Technology Reports
Agile Remote Work App Development

Migrating an Agile Application Development System to an Efficient Remote Work Environment - A Case Study

Communication Device Development Department Takashi Fukuzono Eiko Onuki
Kiki Wakayama

Usually in agile development, members gather at the same location to work. However, to avoid the spread of the novel coronavirus, development systems that incorporate remote work are necessary. Therefore, to maintain efficiency in development, NTT DOCOMO began combining related tools such as those for Web task management and automation, and Web conferencing tools for remote work. This has made it possible to maintain product quality and release products with the same short cycles as before the coronavirus pandemic.

1. Introduction

The docomo TV terminal is a TV device sold by NTT DOCOMO. By connecting it to a TV, the user can watch various video services such as Hikari TV for docomo.

The docomo TV terminal app is a smartphone app for operating the docomo TV terminal, and has numerous functions such as remote-control operation of docomo TV terminal and viewing recorded content in other locations. Also, to support the constant upgrades to video services made in response to changes in the market environment, it is effective to develop the docomo TV terminal app using agile development, which prioritizes and develops required functions in short cycles.

Agile development requires an environment that enables close communications because of its short development cycle and often takes place in the same location. For the docomo TV terminal

app, our development team gathered at one location and carried out development in an environment that enabled such close communications.

However, due to an increase in the number of people infected with the new strain of coronavirus in the Tokyo metropolitan area, and in light of the risk of infection with the new strain of coronavirus within the agile development team, it was decided that, in principle, all development members would work remotely from February 2020, after development tools and communication methods were improved.

This article describes the case study of agile development of the docomo TV terminal app and the innovations involved in remote development.

2. Agile Development Overview

2.1 Framework for Agile Development

Agile development is a general term for software development methods that repeat development in short periods of time. It began when notable persons in agile development declared values and principles of behavior common to agile software development as the Manifesto for Agile Software Development [1].

Because agile development proceeds through repeated development in short periods of time, it is more robust to change than waterfall development*1 and can respond quickly and flexibly to changes in market user needs.

There are various frameworks for agile development, such as Scrum, Extreme Programming*2 and Kanban*3, etc. For our development, we adopted Scrum, a framework that is lightweight and easy to understand (but difficult to learn).

The theory and definitions of Scrum are described in the "Scrum Guide" [2]. Its content is revised as needed. Scrum defines three roles, five ceremonies, and three artifacts based on the three principles of "inspection" to make sure the team is progressing correctly, "adaptation" to improve the process as a result of the inspection, and "transparency" to make sure all the current status and problems are visible.

2.2 Overview of the Scrum Process

1) The Three Roles in Scrum (**Figure 1**)
   (1) Product Owner (PO): Responsible for maximizing the value of the product produced by the Scrum team. Responsible for the product, creates and prioritizes the product backlog, which describes the requirements for all product functions.
   (2) Scrum Master (SM): Responsible for promoting the understanding and practice of Scrum and ensuring that the process runs well. Ensures that the Scrum team is effective by leading them to become self-managed and cross-functional by removing obstacles to their progress and helping them to improve.
   (3) Development member: Development members realize the product. There is no hierarchy. The ability to create products is fulfilled when everyone aligns. The team makes every effort to complete the product backlog items agreed upon with the PO, and is responsible for constant improvement.
2) The Five Ceremonies in Scrum
   (1) Daily Scrum: A place where the status of the development team is examined daily with

*1 Waterfall development: A development method in which the processes of definition of requirements, design, implementation and evaluation are performed in order.
*2 Extreme Programming: A type of agile software development methodology that takes rules of thumb to the extreme for efficient development.
*3 Kanban: A type of agile software development method in which work items are visualized in a list called "Kanban" to continuously implement and improve the work.

