

Innovation in Software Development Process by Introducing Toyota Production System

● Koichi Furugaki ● Tooru Takagi ● Akinori Sakata
● Daisuke Okayama

(Manuscript received June 1, 2006)

Fujitsu Software Technologies (formerly Fujitsu Prime Software Technologies [PST]) has been conducting activities since 2003 to improve productivity using the Toyota Production System (TPS). An agile development process and a store management method were introduced to implement the basic concepts of TPS in the IT software field. We included the basic concepts of TPS (elimination of *muda* [waste], *heijunka* [leveled production], and *jidoka* [automatic detection of abnormal conditions]) and visual management in the agile development process and store management method as practical techniques. PST introduced this agile development process to its software development process and the store management method to support its maintenance process. As a result, PST achieved significant improvements in both processes and in its organizational climate. This paper introduces the TPS concepts employed in the agile development process, describes how *heijunka* is used in the store management method, and examines the effects of implementing agile development and store management at PST.

1. Introduction

Fujitsu Software Technologies (formerly Fujitsu Prime Software Technologies [PST]), following the burst of the IT bubble in 2000, introduced the Unified Modeling Language (UML) and intensified its project management to improve the productivity of software development. These efforts improved productivity, but not enough to overcome IT deflation. To break through this situation, the Toyota Production System (TPS),¹⁾ which has already proven its effectiveness in hardware manufacturing, was introduced experimentally to the software development processes. This means that the concepts of TPS such as the elimination of *muda* (waste), *heijunka* (leveled production), *jidoka* (automatic detection of abnormal conditions), and visual management are practiced in the software development processes. To implement these concepts, the agile development process (hereafter called agile development)

and the store management method, which employ TPS, have both been introduced. This paper describes agile development and store management and the effects of their introduction.

2. Agile development and process improvement

In general, agile development is regarded as the extreme opposite of waterfall development. In waterfall development, each process (e.g., planning, analysis, design, implementation, and testing) is performed only once and in sequence. In agile development, a series of these processes is repeated in what is known as repetition development (**Figure 1**).

Compared to the repetition style of agile development, waterfall development has the following drawbacks.

- 1) Waterfall development assumes that no changes will be made halfway through

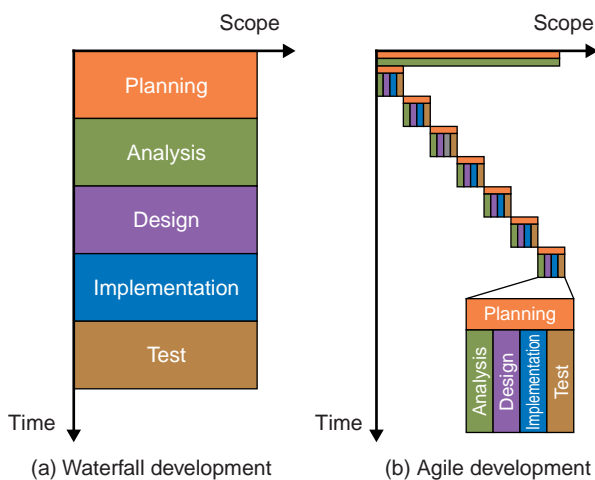


Figure 1
Comparison of waterfall and agile development.

the planning; therefore, the additional person-hours required for track back will be significant if any changes are made.

- 2) Bugs are detected at the review of each process end and during the test of lower processes. If any bugs are detected, the additional person-hours required for track back will be substantial.
- 3) If the quality of design and implementation are not good, the person-hours needed for testing (debugging) will increase.
- 4) Operable software cannot be obtained until the final process is completed.

Agile development has the following advantages to offset the issues inherent in waterfall development.

- 1) Analysis, design, implementation, and testing are repeatedly performed in smaller units; therefore, it is more tolerant of changes and additions to the planning (requirements).
- 2) To complete the required implementation and testing requirements, operable software can always be obtained, even if the scope of the implemented features is small.
- 3) By repeating the regression tests every time a small unit is implemented, failures can be

detected at an early stage.

Unlike other industrial products, a software product is not continuously manufactured. Also, in waterfall development, the series of processes is performed only once during the development period. Therefore, in many cases, lessons that were learned from the successes and failures that occurred during development will not be applied to the next project. Consequently, it is hard to execute the plan-do-check-act (PDCA) cycle.

On the other hand, with agile development, a series of software elements are developed repeatedly by almost the same people using the same material and in the same environment. Therefore, the lessons learned from the development processes will likely be used in the following development processes.

As a result, there are opportunities for the PDCA cycle to be executed. This is the true advantage of agile development.

3. Agile development and concept of TPS

Agile development includes many of the TPS concepts. The following are some examples.

