Investigating the relationship between project constraints and appropriate iteration length in agile development through simulations

Abstract: Agile development is aimed at minimizing overall risk and encouraging rapid and flexible response to specification changes by using an iterative process. Despite its iterative feature, studies on the effects of iteration length have been lacking. Currently, there is no established method to quantitatively determine the appropriate iteration length, and abortion of projects with an inappropriate iteration length has been reported. We therefore create a model of agile development that focuses on iteration length, and propose a method of simulating a particular project to estimate the appropriate iteration length. Furthermore, we simulate diverse situations using various parameters to understand the relationship between the iteration length and project constraints. Our results show that the appropriate iteration length depends on the condition of the project constraints; the larger the amount of uncertainty, the shorter the appropriate iteration length, while the higher the complexity of the project, the longer the iteration length should be.

Keywords: Agile Development, Iterative Software Development, Extreme Programming, Scrum, Simulation, Iteration Length, Software Development Process

1 Introduction

Agile development is aimed at minimizing overall risk and encouraging rapid and flexible response to specification changes by using an iterative process. In recent years, efforts have shifted from looking at agile development as a methodology (doing agile) to looking at it as a style of working where the developers adhere to its founding values (being agile). However, much of the recent studies are still focused on doing agile (Dyba, T., and Dingsoyr, T., 2008), such as the applicability of practices and performance comparisons with waterfall development (Melis et al., 2006; IPA, 2010).

In software engineering, it is said that there is no silver bullet for managing software development because no project has the same constraints (Brooks, 1986; 1997). This is true for development processes as well. Therefore, the parameters of a development processes must be adjusted for each project to improve its adaptability. Specifically, in agile development, there are numerous parameters that should be adjusted because there is a large number of practices that can be adapted.

One of the parameters of agile development is iteration length, but we found that there have been few studies on the iteration length; only qualitative values have been recommended, such as a week for extreme programming and a month for Scrum. Therefore, practitioners must decide the iteration length exclusively from their own experience, and abortion of projects due to inappropriate iteration length

has been reported.

In this paper, we create a model that focuses on iteration length and propose a method of simulating a particular project to estimate the appropriate iteration length. Furthermore, we conduct simulations in diverse situations by varying parameters to understand the relationship between the iteration length and project constraints.

The results of our research show that the appropriate iteration length depends on the condition of the project constraints, such as the amount of uncertainty and the level of complexity. Increased uncertainty shortens the appropriate iteration length, while increased complexity lengthens it.

There are three contributions in this paper: a model for estimating the appropriate iteration length, a method for simulating a particular project, and the investigation of the relationship between project constraints and the appropriate iteration length.

The remainder of the paper is organized as follows. Section II describes the problem of deciding an appropriate iteration length in agile development. Section III explains the proposed methodology. Section IV gives the results of simulations and their implications. Section V briefly discusses related work. Section VI gives a summary of the paper along with future works.

2 Deciding the iteration length

As we have mentioned above, agile development is an iteration-based development process, and it is not an exaggeration to say that the success of a project depends on the iteration length. The iteration length influences the scope of iteration (Anderson et al., 2011), how changes are managed and implemented and how the development cycle is generated.

However, there have been few studies on the iteration length, and only general guidelines have been suggested, such as a week for extreme programming and a month for Scrum. Therefore, managers are left to make decisions on the iteration length exclusively from their own experience. An incorrect decision may lead to project failure as explained below.

On the one hand, an excessively long iteration length reduces the opportunities for obtaining feedback from customers and makes it difficult to deal with specification changes. Furthermore, it increases the scope of iteration and makes the development more complex. On the other hand, an excessively short iteration length increases the numbers of iteration planning phases, thus increasing overheads and leading to cost escalation. In the meantime, developers may need to split the requirements to adjust each task size to the iteration length. Although this can reduce the complexity of development, it increases integration costs.

3 Estimation method of appropriate iteration length

In this section, we present our proposed method for estimating the appropriate iteration length. We explain our choice of the simulation methodology, and describe the extraction of the common agile development model, the static structure of the