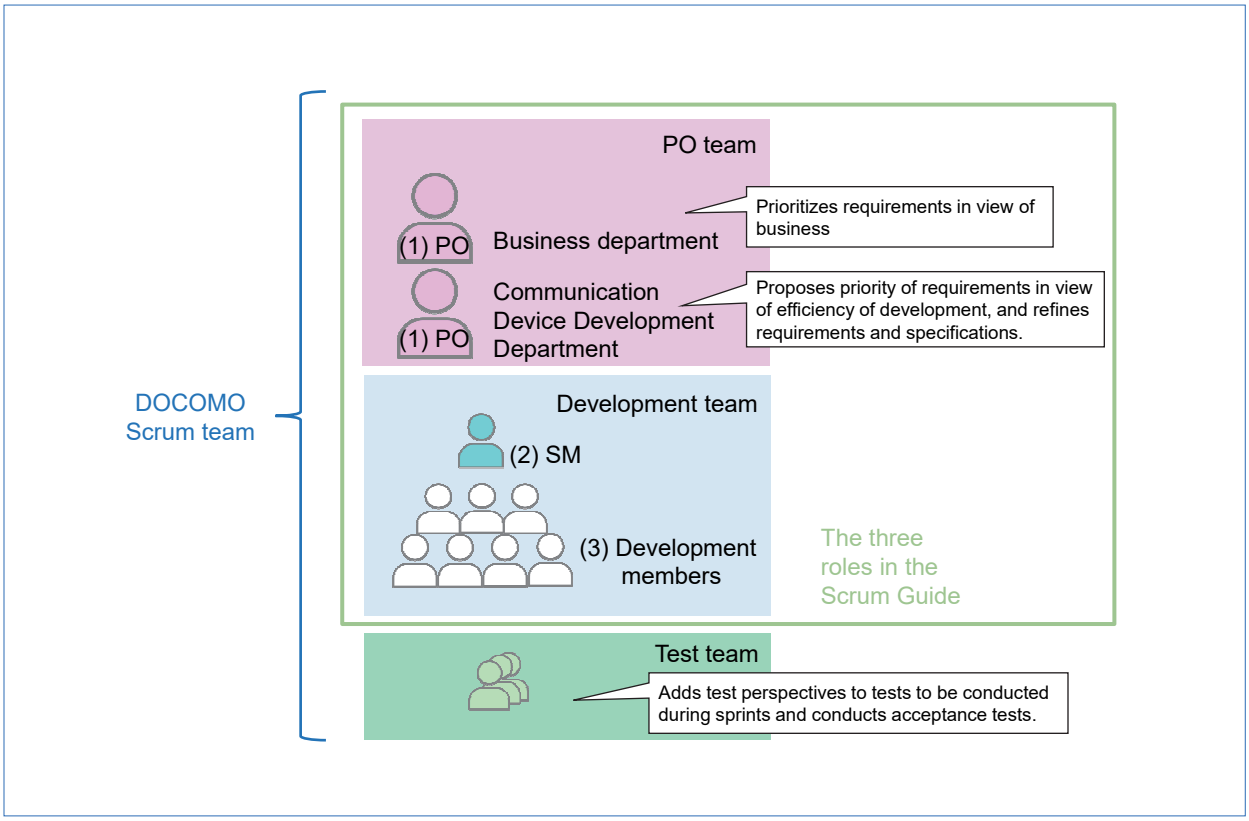

Figure 1 Organization chart

three questions (what was done yesterday, what needs to be done today, and are there any obstacles?). 15 minutes maximum, no extensions.

(2) Sprint*4 review: A place for the PO to review the development team's sprint artifacts (apps). These are presented to the relevant stakeholders. After that, the team may get feedback and revise the apps.

(3) Sprint planning: Also called a planning meeting, and where a development plan for a sprint is made. The PO talks about the what and why of the product backlog, and the development team talks about the how. The development team also talks about how likely they are to achieve the requirements of the product backlog in this sprint.

(4) Sprint retrospective: This ceremony is also called a "look back" and is used to repeat improvements so that the work can be done even better. The ceremony entails sorting out what went well in the sprint and what can be improved in the future. Rather than fixing bugs, fix the processes in which bugs are created. In our project, we use the KPT method*5 to share and agree with members on what to continue and improve in the following sprints.