- 1) Pull system

The concept of the pull system is embodied in the acceptance test of agile development. The acceptance test tests whether the software meets the customer's requirements. This test drives the implementation, which is very different from the situation in traditional development, in which the incoming specifications from the upper processes drive the implementation.

- 2) Just-In-Time

The Just-In-Time concept prioritizes the customer's needs and implements the prioritized functions sequentially. Also, this is described as You Aren't Going to Need It (YAGNI).

- 3) Visual management

In agile development, "mirroring," which is a practice of Extreme Programming (XP), embodies the visual management technique, which makes a project's status self-explanatory using analog

media and encourages much faster feedback in the action level.

4) *Heijunka*/multi-skill development

Multi-skill development, which gives each worker multiple work skills, and *heijunka*, which reduces each worker's working hours, are the principles of agile development.

4. Agile development

In agile development, the development period is divided into units called iterations. An iteration is a period in which processes are executed to develop the scope of a function.

The first iteration starts at the beginning of development. At this point, the prioritized customer requirements are clearly indicated to the development team, which implements them according to their priority. Also, the end date is decided when the iteration starts. The time period is typically from one week to a month. Even if the requirements have not all been implemented, the end date is given priority. Any requirements that have not been implemented (lower priority) are postponed. If all the requirements have been implemented before the end date, new customer requirements can be worked on. Therefore, although all of the functions might not be implemented as originally estimated, the provision of outcome will not be delayed. Therefore, the delivery date is given priority over the function scope. Once the iteration is terminated, even if all of the customer requirements have not been implemented, the high-priority functions will have been implemented, and operable software can be output.

As soon as the first iteration is terminated, the second iteration begins. At this time, as with the initial iteration, the prioritized customer requirements and the operable software from the initial iteration will be entered. This iteration is often set to the same length as the initial iteration. Repeating the same time period is an effective way for the team to learn the development rhythm within the time frame. When the

second iteration is terminated, the operable software is output in the same way as in the first iteration. This software has the functions that were implemented during the second iteration as well as the initial function scope.

As each iteration is executed, the software is developed and the implemented scope is extended.

Also, the software can be released to the customer whenever an iteration is complete.

Figure 2 shows the development procedure within an iteration. The customer requirements entered when the iteration starts are sorted in a simple, single function called a story. It is preferable if the customer creates a story on their own; however, in many cases, a role called the *okyakusama* (customer) proxy is set within the development team. The customer proxy consists of one or more team members who represent the customer's interests. The members who best understand the customer's position undertake this role, and the requirements of stories always have top priority.

The created stories are divided into units called tasks at a staff meeting called a planning meeting (or planning game). A story is a function unit from the customer's viewpoint, and a task is an implementation unit from the developer's viewpoint. In other words, the planning meeting can be described as the design task. This planning meeting is held when the iteration starts.

Tasks that have been divided at the planning meeting are assigned to the developers at the stand-up meetings (brief all-staff meetings) that are held every morning. The tasks are often assigned according to the developers' own choices rather than as instructed by the manager or supervisor. The developers are required to understand the entire development picture and determine what should be done and take action on their own initiative.

Also, when the pair programming phase is executed, the pairs will be decided at the stand-up meeting on that day. Ideally, the pairs are