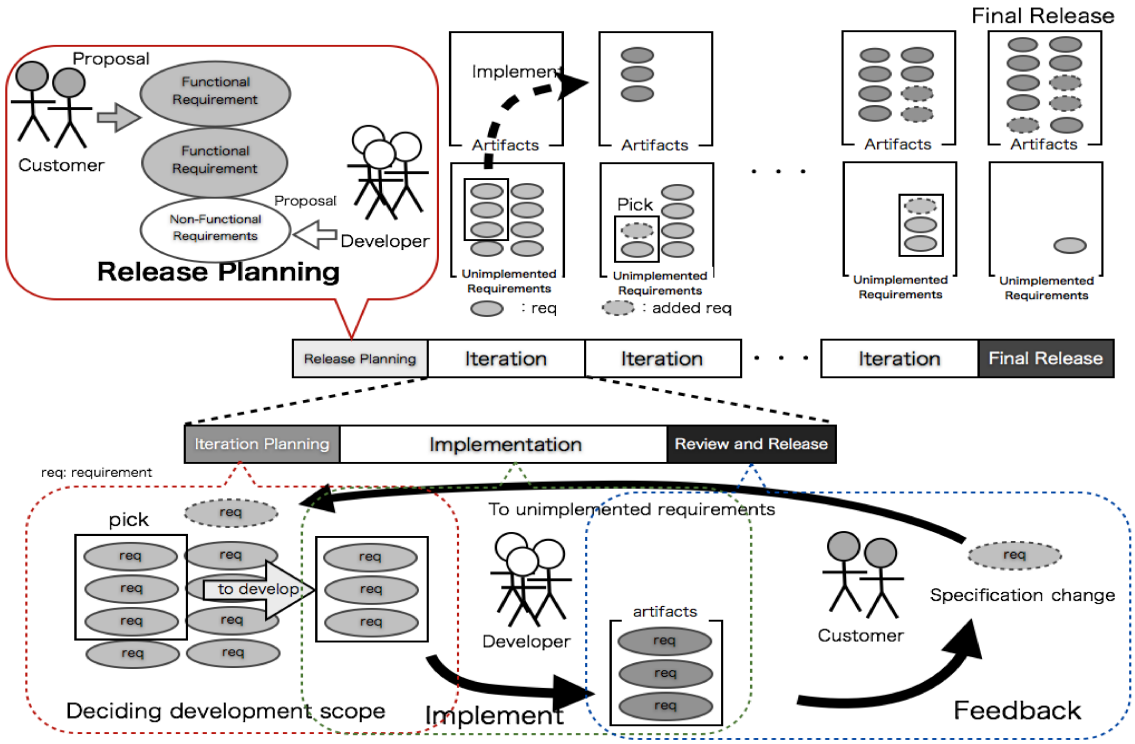

Figure 1 Common model of agile development

simulator model, and the dynamic behavior of the simulator model.

3.1 Reason for selecting simulation methodology

There are two main methods of investigating development processes: observing real projects and finding general principles by using statistical methodologies; and creating a model and using it to understand or estimate the real phenomena.

The former method requires the observation of an enormous amount of data to find the principles, considering the possibility that the data includes artificial errors. On the contrary, in the latter way, a model can be constructed from the results of previous studies or well known events. We determined the latter method to be more suitable in the case of agile development, although the results can only be applied to situations within the scope of the model and must be validated using some experimental researches. We use two case studies for validation in Section Ⅳ.

3.2 Extraction of the common agile development model

To create the simulator model, we first establish the general process model of agile development. Figure 1 shows the model created by combining the extracted common features of extreme programming and Scrum; extreme programming and Scrum are known to be the most frequently used agile development methodologies

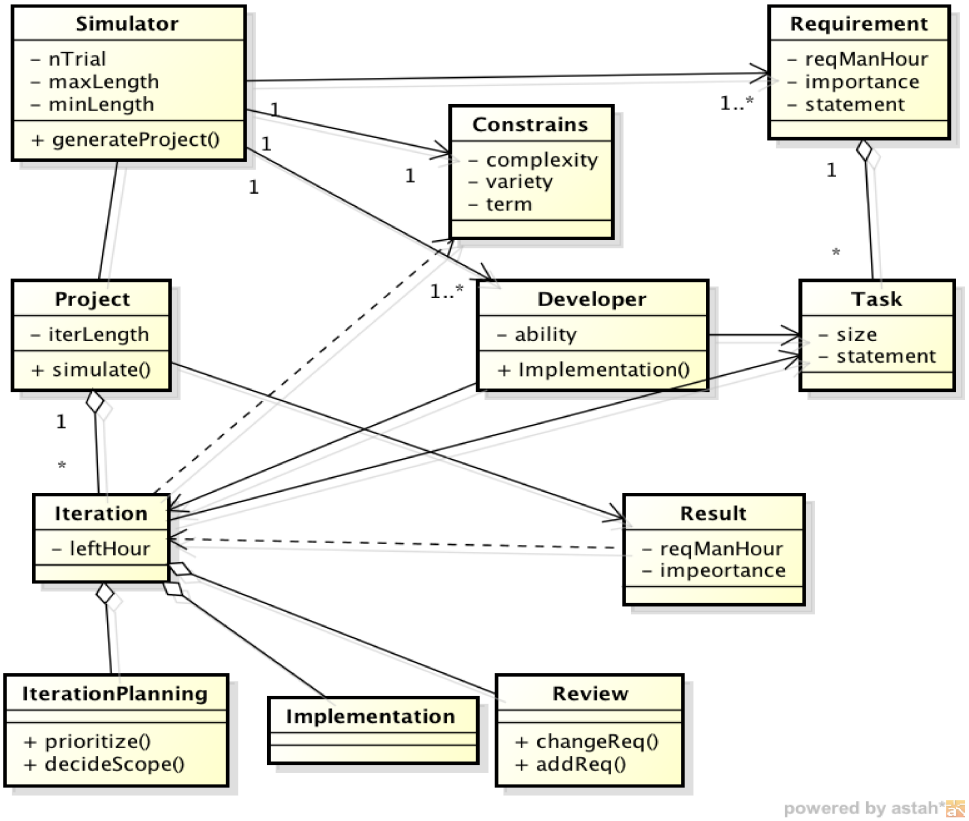

Figure 2 Class diagram of the simulator

worldwide.

There are two main roles shown in Figure 1, customers and developers. Briefly, customers order requirements to the developers and review artifacts, while developers implement the requirements to demonstrate artifacts to the customers.

At the beginning of a project, the customers put forth initial requirements of the project and the developers suggest additional requirements that are necessary (Release Planning). Then the implementations of the requirements begin on an iteration basis. Each iteration has three phases: iteration planning, implementation and release and review(Schwabe and Beedle, 2001).

In the iteration planning phase, unimplemented requirements are ordered according to priority, and the top ones are chosen as the development scope of the iteration. These chosen requirements are then implemented in the implementation phase, and specifications are changed on the basis of customer reviews on artifacts of the iteration in the release and review phase.

3.3 Static structure of a simulator model

As mentioned above, analysis of an entire software development project requires enormous data. However, a project can be divided into smaller parts, and the inves-

Table 1 Parameters used in the simulator

Parameter name		**Values**	Unit
Project Constraints	Duration	Integer greater than or equal to 1	days
	Uncertainty	0, 5, 10, 15, 20, 25, 30	%
	Complexity	25, 50, 75	%
Ability of developer		0.25(Beginner), 1(Intermediate), 2.5(Advanced)	unitless
Requirement	Estimated effort	Integer greater than or equal to 1	man-days
	Importance	Integer greater than or equal to 0	unitless

tigation of one of these parts is much easier; there are a number of previous studies investigating the features of partial processes of software development projects. We refer to these previous studies and combine them with the model described above to construct the simulator model.