(5) Backlog refinement*6: Maintenance of the product backlog for subsequent sprints.

*4 Sprint: A short development period, limited by the Scrum Guide to no more than one month.

*5 KPT method: A framework for reviewing the progress of a project, in which "Keep (good things)," "Problem (issues)," and "Try (improvement measures for problems)" are listed and reviewed for approaches and procedures during a sprint period.

*6 Backlog refinement: One of the events defined in Scrum. The process of cleaning up the product backlog for subsequent sprints.

3) The Three Artifacts in Scrum

(1) Product backlog: A description of the requirements for each function of the entire product, the so-called wish list. Prioritizes and arranges the features with the highest value so that they are developed in order from the top of the list. Priorities need to be constantly updated so that they are the latest.

(2) Sprint backlog: The division of the product backlog into specific tasks.

(3) Sprint artifact: Created by the development team on a sprint-by-sprint basis and is an artifact for which release decisions can be made. In app development, this means a working app and documentation.

2.3 Structure of Our Project

For our project, the business department and the Communication Device Development Department are acting as the PO team on Scrum development. The Communication Device Development Department is responsible for proposing priorities based on development efficiency, while the business department is responsible for final prioritization based on business. The SM and development team consist of members from the development contractor, and the test team consists of members from the test contractor (Fig. 1).

3. Scheme for Short-cycle Release

3.1 Requirements Definition Phase

1) Development Prioritization

The business department assigns business priorities to the development requirements, and the Communication Device Development Department and the development team arrange the development requirements in detail in order of highest priority. Requirements arranged in detail are fed back to the business department to determine development priorities.

(1) Business priority assignment

For each development requirement, the business department explains to the Communication Device Development Department and the development team the priority of the requirement and how it will benefit the business. By doing so, the entire team gains common recognition of the objective, and the Communication Device Development Department and the development team are able to arrange requirements into details and prioritize the development appropriately in subsequent processes.

(2) Arranging requirements in detail

The Communication Device Development Department and the development team refine the requirements in order of priority of development requirements assigned by the business department, and further divide functions. The Communication Device Development Department assigns development priorities to the functions arranged in detail judged on business effects, development scale and development efficiency. The divided detailed functions and development priorities are fed back to the business department, which makes the final decision on development priorities.

2) Integrated Ticket[*7] Management

Efforts are made to efficiently update and share

*7 Ticket: A task or issue that arises in a development project.

tickets with all members, from the PO team to the development team (**Figure 2**). After the PO team describes the requirements (what, why) as parent issues in the ticket management tool, the development team describes the details of the work (how) to realize these as sub issues. In addition, by discussing how to implement the changes on the source code review tool and linking it to the ticket management tool, it is now possible to understand and manage changes in the source code due to which requirements and which tasks in an integrated manner.

As a result, misunderstandings and omissions in requirements, tasks, and source code modifications are less likely to occur, which means reduced reworking.

(1) Requirements

In the requirements stage, the PO team not only describes the details in the ticket management tool in as much detail as possible, but also focuses on the background and purpose of why the development requirement was made. As a result, we created an environment in which each member of the development team can actively consider the design from a business perspective when the team moves into doing specific tasks.

(2) Work details

The parent issue is the requirement, and the development team describes the work required to implement the software based on the requirement as sub issues. Linking