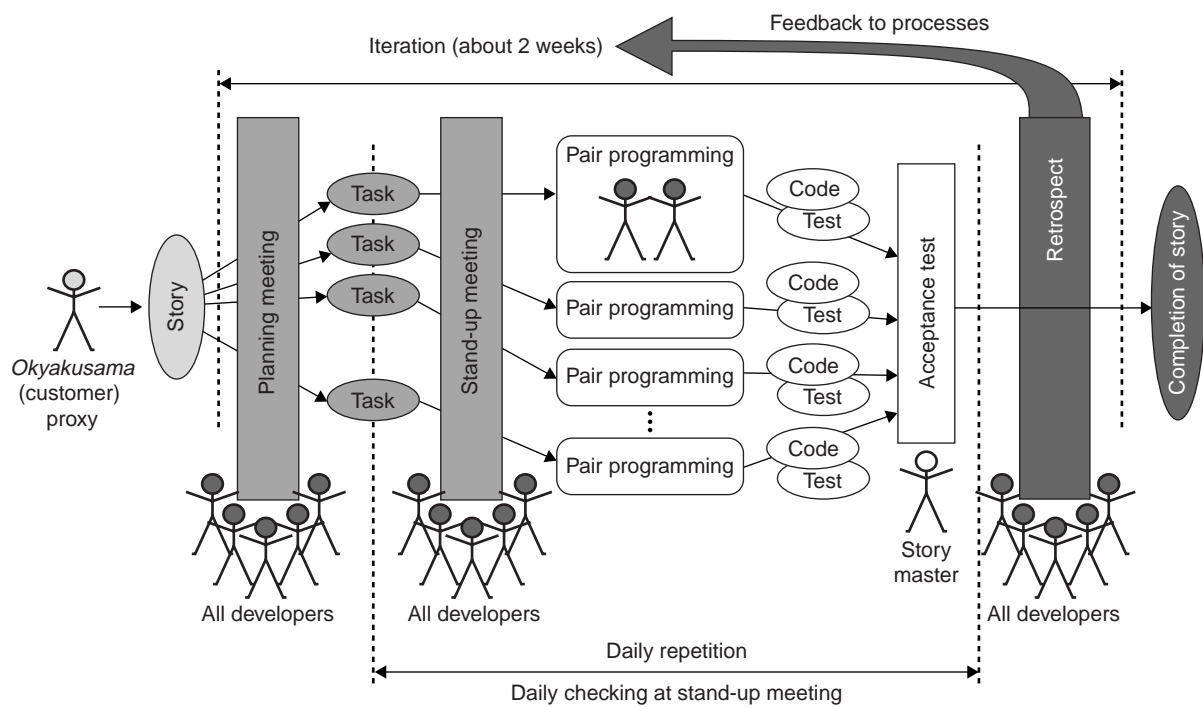


Figure 2
Agile development procedure.

switched every day. Pair switching encourages the spread of knowledge and therefore can provide a higher learning effect within the development team.

Also, by making the program code and components available to the whole team instead of allocating it to specific persons, “unbalanced intelligence” about the software development can be prevented.

The test program is created at the same time as the code is developed by the pairs.

The developed program is tested by this test program, and the tasks are completed. When all of the tasks that compose a story are completed, the completion of the story is tested by the acceptance test, which is a test program created by the customer or customer proxy.

From another viewpoint, this is a test procedure in which the criteria for completion are clear. The story is completed by passing the acceptance test. Within an iteration, procedures from the implementation of the tasks to the

completion of the story by the acceptance test are repeated daily.

At the end of an iteration, a meeting called the *kaiko* (retrospect) is held, in which all the developers participate.

In the *kaiko*, all of the details relevant to the iteration are reviewed, for example, the coding, testing, and meetings and even the air ventilation, lighting, and telephone manners of team members are reviewed.

Then, the staff do the following.

- 1) Review the activities that will continue in the next iteration
- 2) Discuss any problems that occurred, find their solutions, and confirm their implementation in the next iteration
- 3) Reach an agreement about new efforts that must be made in the next iteration.

Bringing the resulting feedback from this *kaiko* meeting into the development process drives future improvement.

5. Implementation of agile development

The following is an example implementation²⁾ of agile development.

In this example, a prototype system was developed for the manufacturing industry with a development period of about two months. The system is an infrastructure Web application that uses server-side Java (servlets). There were nine members in the development team.

The following lists the amount of software-development experience and object-oriented development experience (in that order) in years for each member.

Member X: Tracker and customer proxy: 9 and 8

Member Y: Manager and coach: 5 and 3

Member Z: Coach: 3 and 2

Member A: Developer: 3 and 1

Member B: Developer: 3 and 1

Member C: Developer: 0 and 0 (new employee)

Member D: Developer: 2 and 1

Member E: Developer: 4 and 1

Member F: Developer: 14 and 7

Members E and F joined the project at the end of July. Also, none of the staff had previous experience of agile development. The introduced practices of XP were as follows.

1) Iteration (iterative development)

At the beginning, a five-day iteration was used as a trial and then a two-week iteration was set three times.

2) *Kaiko* (retrospect)

3) Visual management (mirroring)

Story cards, task cards, and a burndown chart (graph of number of backlogs) were posted on a wall so the developers could check the progress in real-time.

4) Pair programming

5) Co-ownership of source code

6) Coding criteria

7) Stand-up meetings

8) Continuous consolidation

9) Unit testing

10) Acceptance testing

11) Customer proxy

12) Refactoring

Test-driven developments were not performed because, since this was the first agile development for the staff members, certain technical risks had to be avoided.

6. Evaluation of agile development

The evaluation of the agile development conducted in the example implementation described in the previous section is detailed below.

The viewpoints of this evaluation are productivity, quality, cost, delivery date, and the opinions from developers.

6.1 Productivity

The productivity in each iteration and the productivity in the entire development are indicated below using a productivity index.^{note)}

Iteration 0: 0.760

Iteration 1: 0.949

Iteration 2: 0.911

Iteration 3: 1.962

Average throughout: 1.268

The productivity at the beginning of the development was 24% lower than PST's standard productivity; however, it was improved at the last stage and indicated a 26.8% increase throughout the agile development. The reasons for this productivity improvement are examined below.