Figure 2 shows a class diagram of the simulator. The simulator creates a number of trial projects based on the given Parameters: Constraint, Developers, and Requirements. Each Project involves a number of Iterations and produces a Result. Iterations are constructed by IterationPlanning, Implementation, and Review; Requirements have multiple Tasks to implement.

As shown in Figure 2, we use five parameters to identify a project feature. Because the simulator model is only for estimating the appropriate iteration length, we have eliminated some parameters to avoid unnecessary complexity. We have chosen the parameters based on a study by the Information-technology Promotion Agency, Japan on non-waterfall development (IPA, 2009). Table 1 shows the parameters.

Duration

Duration refers to the number of days from the first to the last day of the project, excluding non-work days. We assume that the first day of the hypothetical project is day one of the first iteration; our method expects the iterations to start immediately after the initial requirements have been set.

Uncertainty

The uncertainty describes the probability of a specification change. We premise that a customer can add, modify, or remove requirements as a specification change. It is impossible to know the exact amount of uncertainty, but it can be estimated by considering the novelty, area, and concreteness of the project on the basis of experiences. Additionally, specification changes occur for two reasons: customer feedback based on the artifacts, and the volatility of the market or advances in technology. For simplicity, we suppose that uncertainty results in both types of changes. The former is incorporated using the model discussed in (Nakatani, 2006); the latter is added in by considering the elapsed time from the beginning of the project.

Complexity

Complexity indicates the probability of generation of dependencies among requirements. As shown in Figure 3, a high complexity project corresponds to a large number of dependencies among requirements. Integrity costs are generated when the requirement that is currently being implemented or changed

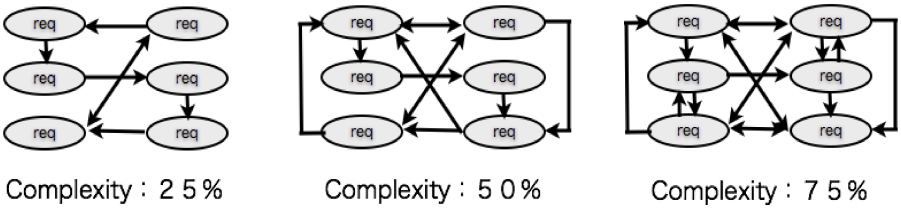

Figure 3 Relationship between complexity and volume of dependencies

depends on another requirement that has already been released. Integrity is considered as extra tasks; the tasks are added to the requirement. The explanation of the task is mentioned below. Similar to variety, the complexity cannot be determined before the project starts, meaning that it should be estimated by considering the area and scope of the project on the basis of past experiences.

Ability of developer

Developer abilities indicate the development ability of the developers who are working on the project. To determine a basic value of developer ability, we consider two previous reports on how developer ability affects productivity, (Boehm and Papaccio, 1988) and (Jones, 2000). In the former study, it was concluded that there is an approximately 28-fold difference between the productivity of a beginner and an advanced developer. In the latter study, the evolution of integrated development environment, object oriented programing, web frameworks, and testing frameworks were found to affect software development productivity. Additionally, it was determined that there is a 10-fold difference between beginner and advanced developers, and a 2.5-fold difference between intermediate and advanced developers. We considered the latter study to be more suitable for the case of agile development, and classified the ability of the developer into three levels: beginner, intermediate, and advanced.

Estimated effort of requirements

This is the estimated amount of labor required to implement the requirements initially given by the customer. The ability of an intermediate developer is used as the unit.

Importance of requirements

This indicates the relative importance of the requirements initially given by the customer. For example, let us consider a contents availability management system that has four requirements: "searching for contents", "registering contents", "sorting results", and "auto completion". The requirements "searching for contents" and "registering contents" are the essential parts of the system and should have an importance of 8 to 10. Requirements that are not essential but important, such as " sorting results" should be marked between 5 and 8. Additional requirements such as "auto completion" should be marked between 1 and 5.

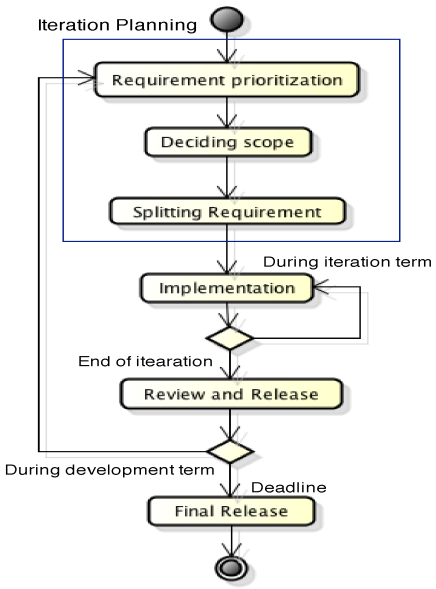

Figure 4 Dynamic behavior of the simulation

The simulator uses the above parameters to specify the project features and calculates the results as described below. The appropriate iteration length is estimated by comparing the results of different projects. The simulator results are evaluated in a comparable method, so we can use relative values for the importance of a requirement.

Progress
: Progress is given by the sum of the values of importance of the requirements that have been implemented at the end of the project.

Cost
: Cost is the total number of man-hours of work including the customer and developer man-hours spent in iteration planning. We do not focus on the material cost and equipment cost because they do not depend on the iteration length.

Progress / Cost
: The value of Progress divided by Cost is calculated as a measure of cost-effectiveness.

3.4 Dynamic behavior of the simulator model

The dynamic behavior of the simulator model is shown in Figure 4, and its details are as follows.