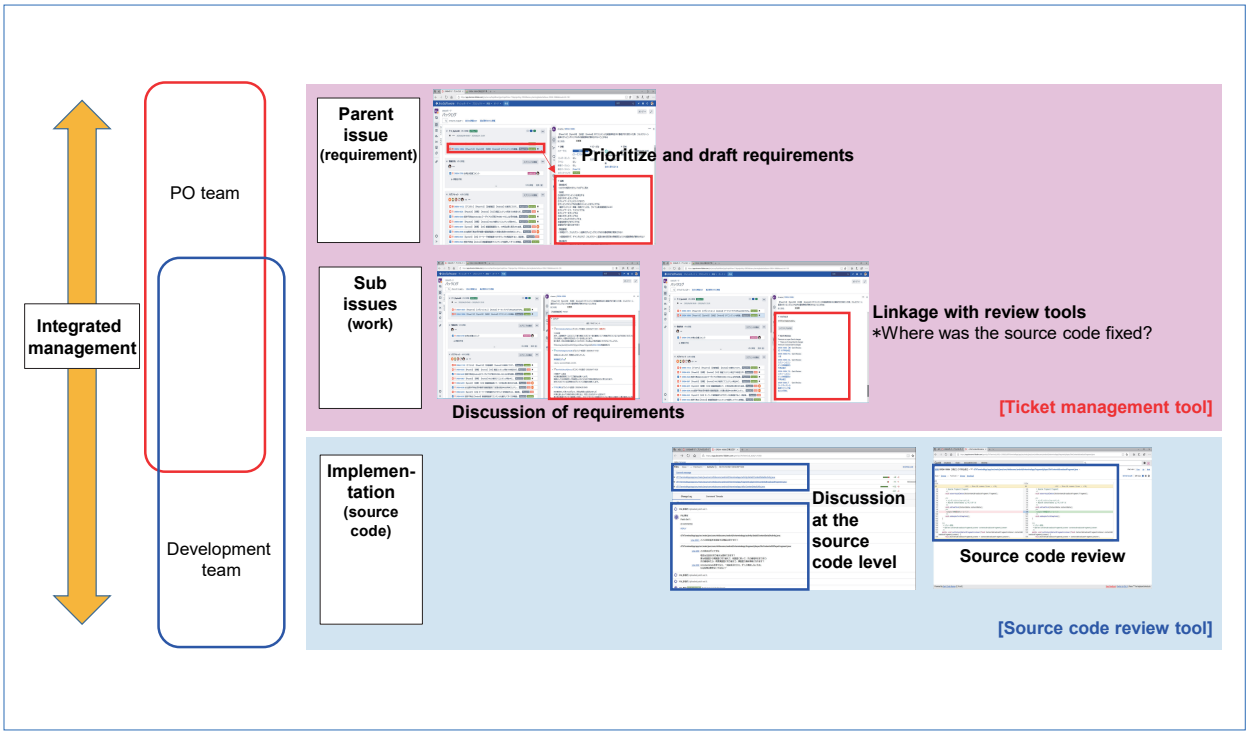

Figure 2 Integrated ticket management

requirements to work details ensures that no content to be implemented is omitted.

Also, linking work content with the source code review tool makes it possible to understand processes in later reviews by mapping modifications in the source code.

(3) Supplement with text chat tools

Tickets are not created for minor Q&A, etc., so instead, text chat tools are used to conduct Q&A. Compared to e-mail and ticketing tools, text chat tools do not require greetings and can be used to ask questions without strict classification of the content. This has the effect of reducing communication costs such as time spent on writing, and even when details have not yet been decided, the content is solidified as the chat continues, which is useful for aligning members' thinking before ticketing and for detailing ticket content.

3.2 Development Phase

For the development phase, we have built a system that does not distribute information such as documents and source code necessary for development to individual work terminals, but stores such information on the cloud so that members can share it. The system can also be linked to tools that need to be executed continuously, such as post-build signing, so that builds[*8] can be executed automatically on a regular basis.

1) Development Tools in the Cloud

(1) Cloud management of source code

The source code is stored in the cloud and can be easily accessed from each person's work terminal regardless of location, so that the same source code can be shared by the entire development team. Also, by explicitly displaying commercial branches[*9] and development branches, we also devised a way to prevent regression[*10] and other problems.