1) The functions were implemented according to their priority so that unnecessary functions were not implemented. Also, stories were divided into tasks and then developed one by one (single-flow production) so that waste (i.e., untested programs) was eliminated. In addition, because of the existence of the customer proxy, no intermediate output such as unrequested documents was created.

note) A comparison of the number of source code lines written by each member per month against a standard PST value

2) Because of the pair programming, the necessary information was always available for the developers. Also, by co-owning the source code, any developer could modify any program. This prevented lags due to modifications by other developers, which tends to occur during traditional development.

3) The daily progress was checked in the stand-up meeting every morning, which enabled the team to provide a timely response to latency and optimize the task schedule.

4) The *kaiko* meeting held for each iteration implemented the PDCA cycle within the development process. This meeting enabled the development team to share their problems and give feedback for the next iteration.

The above practices are considered to improve productivity.

6.2 Quality

Agile development is not like waterfall development, which is divided into processes such as planning, analysis, designing, implementation, and testing, and therefore the quality cannot be verified by tracking down the occurrence of bugs at each process. However, developers are finding that agile development promotes high quality.

1) With the pair programming, developers can

ask questions or discuss their concerns at any time. Also, the program can be reviewed at any time, which dramatically suppresses the creation of bugs compared to the traditional method.

2) The automated regression tests reveal bugs within a day. Also, the program code is immediately tested with the test program that is created at the same time and then debugged.

6.3 Cost

Figure 3 shows the overtime hours of the developers in each iteration.

As the iteration precedes, the difference in overtime hours between developers decreases, which means *heijunka* is taking place.

The *heijunka* of the working hours is affected by the degree of multi-skill development of the developers. If any developer can work on any job, the work hours can also be leveled. **Figure 4** shows the commitments of each developer to the programs as measured using the commitment log of the Concurrent Versions System (CVS), which is a configuration management tool. As can be seen, each developer had multiple commitments, which indicates that multi-skill development was occurring.

In the traditional method, each developer is in charge of a component or a program, which,

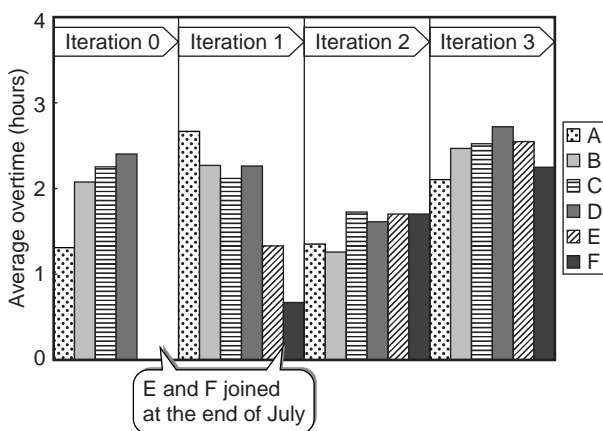


Figure 3 Developer's overtime hours in each iteration.

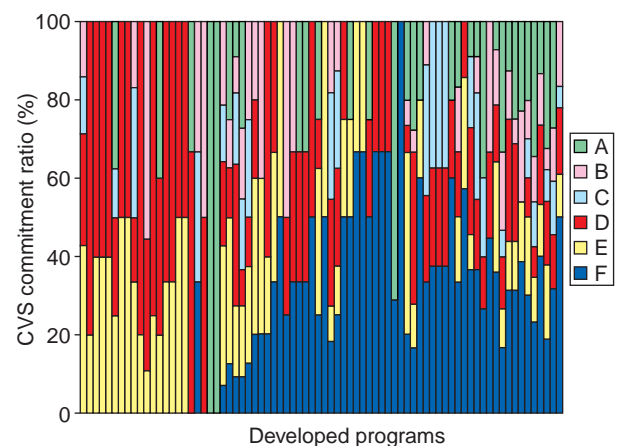


Figure 4 CVS commitment ratios of developers.

consequently, has the idiosyncrasies of the developer.

Co-ownership of the source code and pair programming help implement multi-skill development. No one is appointed to work on a specific program: each day, the developers work on a different program and with a different partner so that the experienced developers pass on their knowledge. This mechanism facilitates multi-skill development.

In this agile development, compared to the previous waterfall development of a similar system, the cost rate was reduced by 16% (cost reduction).

6.4 Delivery date

Figure 5 shows the transition of the number of customer requirements and implemented requirements in this agile development.

Fifty percent of the customer requirements

were clear since the beginning of the development, and the remainder were received sporadically during the development. Also, many requirements disappeared from the list. New requirements from the customer were implemented in the order they were received, and all the requirements were implemented by the release date. The average time between the arrival of a requirement and its implementation was one week. In fact, on average it took as little as one week to implement a requirement after it was received.

