp. 6–9, 2009.

[2] F. Sallyann and S. Helen, "Currents trends, people, projects the top 10 burning research questions from practitioners," *IEEE Software*, vol. 27, no. 5, pp. 8–9, 2010.

[3] J. Shim, S. Lee, and C. Wu, "A unified approach for software policy modeling: Incorporating implementation into a modeling methodology," *Lecture Notes in Computer Science*, vol. 2813, pp. 118–130, 2003.

[4] M. Shen, G.-H. Tzeng, and D.-R. Liu, "Multi-criteria task assignment in workflow management systems," in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, 2003, pp. 6–9.

[5] J. Duggan, J. Byrne, and G. Lyons, "A task allocation optimizer for software construction," *IEEE Software*, vol. 21, no. 3, pp. 76–82, 2004.

[6] D. K. M. Mak and P. B. Kruchten, "Task coordination in an agile distributed software development environment," in *Proceedings of the 2006 Canadian Conference on Electrical and Computer Engineering*, 2006, pp. 1625–1630.

[7] M. Yilmaz and R. O'Connor, "A market based approach for resolving resource constrained task allocation problems in a software development process," in *Proceedings of the 19th European Conference on Systems, Software and Services Process Improvement (EuroSPI 2012)*, 2012.

[8] J. Grenning, "Planning poker or how to avoid analysis paralysis while release planning," Hawthorn Woods: Renaissance Software Consulting, Tech. Rep., 2002.

[9] B. Faltings and M. Yokoo, "Introduction: Special issue on distributed constraint satisfaction," *Artificial Intelligence*, vol. 161, no. 1-2, pp. 1–5, 2005.

[10] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan and Claypool Publishers, 2010.

[11] J. Ferber, *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*. Harlow: Addison Wesley Longman, 1999.