1. There are three phases in iteration planning: (a) requirement prioritization, (b) deciding the scope, and (c) requirement splitting.

(a) Requirements are prioritized in accordance with their importance and the dependencies. Requirements are first sorted by importance, and then for each requirement, its priority is raised if it depends on another higher priority requirement. This strategy originates from the practice of value-driven development (Larman, 2003) and the description, "working software over comprehensive documentation", from the agile manifest(Kent et al, 2001)

(b) To decide the development scope, generally, a scope-box method and a time-box method should be considered. The former is a means of deciding the length of the iteration from the development scope and the latter is a means of deciding the development scope from the iteration length. In our method, we use the time-box method because we need to assume that the iteration length is constant for the entire project to observe the effects of iteration length. Therefore, we decide the development scope from the iteration length, number and ability of the developers, and the average estimated effort of unimplemented requirements.

$$DevelopmentScope = \frac{L * \sum_{i=1}^{n} Da_i * m}{\sum_{k=1}^{m} Re_k}$$

L: iteration length, n: number of developers, m: number of requirements, Da: ability of Developer, Re: estimated effort of requirement

(c) In our method, requirements are defined in terms of their functionality, regardless of whether they are functional or non-functional. In the requirement splitting phase, the requirements are divided into tasks based on their estimated effort. To avoid a procedure error due to requirement splitting, we fix the size of each task to 0.5-2 man-days, which was previously reported as an appropriate value (Larman, 2003).

The time spent on iteration planning depends on the project, the customers, and the developers. For modeling purposes, however, we assumed that it only depends on the iteration length and the number of developers because dividing requirements into tasks and assigning those tasks to the developers would be the most time consuming steps of the phase.

2. In the implementation phase, developers implement the tasks that are selected as the development scope in iteration planning. Figure 5 shows a state transition diagram of the implementation. A developer can be in one of two states: empty or assigned. Each task can be in one of three states: to-do, doing, or done. The state of both the developer and the task is updated every hour, and the rules of the state changes are as follows.

   - A developer in the empty state:
     - If there is a task in the to-do state, the developer is assigned to the task. This results in the developer state changing to assigned, and the task state changing to doing.
     - If there is no task in the to-do state, nothing happens.
   - A developer in the assigned state:

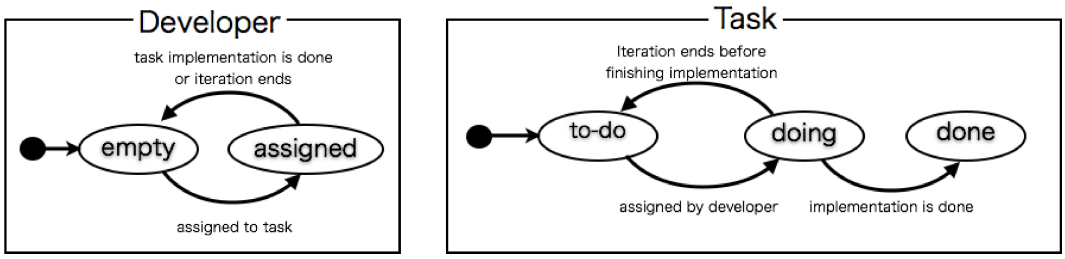

Figure 5 State transition diagram of implementation

- If the task is still incomplete, the developer continues to implement the assigned task; there are no state transitions.
- If the task has been fully implemented, it enters the done state, and the developer returns to the empty state.

Task implementation is processed as follows.

$$S_{after} = S_{present} - (C * \frac{D_a}{8})$$

S: task size, C: volatility of development efficiency which is randomly selected from 0 to 2.0, D_a: ability of developer

In our method, a developer works 8 hours a day and never works overtime. This is based on the extreme programming practice 40-hour work weeks. Therefore, the total time of the implementation phase is given by following.

$$T = L * 8 - T_{ip}$$

T: iImplementing time, L: iteration length, T_{ip}: iteration planning time

For example, in the case of 7 days per iteration and 6 hours of iteration planning, the total implementation time is 50 hours. A task that is in the done state is considered as an implemented task. The requirement that all tasks are implemented and that all of the requirements that it depends on have already been implemented becomes the artifact of the iteration.

3. In the review and release phase, specifications are changed or implemented requirements are accepted. The probability of a specification being changed depends on the project uncertainty. When a specification is changed, multiple tasks are added to the implemented requirement, or new requirements are added to the project. In the latter case, the new requirements have higher importance on average than the initial requirements because the customer should be getting familiar with the project. Meanwhile, the number of added requirements is limited by the initial number of requirements and project uncertainty, based on a requirements elicitation model described in a previous study (Nakatani, 2009).

4. The above iterations are repeated until the last day of the project, at which point the final release phase starts. In this phase, the sum of the number of implemented requirements and the total number of man-hours of work are calculated and output as the result. Unimplemented requirements and incomplete requirements are not considered as artifacts.

3.5 Use of our method

Our method can be used to help decide the iteration length of an actual project. The expected users are project managers or product managers who are required to decide the iteration length. The results of the simulation can be used to decide the iteration length before starting the project, or to review the project after its completion by comparing the project results with those obtained from the simulator. It can also be used during the course of a project for reconsidering the iteration length, particularly when an unexpected accident occurs or when progress is delayed. It should be kept in mind that even though we provided instructions for extracting parameters, there are variabilities in the parameters that may change the result. We will discuss how to input the parameters in Section VI as future work.

4 Results of simulation and discussion

First, we validate the simulator model using case studies of two projects. We simulate the projects using our method and compare the results with actual reviews from the stakeholders of the projects. Since the projects have already been completed, we extract the parameters from the actual project history to make the experiments more accurate. Second, we vary the parameters to create different situations and analyze the relationship between the project constraints and the appropriate iteration length by using the validated simulator model. Since we use the Monte Carlo method in the simulator to conduct 10,000 simulations for each iteration length to minimize the error effect; the given results are average values of them. The experiment has been done on a computer with Mac OSX 10.6.7, 2.3GHz Intel Core-i5 and 8GB 1333 MHz DDR3.