This rapid rate of implementation was achieved due to the practices of customer proxy, iteration, and stories. The requirements were prioritized, the implementation of single-flow production shortened the delivery date, and the regression tests were performed in an environment of continuous integration. Including refactoring, we think we have formed a development process that ensures quality and also allows additional

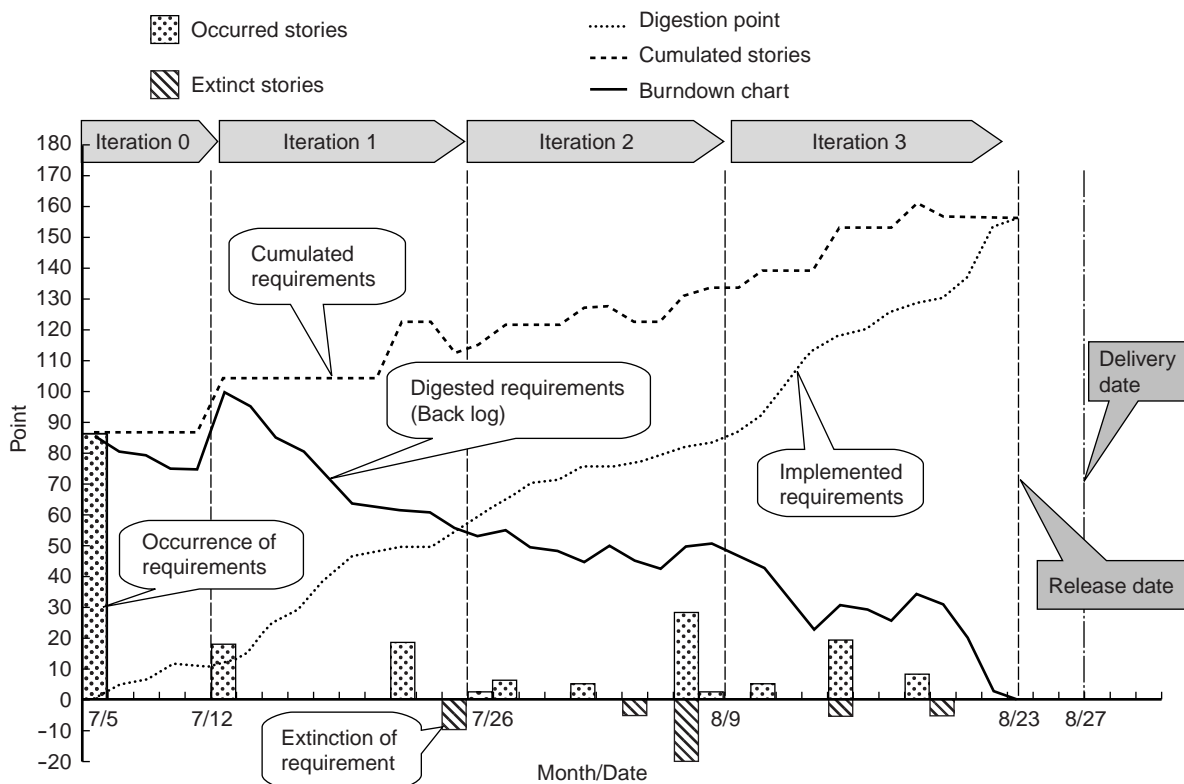


Figure 5 Transition of customer requirements and implemented requirements.

requirements to be implemented.

6.5 Opinions from the development members

The following are some of the opinions of the development members in the agile development project.

- 1) "In the past, when I became overloaded, I was under pressure and did not want to come to work. However, this time, even if I was overloaded, I did not feel under pressure."
- 2) "It was easy to see the quantity of work per day at the task allocations in the stand-up meetings."
- 3) "It was good to have a coach. We could ask the coach questions at any time."
- 4) "The atmosphere of the entire team has been good, and the members' motivation remained high until the end of the development."
- 5) "The pair programming system prevented the work from becoming stuck."
- 6) "During the pair programming, we worked while talking about things we knew or did not know. So eventually we learned new things."
- 7) "Due to the pair programming, the skills of the team quickly improved."
- 8) "The pair programming reduced mistakes and facilitated efficient work."

These quotes show that the developers kept their motivation high and were able to work with a stable mental state during the entire development period.

7. Expansion of agile development

This example of agile development described in the previous section was a relatively small development for a prototype system. It was performed on an XP basis and was suitable for what is generally considered the proper range of XP (i.e., for software that is not life-critical and with a team of less than 20 people).