(2) Review circulation tool

Although the source code is reviewed by experts from the development team, we introduced a tool to track the log of who circulated the code and what was pointed out and corrected so that the opinions of the team can be widely confirmed. This helps to eliminate omissions of revisions and also enables full management of review progress within teams.

(3) Source code analysis and obfuscation[*11]

Before a build, static parsing[*12] of the source code is performed. Previously, results were not shared because we performed this on each person's work terminal. Currently, the results of the analysis are displayed in the cloud so that each person can check them and they can be deployed horizontally.

Apps are also obfuscated before being made commercially available to avoid analysis by reverse engineering[*13]. To ensure that there are no obfuscation omissions, obfuscation is performed using a tool in the cloud, and the results are logged on a server so that the status of obfuscation can be shared among development team members.

2) Tool Automation

(1) Automatic build execution

In general, development team members often build applications individually as needed,

*8 Build: A process or operation to create files executable on a terminal or distribution packages based on source code.

*9 Branch: In version control, a branch from the master history that is recorded. For example, a commercially available version is called a commercial branch, and a version that is not yet commercially available and still under development is called a development branch.

*10 Regression: Reduced performance or function degradation that occurs with software upgrades, mainly caused by the revival of past defects.

but in such cases, there are issues such as the version to be verified being different for each member. For this reason, we introduced a rule that the system would automatically run builds on a regular basis (specifically, once a day at night) and work the next day with that app as the latest version. This means the entire team can now share the most recent app version, and hence there is no more regression caused by proceeding with verification work on old apps. In addition, by keeping and sharing the history of the changes as a log on the server, the entire development team is made aware of source code changes and the number of missed changes has been reduced.

(2) Signing automation

To clarify to the user that apps are for services provided by NTT DOCOMO, apps are given a signature. Until now, apps built by the development team have been uploaded to the signature granting server by the Communication Device Development Department for signature granting. However, to eliminate the hassle of transferring files to and from the development team and to speed up development, the Communication Device Development Department is now responsible for building source code stored as a commercial branch in the cloud. In addition, the manual signature granting process was eliminated by creating a script[*14] that automatically sends apps to the signature granting server after a build and grants them signatures.

3.3 Test Phase

Conventionally, after the final sprint when development of all requirements was complete, operation of all requirements was checked as an acceptance test and a market release decision was made. This required lots of time required for acceptance testing, and many defects were detected at the end of development. This approach resulted in rework and time needed to fix problems before market release, making it impossible to achieve short release cycles.

Therefore, to detect defects as early as possible in the development stage, we decided to conduct some of the tests, which had been conducted collectively in acceptance tests, in front-end processes during the sprint period.

1) User Scenario Testing

As shown in **Figure 3** (a), before improving the test phase, user scenario testing was conducted after the final sprint when all functions had been implemented, which resulted in long test time and detection of many defects at the end of development. Therefore, we changed to a policy of conducting user scenario testing of developed requirements in each sprint (Fig. 3 (b)). This resulted in early detection and solving of problems early in the development process. The time required for testing was also shortened by changing requirement verification after the implementation of all functions to exploratory testing[*15].

2) Regression Testing

Similar to user scenario testing, before test phase improvements, regression testing was started after the final sprint when all functions had been implemented, and defects were detected at the end of development (Fig. 3 (a)).

*11 Obfuscation: Processing of source code to make it difficult for humans to understand without changing the behavior of the program so that malicious users cannot easily decipher or tamper with the application.

*12 Parsing: Analyzing source code to find out where it violates rules or syntax.

*13 Reverse engineering: The study and clarification of the technical information of an application by observing the behavior of its software or analyzing its source code.

*14 Script: A simple programming language for describing programs for simple processes. A program described by a script may also be called a script.