4.1 Web application project for a patient management system (case study 1)

We use the XP Practice Report from Eiwa System Management, Inc. as case study 1 (Eiwa System Management, Inc, 2006). This public report describes a project that was launched to evaluate agile development, and gives detailed project constraints and reviews from the developers about the iteration length. In this project, a Web application for a patient management system is developed in extreme programming with five developers: a beginner programmer who joined the company the previous year, two intermediate-level programmers who have worked at the company for a few years and two advanced level-programmers.

This project has a development time of one month. Since the project uses extreme programming as the development process model, it adopts an iteration length of a week (five days). Therefore, this project involves four iterations. Initial

Table 2 Extracted parameters for case study 1

Name	Value
Development term	20
Uncertainty	10%
Complexity	50%
Developers	0.25, 1, 1, 2.5, 2.5 (five people)
Estimated effort of requirements	10, 10, 15, 10, 10, 10, 5 man-days
Importance of requirements	0.3, 0.6, 1, 0.3, 0.6, 0.6, 1

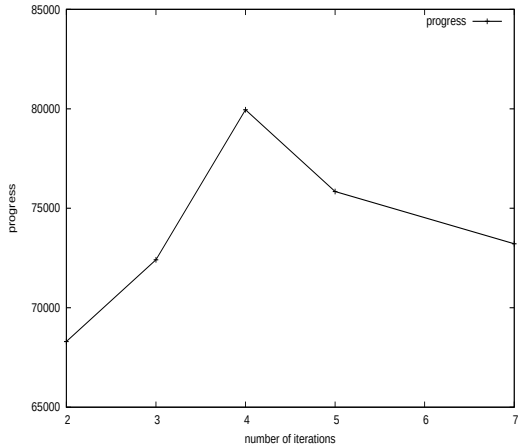

Figure 6 Simulation results for case study 1. x: number of iterations, y: progress

requirements are constructed from seven main user stories; some requirements are gradually fixed through feedback from the customer. From the above conditions, we extract the parameters for the simulator, which are shown in Table II. We calculate the uncertainty from the actual specification changes because details on the uncertainty and complexity are not given. Moreover, we estimate the number of dependencies between the requirements from the user story names, and determine the complexity to be its reciprocal.

Figure 6 shows the progress fluctuations of the simulations of case study 1. The x-axis shows the number of iterations and the y-axis shows the progress valuef. We can see that the progress value is highest for the case of four iterations, or five days for each iteration. In this model, normally, progress should be higher when the number of iterations is increased because it increases the opportunity to accept the requirements which are found later. In this case, however, the progress for five and seven iterations is lower than that for four iterations. This indicates that an extremely small scope reduces productivity in high complexity projects.

Figure 7 shows the cost fluctuations of the simulations of case study 1. The x-axis shows the number of iterations and the y-axis shows the cost value. Cost tends to be reduced when the number of iterations is decreased, except for the case of five iterations where the cost dips lower. This implies that the total labor times

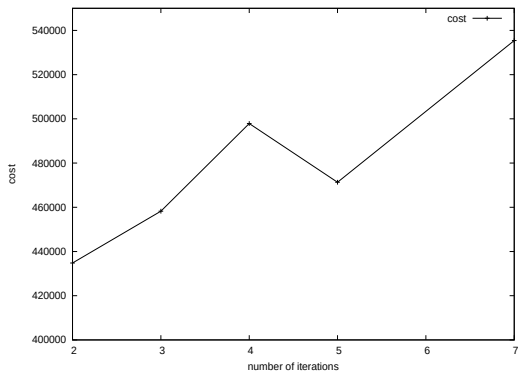

Figure 7 Simulation results for case study 1. x: number of iterations, y: cost

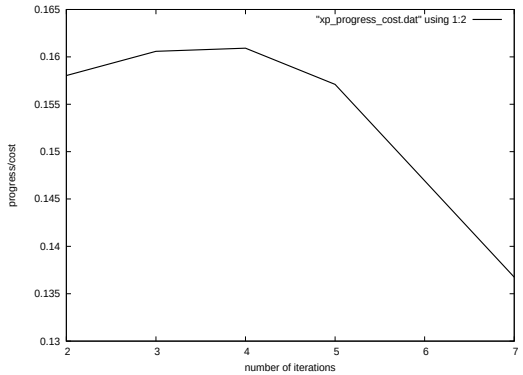

Figure 8 Simulation results for case study 1. x: number of iterations, y: progress/cost

for development are almost the same except for the case of five iterations (the cost differences are made by the time required for iteration planning). We suppose the following two points contributed to the lower cost for five iterations.

- In our model, small development scope increases development efficiency. Under the case of five iterations, the development efficiency is higher than the case of four iterations and it decreases the labor times for development.

- In our model, integration costs are considered as development task. Under the case of seven iterations, integration costs are higher than those of the case of five iterations.

Figure 8 shows the value of progress divided by cost for the different numbers of iterations. It is highest in the case of three and four iterations which agree with a comment in the review of the project that the iteration length was slightly shorter than appropriate, even though the project succeeded. Therefore, this demonstrates that the simulator works well for this project.