There are many issues to be solved, for

example, the implementation method and the verification of quality, before agile development can be applied to large-scale projects involving more than about 20 people and the development of mission-critical systems. However, there seems to be many procedures in agile development that can also be effective in waterfall development, for example, visual management; the iteration/retrospection meeting (*kaiko*), which implements the PDCA cycle within the development process; and the use of pair programming, which levels the loads and abilities of the developers. In the future, the implementation of agile development practices will be expanded step-by-step into traditional development processes such as waterfall development.

8. Process improvement by store management method

Agile development is intended for software development processes; however, the store management method is intended for much larger processes. The store management method improves a process by implementing the TPS concept of leveled production. In this section, we describe this method using the work model shown in **Figure 6** as an example.

This work model has three types of operation works that require randomly generated work hours and abilities (a, b, and c) and a person in charge of each operation to process the work sequentially.

Generally speaking, regarding the work generated in the operation processes, workers are assigned to each operation and perform the work. However, when work is done in this style, the work capability and the workload depend on the operations, which reduces the overall performance. The store management method improves the overall performance by gathering the instruction management for the work and minimizing/*heijunka* the workload differences and work capabilities between workers. **Figure 7** shows

the *heijunka* improvement that was achieved in this example.

The following *heijunka* practices were implemented.

- 1) Work execution using the store pull system
- 2) Single-queue workload management
- 3) Visual management of workload and auto-

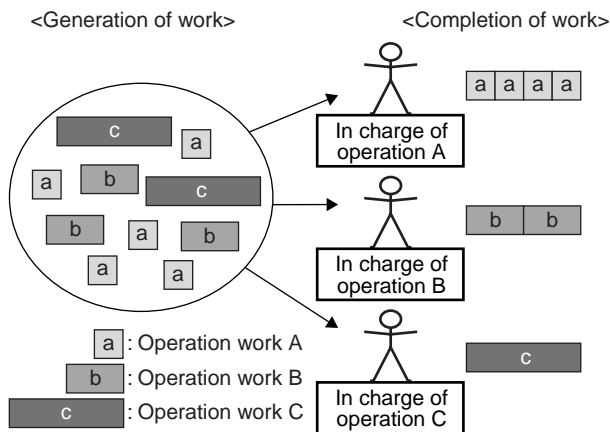


Figure 6
Work model for store management.

nommic *heijunka* control

These practices are described below.

- 1) Work execution using the store pull system

As shown in **Figure 8**, the work generated in the operations were managed in work instruction boxes called Store-IN boxes. The workers chose one work item and then started working on it.

It is important that the workers focus on one work item at a time (single-flow production). Otherwise, they will waste time switching between work items and might, as a result, be unable to complete them within the allocated amount of time, leading to a reduction in overall lead-time.

- 2) Single-queueing workload management

As shown in **Figure 9**, when work items were generated, instead of assigning them in equal amounts to each worker, they were managed using a single queue of Store-IN boxes and processed in sequential order. When randomly generated work was processed, this method greatly