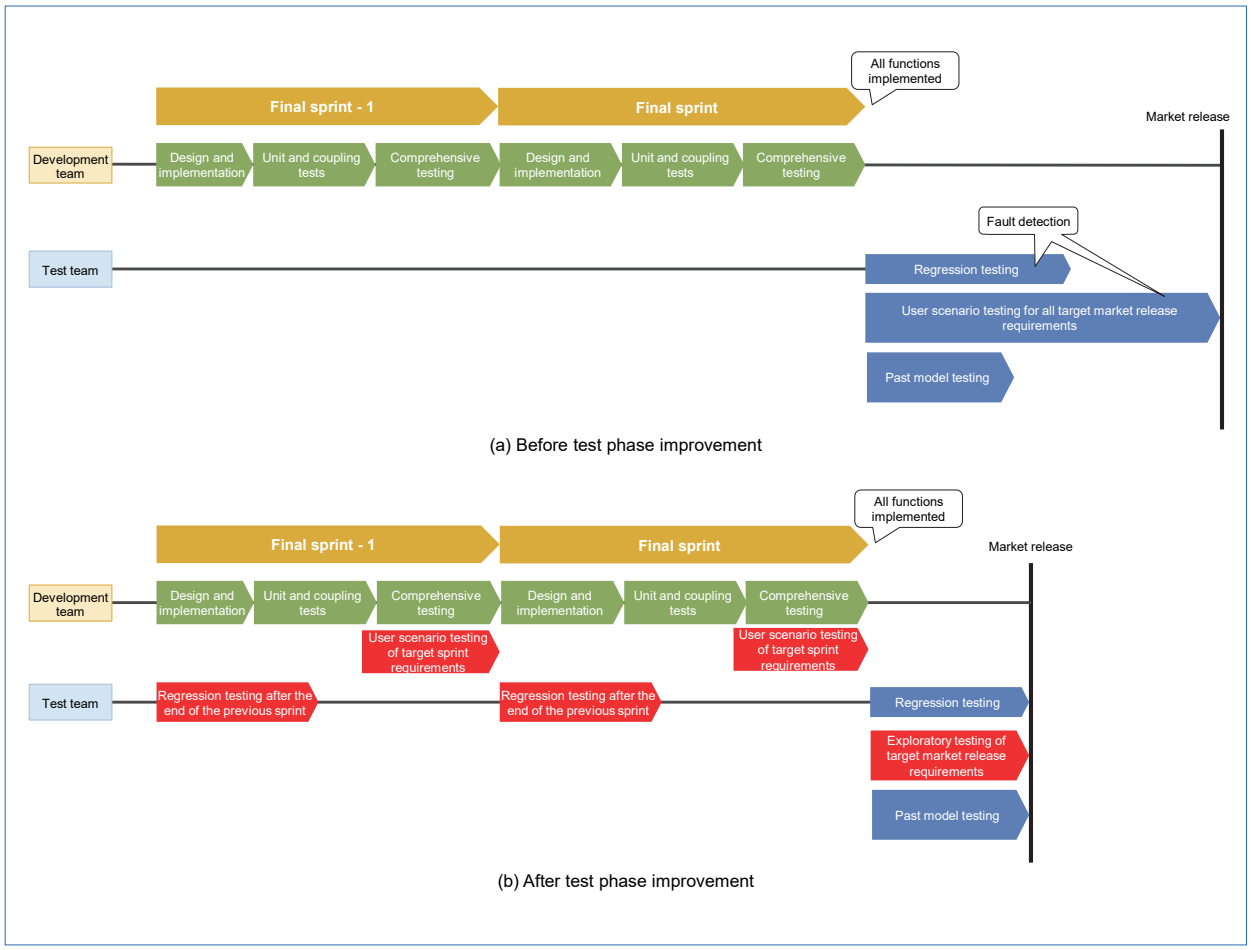

Figure 3 Shortened test time with improved test phase

Therefore, we changed the policy to conduct regression testing on artifacts at the completion of each sprint. In consideration of the number of man-hours required for testing, we also automated regression testing. This made it possible to detect and correct defects in the early stages of development. Defects are now rarely detected at the end of development (Fig. 3 (b)).