Table 3 Extracted parameters for case study 2

Name	Value
Development term	100
Uncertainty	25%
Complexity	25%
Developers	2.5 (a person)
Estimated effort of Requirements	5, 5, 10, 10, 15, 10, 10, 10, 5, 10, 10, 15, 10, 10, 10, 10man-day
Importance of Requirements	0.3, 0.3, 0.6, 0.6, 1, 0.6, 0.6, 0.6, 0.3, 0.6, 0.6, 1, 0.6, 0.6, 0.6, 0.6

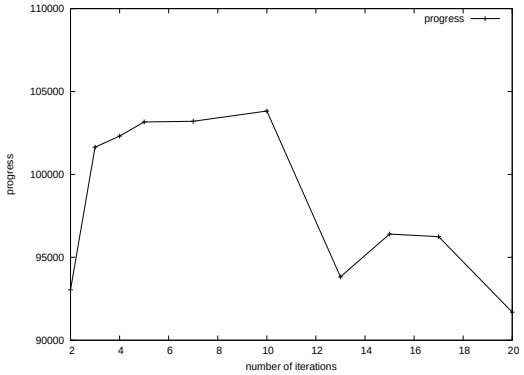

Figure 9 Simulation results for case study 2 x: number of iterations, y: progress

4.2 Correspondence education system project (case study 2)

For case study 2, we consider a project on a correspondence education system carried out by Probizmo Co, Ltd. This project was reported in an IT development plan of Shimane prefecture.

In this project, the development term was 100 days and the developer was an intermediate level programmer who used Scrum. There were 16 initial requirements, and the iteration length was a month because Scrum was used, meaning that this project was completed in four iterations. Regarding the project features, it was reported that the complexity was very low with few specification changes. The extracted parameters are presented in Table Ⅳ.

Figure 9 shows the progress of the simulations for case study 2. Progress increases significantly for less than ten iterations except for the case of two iterations. This indicates that for more than twelve iterations, the iteration length may have been too short for the complexity of the project, or that the amount of specification change may have exceeded the limitation.

Figure 10 shows the cost of the simulations of case study 2. Cost tends to be reduced by decreasing the number of iterations. The values of progress divided by cost for the case study 2 are shown in Figure 11. The highest values are observed for three or four iterations. According to an answer from a stakeholder for the question, "Do you have any particular comment on the overall development project?", the

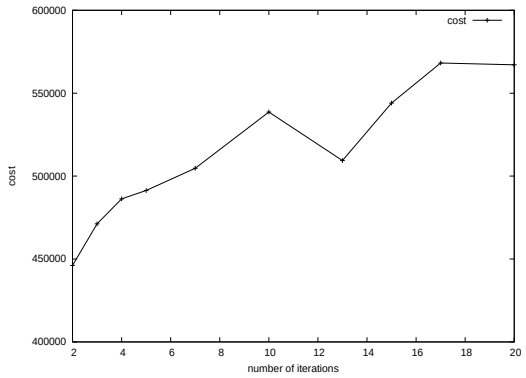

Figure 10 Simulation results for case study 2 x: number of iterations, y: cost

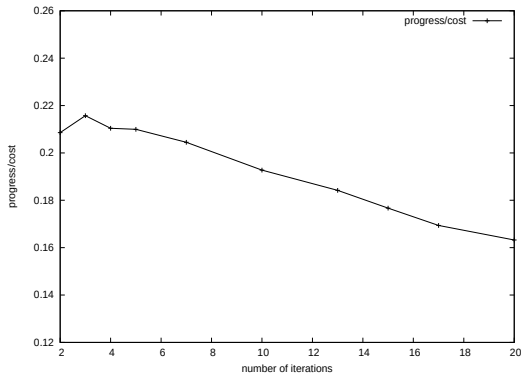

Figure 11 Simulation results of case study 2 x: number of iterations, y: progress/cost

project was successfully completed in four iterations, and the iteration length fit the project. Note that this project was done by one developer; even if this project used the agile development process, it is not representative of a non-trivial software project. Therefore, this result is limited to projects with one developer and can not be simply applied to a team development. However, this shows that the simulator works reasonably well at least for this project.

4.3 Relationship between project constraints and iteration length

With the simulator validated by case studies as above, we next varied the parameters for investigating the relationship between project constraints and appropriate iteration length. The base parameters are fixed as follows

Development term: 60 days (assume three months)
Uncertainty: 10
Complexity: 25
Developers: 0.25, 1, 1, 1, 2.5 (five developers)
Number of requirements: 30 (suitable size for development in three months by

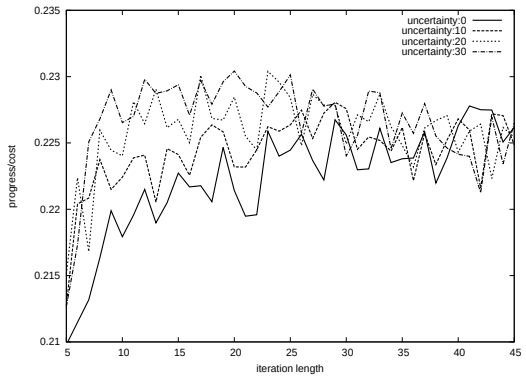

Figure 12 Simulation results with varied uncertainty (Raw value)

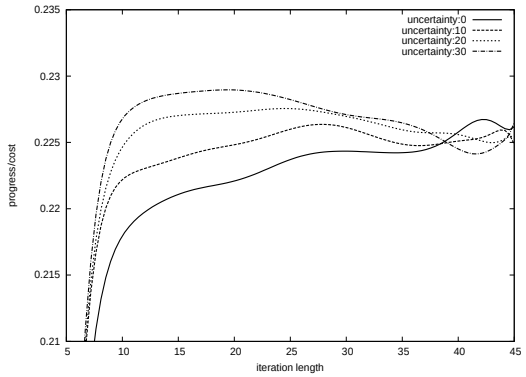

Figure 13 Simulation results with varied uncertainty (Approximated value)

five developers)

Figures 12-15 show the simulation results. For all of the figures, the x-axis describes the length of iterations and the y-axis shows the value of progress divided by cost.