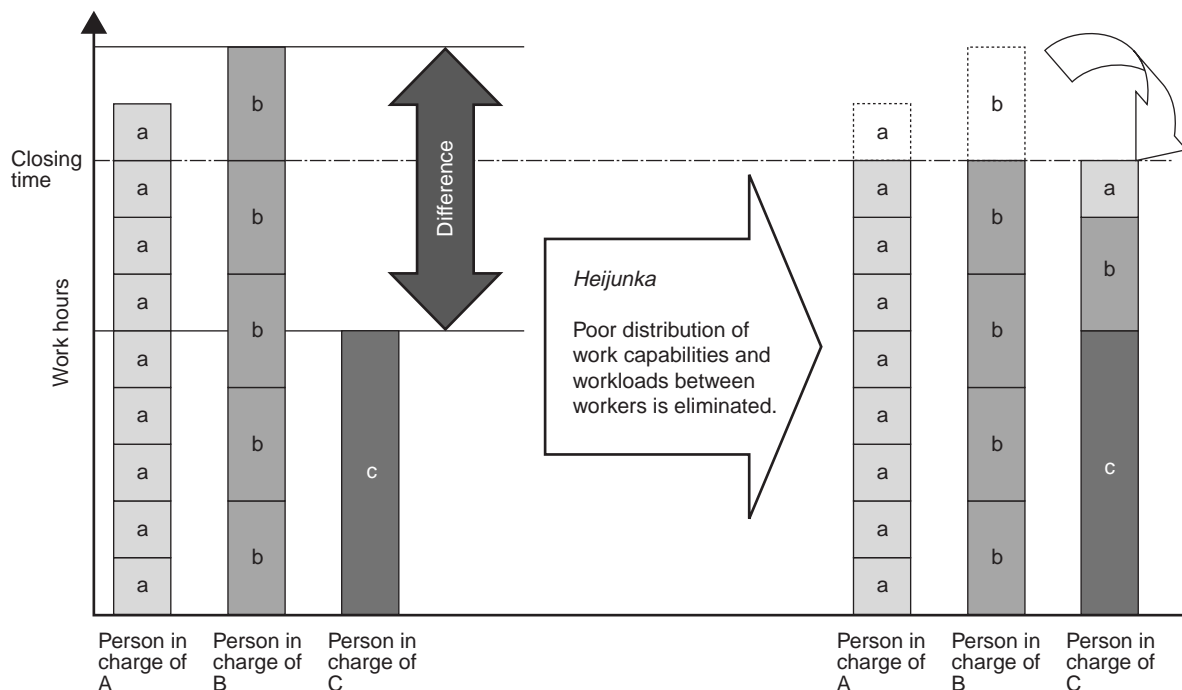


Figure 7
Heijunka improvement by store management.

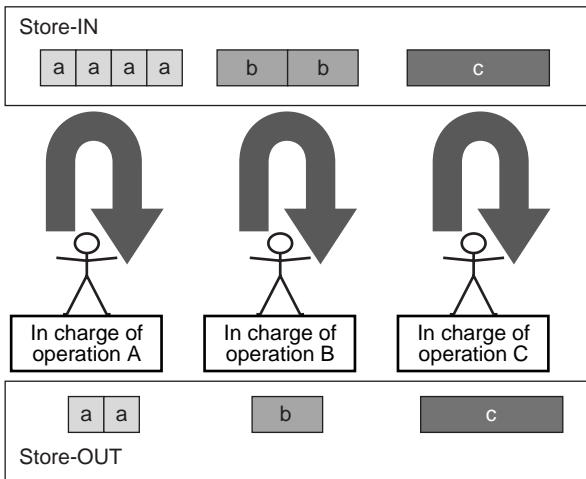


Figure 8
Work execution using store pull system.

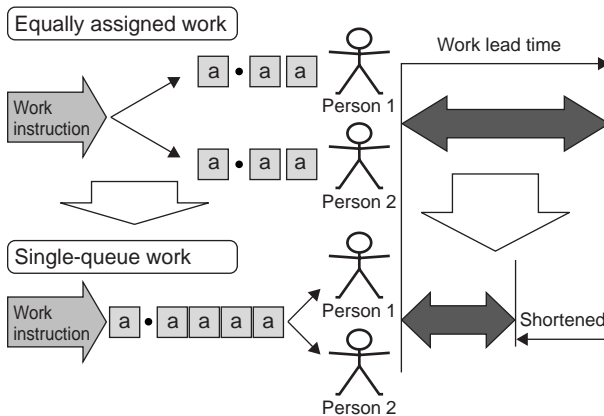


Figure 9
Shortened lead time by single-queue workload management.

reduced the work lead-time compared to the equally-assigned work method.

3) Visual management of workload and autonomous *heijunka* control

As shown in **Figure 10**, the Store-IN status was displayed in an easy-to-visualize way on boards and other media (visual management) so workers could see what work was pending for each operation. By checking the load status of the queued work on the boards, workers could determine whether any additional workers were required or whether certain workers should be switched to different workers. *Heijunka* was implemented by the workers acting independently

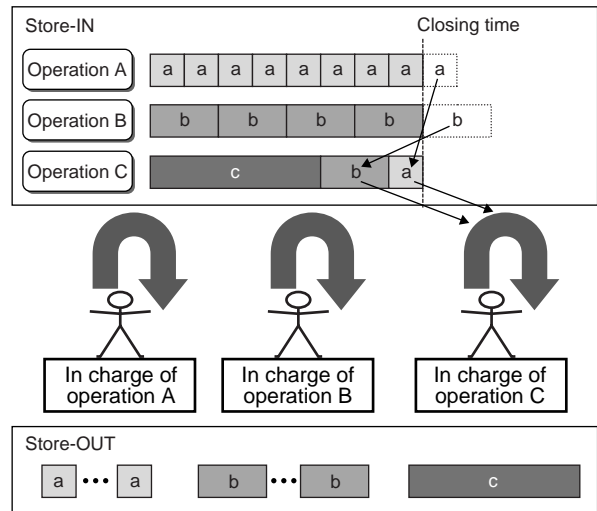


Figure 10
Visual management of workload and autonomous *heijunka* control.

to maintain the balance between the workload and work capability. By solving the issues generated in the course of implementing these practices one-by-one, the operation process was improved.

9. Effects of introducing store management method

PST's software maintenance support business is not suited to agile development. However, PST improved the operation of this business by introducing the store management method. The problems affecting the traditional operation of this business and the advantages of the store management method are described below.

The problems were as follows.

- 1) The workload tended to be concentrated at the end of the week and in the late afternoon. Therefore, workers frequently had to work overtime and on holidays.
- 2) Because of the workers' differing abilities, some workers had an especially high workload.

Implementing the store management method had the following effects.

- 1) Visual management of the workload status enabled the team members to always understand the distribution status of the workload and to level the workload independently. Since the store

management method was introduced, none of the current team of workers has had to work during a holiday.

2) The difference in overtime hours was reduced to about 30% by leveling the workload between workers.

3) The teamwork capability was enhanced by about 20%.

Before the store management method was introduced, there were no methods to measure the quantity of work in the software maintenance support operation; therefore, it was not possible to understand the work capability quantitatively. However, after the introduction of the store management method, all of the work has been recorded on work cards so the work capability can be understood and evaluated by analyzing and aggregating these work cards. This is another big advantage of the store management method.

Figure 11 shows the number of work cards that were written during a five-month period after the store management method was implemented in the operation management division of PST. The figure gives a quantitative indication of how the work capability improved.

By introducing the store management method, productivity was improved by about 80% in 4 months.

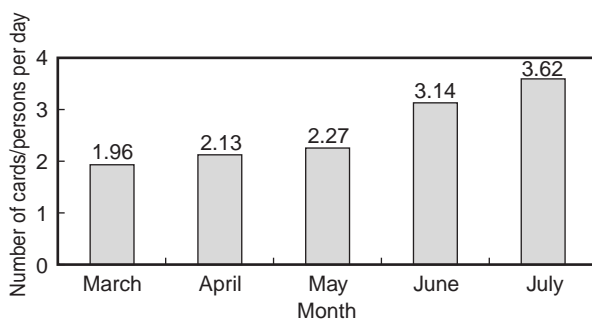


Figure 11
Quantification of improvement based on number of work cards.

10. Conclusion

When implementing the TPS concept using agile development and the store management method, we found it is useful to return to the fundamental principle of “manufacturing” in the software field and to improve our software development process.

The concept of TPS was introduced to some of the divisions experimentally in 2003, and by the end of 2004, it had been introduced to every PST division. PST’s profits stopped declining in September 2004, and the company has been on a gradual recovery since then.

Acknowledgement

We are very grateful to Mr. Junichi Matsui of Consultsourcing Co., Ltd. for his help in introducing the store management method to PST’s software development process.

References

- 1) T. Ohno: Toyota Production System: Beyond Large-scale Production. (in Japanese), Diamond, 1978.
- 2) A. Sakata: Agile Practice. Is agile management the Toyota Production System? — Attempt of Software Multi-Skill Development. (in Japanese), Developers Summit 2005 (presentation material), 2005.



Koichi Furugaki, Fujitsu Ltd.

Mr. Furugaki received the B.S. and M.S. degrees in Chemical Engineering from Tokyo Institute of Technology, Tokyo, Japan in 1974 and 1976, respectively. He joined Fujitsu Ltd., Kanagawa, Japan in 1976, where he has been engaged in development of mainframe operating systems.

E-mail: furugaki.kouich@jp.fujitsu.com



Tooru Takagi, Fujitsu Software Technologies Ltd.

Mr. Takagi received the B.S. degree in Engineering from Ritsumeikan University, Kyoto, Japan in 1985. He joined Fujitsu Aichi Engineering Ltd. (currently Fujitsu Software Technologies Ltd.), Aichi, Japan in 1985, where he has been engaged in research and development of OS subsystems. He was also engaged in development of

Web systems from 2000 to 2003 as an SE. He promotes software development based on the Toyota Production System.

E-mail: takagi.tooru@jp.fujitsu.com



Akinori Sakata, Fujitsu Software Technologies Ltd.

Mr. Sakata received the B.S. degree in Engineering from Yamanashi University, Yamanashi, Japan in 1989. He joined Fujitsu Aichi Engineering Ltd. (currently Fujitsu Software Technologies Ltd.), Aichi, Japan in 1989, where he has been engaged in research and development of mainframe operating systems. He was also engaged in

development of Web systems as a developer from 2000 to 2003. He promotes software development based on the Toyota Production System.

E-mail: sakata.akinori@jp.fujitsu.com



Daisuke Okayama, Fujitsu Software Technologies Ltd.

Mr. Okayama received the B.S. degree in Engineering from Yamanashi University, Yamanashi, Japan in 2002. He joined Fujitsu Prime Software Technology Ltd. (currently Fujitsu Software Technologies Ltd.), Aichi, Japan in 2002, where he was engaged in development of Web systems as developer from 2002 to 2003. He was then

engaged in development of a financial accounting system until 2004. He promotes software development based on the Toyota Production System.

E-mail: okayama.daisuke@jp.fujitsu.com