First, we focus on the uncertainty. Figures 12 and 13 show the simulation result by using the fixed base parameters with varied uncertainty. Specifically, Figure 12 shows the raw values of the result, while Figure 13 shows the approximated values of the result to clarify the tendency. Each line in the graphs represents a different value of uncertainty: 0, 10, 20, or 30.

According to the graphs above, a high uncertainty tends to decrease the appropriate iteration length and a low uncertainty tends to increase it. This may be because frequent specification changes call for quick responses to them in order to reduce the integration cost and allow the preferred requirements to be implemented first. On the other hand, when the uncertainty is low, the overhead of iteration is affected if the iteration length is too short.

Next, we focus on the complexity. Figures 14 and 15 show the simulation result by using the fixed base parameters with varied complexity. As above, Figure 14

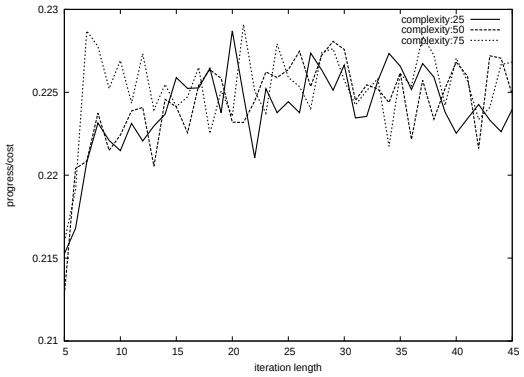

Figure 14 Simulation results with varied complexity (Raw value)

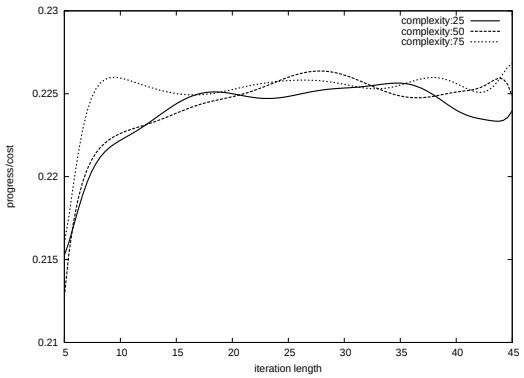

Figure 15 Simulation result with varied complexity (Approximated value)

shows the raw values of the result, while Figure 15 shows the approximated values to elucidate the tendency. Each line in the graphs represents a different value of complexity: 25, 50, or 75. It can be seen that a high complexity increases the appropriate iteration length. In the case where there are a large number of dependencies among requirements, a short iteration length reduces the development scope regardless of those dependencies and increases integration cost. Additionally, the results fluctuate more with increased complexity. This is due to the method used to generate the dependencies, which sets the dependencies randomly using the complexity values; this may have reduced the accuracy of the results.

Overall, the dispersion of the data appears to be high. This may be because of the way we treat the surplus days at the end of the project. In our simulator, surplus days are treated as follows. If the surplus days are less than half of the iteration length, surplus days are included in last iteration: e.g. in the case of 60 days duration with 11 days iteration length, surplus days are 5 days. Therefore, the last iteration becomes 16 days. If the surplus days are greater than or equal to half of the iteration length, surplus days are treated as the last iteration: e.g. in the case of 60days duration with 9 days iteration, surplus days are 6 days. Therefore, the surplus days become the last iteration.

Additionally, we notice that the five day iteration length seems to not be effective in any of the cases, although it is advocated by extreme programming. We believe that this is caused by our requirements volume adjustment. We choose a manageable volume of requirements as a fixed base parameter to simulate general, typical situations. Therefore, almost all the requirements are implemented in all cases. In such cases, five-day iterations seem to have too much overhead cost to be effective. In contrast, it is said that five-day iterations fit with normal work week, with a weekend to recharge between iterations. We haven't considered this sort of human factors so far, and have determined that it should be adopted as part of our future work.

4.4 Threats to validity

As we described above, we only use five parameters to characterize a project. This is because the simulator model is only for estimating the appropriate iteration length, and we eliminate unrelated parameters to avoid unnecessary complexity. However in a real project, no parameters can be ignored to determine its results. Additionally, our model is based on the common model of agile development that we extract from extreme programming and Scrum which could be a threat to internal validity.

Regarding threats to external validity, we validate the model with two case studies. Although the validity of the model is verified in both cases, its validity is limited to the range of the cases studied. In the future, we will consider the generalizability of our model by applying it to more cases.

5 Related works

Several software process simulation methods have been previously proposed to estimate the impact of processes before the start of a project. Barghouti and

Rosenblum proposed methods for simulating and analyzing software maintenance processes (Barghouti and Rosenblum, 1994). Otero et al. used simulation to optimize resource allocation and the training time required for engineers and other personnel (Otero et al., 2009). Rus. et al. helped apply the plan-based development using a system dynamics simulation model (Rus. et al., 1999).

The studies above are focused on plan-based development or a part of the development process, but there has been previous research on the agile development process as well. Port and Olkov created an original simulation model and used it to investigate requirement prioritization strategies (Port and Olkov, 2008). Moreover, they proposed a new prioritization methodology which combined the strategies of agile development and those of waterfall development. Melis et al. proposed an event-driven simulator for Extreme Programming practices, such as test-driven programming and pair programming (Melis et al., 2006). Anderson et al. presented an event-driven simulator of the Kanban process, and used it to study the dynamics of the process and to optimize its parameters (Anderson et al., 2011). They also used the simulation to evaluate the Scrum and Kanban approaches on the basis of actual software maintenance processes. Concas et al. assessed an agile project through empirical research using quantitative object-oriented metrics (Concas et al., 2008). They separated a project into five phases and evaluated each of them to investigate the applicability of practice. Garcia-Magarino et al. defined the processes of Scrum for multi-agent development using an agent process model (Garcia-Magarino et al, 2009).

There are two points in our work that are different from the related works. One is that our model is able to assess an actual project in whole. The other is that the purpose of our work is not to investigate the applicability of the practice, but to improve the effectiveness of the practice. Moreover, according to the classification of existing studies of agile development (Dyba, T., and Dingsoyr, T., 2008), studies tend to focus on extreme programming and the number and quality of studies on general agile software development are sorely lacking. In addition, most of the studies investigate the introduction of agile processes or compare the performance of agile development with that of traditional plan-based development. Therefore, it could be said the focus of our research complements this situation.

6 Conclusion and future works

In this paper, we simulated agile development while focusing on the iteration length to help apply agile development effectively to actual projects. There are three contributions in this paper: creating a model for estimating the appropriate iteration length, providing a way of simulating a particular project, and investigating the relationship between project constraints and appropriate iteration length. The results of our research show that the appropriate iteration length depends on the condition of project constraints; an increase in uncertainty reduces the appropriate iteration length, and longer iteration lengths are suited for high complexity projects. However, our model has a few threats to validity and needs to be validated more carefully with additional case studies.

As future work, we will make an effort to improve our model through the following.

- Consider the development team as a factor. This means not treating the developers as individuals but as a team, and focusing on their communication and interactions as they are essential for software development.
- Adopt the human effort as a factor to investigate the effectiveness of iteration lengths studied for typical business hours.
- Develop easier ways to input parameters, that utilize strict conventions, enabling users to make intuitive decisions. We are considering a questionnaire style method since it is known to represent the user's understanding of ideas in a unified process.

References and Notes

Independent administrative corporation, information processing promote organization, Software Engineering Center. (2009) Observation about non-waterfall development observation report, information property 0507

Dan Port, Alexy Olkov. (2008) Using Simulation to Investigate Requirements Prioritization Strategies, Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering

Masaru Amano. How to proceeding the agile development, http://www.slideshare.net/esmsec/ss-5656398

Akira Hattori., Koichiro Ochimizu.(2006) Validating the organization pattern using probability petri-net, Computer software vol. 23 No.1

Craig Larman(2003) "Agile and Iterative Development", Addison-Wesley Professional, 1 edition

Toshiya Ikegami. (2010) Why is agile failed, Nikkei System, http://itpro.nikkeibp.co.jp/article/COLUMN/20101215/355245/

Shusuke Shitara. (2009) Agile Transparency, http://gihyo.jp/dev/serial/01/agile-transparency/0002, (Accessed 10 November 2009)

Pekka Abrahamsson. et al, (2006) Extreme Programming and Agile Processes in Software Engineering: 7th International Conference, XP 2006, Oulu, Finland, June 17-22, 2006, Proceedings , Springer (Accessed 26 July 2006)

Ken Schwabe. and Mike Beedle. (2001) Agile Software Development with Scrum, Prentice Hall (Accessed 21 October 2001)

Boehm, B. and Papaccio, P. (1988) Understanding and Controlling Software Costs., IEEE Transactions on Software Engineering

Jones, C. (2000) Software Assessments, Benchmarks, and Best Practices, Addison-Wesley.

Tetsuo Tamai. (2004) Software Engineering, Iwanami shoten

Joana Rus. et al (1999) Software process simulation for reliability management, Journal of Systems and Software Volume 46 Issues 2-3, 15 April 1999, pages 173-182

H. Sackman, W. J. Erikson, E. E. Grant. (1968) "Exploratory experimental studies comparing online and offline programming performance", Communications of the ACM Volume 11 , Issue 1 1968

Tom DeMarco, Timothy Lister (2001) "Peopleware", Nikkei-BP

Takako Nakatani. et al, (2010) A case study of requirements elicitation process with changes, IEICE Transactions 93-D: 2182-2189, 2010

IT development plan of Shimane. (2010) Validating the business model of Ruby, http://www.pref.shimane.lg.jp/sangyo/it/

Eiwa System Management, inc. XP practice report http://objectclub.jp/community/XP-jp/xprelate/xppracticereport

Barghouti, N. S., Rosenblum, D. S. (1994) A Case Study in Modeling a Human-Intensive, Corporate Software Process. Proc. 3rd Int. Conf. On the Software Process(ICSP-3). 1994, IEEE CS Press.

Melis M. Turnu I., Cau A. and Concas G. (2006) Evaluating the Impact of Test-First Programming and Pair programming through Software Process Simulation. Software Process Improvement and Practice, vol. 11, 2006, pp. 345-360.

Melis M., Turnu I., Cau A. and Concas G. (2006) Modeling and simulation of open source development using an agile practice. Journal of Systems Architecture, vol. 52, 2006, pp. 610-618.

Otero, L.D., Centeno, G., Ruiz-Torres, A.J., Otero, C.E. (2009) A systematic approach for resource allocation in software projects. Comput. Ind. Eng. 56(4) 1333-1339.

Anderson, D.J.. Concas, G.. Lunesu, M.I., and Marchesi, M.,.(2011) Studying Lean-Kanban Approach Using Software Process Simulation. Proc. Agile Processes in Software Engineering and Extreme Programming 12th international Conference, XP 2011, Madrid, Spain, May 10-13 2011.

Dyba, T., Dingsoyr, T. (2008) Empirical studies of agile software development: A systematic review. Information and software technology, 50(9), 833-859.

Concas, G., Francesco, M., Marchesi, M., Quaresima, R., and Pinna, S. (2008). An agile development process and its assessment using quantitative object-oriented metrics. Agile Processes in Software Engineering and Extreme Programming, 83-93.

Garcia-Magarino, I., Gomez-Rodriguez, A., Gomez-Sanz, J., and Gonzalez-Moreno, J. (2009) Ingenias-SCRUM development process for multi-agent development. In International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008) (pp. 108-117). Springer Berlin/Heidelberg.