3) Effectiveness

Since we had already confirmed that artifacts were up to the market quality level at the completion of each sprint, after implementation of all functions was complete, only the minimum operation check was conducted as the final quality check before market release. Testing used to take about a month on average after implementing all functions, but this has been reduced to about three business days.

4. Changes Associated with Remote Development

4.1 Overview

Considering the risk of a mass outbreak of the

*15 Exploratory testing: A method of executing tests while checking software behavior and test results, rather than creating test cases in advance. Since it does not involve prior test design or pattern exhaustive testing, exploratory testing can be conducted more quickly than conventional descriptive testing and is well suited for agile development.

novel coronavirus in the agile development team, the team shifted to remote work development. This involved various Scrum ceremonies also being changed from face-to-face meeting to methods suitable for remote development. Although there were some issues unique to remote work, such as communications and security, we were able transition smoothly by devising methods that did not compromise productivity.

4.2 Examples of Migration to Remote Work

1) Various Scrum Ceremonies

(1) Daily Scrum

Before transitioning to remote work, all members would gather in front of a large screen monitor displaying the ticket management tool to check the progress of each member. After the transition to remote work, a Web conference system was introduced to check the progress of each member while sharing the ticket management tool on the screen so that the daily Scrum can be held as it was before transitioning to remote work.

(2) Sprint planning, sprint retrospective

For these two ceremonies, we are conducting Web conferences in which participants can see each other's faces, as these are Scrum ceremonies with a lot of discussions among members. Early in the transition to remote work, we would hold Web conferences without showing our faces, but since we could not read each other's reactions and thoughts through audio alone, we gradually lost the ability to have active discussions due to anxiety and the impact this had on motivation.

Therefore, we made it a rule in principle to participate in Web conferences showing our faces. By showing our faces to each other we can obtain additional visual information from non-vocal information such as facial expressions, gestures and eye contact. As a result, we are now able to have active discussions to a degree similar to before the transition to remote work.

For sprint retrospectives, before the transition to remote work, the KPT method was used with labels and simili paper. While physical tools are no longer available due to the transition to remote work, sprint retrospectives can be operated in the same way as before the transition to remote work by using spreadsheets[*16] instead. As a side effect of using spreadsheets, it became easier to check past history because it was electronic.

(3) Sprint review

Before the transition to remote work, operations were checked using the actual device at the sprint review and the PO team made the acceptance decision on the spot. In contrast, since the transition to remote work, we have been using a test distribution tool to distribute the app as a sprint artifact so that each member can download the app to an actual device at home and check its operation. We also have prepared demonstration videos and play them during sprint reviews to prevent any discrepancies in perception among members.

These initiatives have made it possible for the PO team to make acceptance decisions

*16 Spreadsheet: A type of Web application that can be edited by multiple people at the same time. Because it can be known who is editing which part of the sheet in real time, a spreadsheet can be used as a substitute for sticky notes or simili paper and hence enables efficient work.

at sprint reviews, just as they did before the transition to remote work.

(4) Backlog refinement

Backlog refinement, which used to be conducted in person before the transition to remote work, can now be conducted in the same way as before the transition thanks to the introduction of the Web conference system.

2) Others

(1) Communication outside Scrum ceremonies

The entire team is always connected by voice during working hours by using a tool that has a voice chat function. This enables member to feel free to talk to each other even when minor questions arise and communicate closely similar to when everyone was at the same location before the transition to remote work.

Other than voice, we also use text chat as a real-time communication method. To facilitate communications when complex explanations are required for operating procedures or causes of problems for example, text chat can be used as a supplementary tool to voice communication. Before the transition to remote work, we used a white board to align members' thinking, but since the transition to remote work, we use text chat instead.

(2) Workshop

We have deepened relationships among the members and promoted self-organization*17 by holding regular workshops where each group discusses issues related to development and feeds back the results to the development. Since it became difficult to hold face-to-face meetings after the transition to remote work, relationships cultivated during development at our base are being utilized even after the transition to remote work and have led to the maintenance of productivity even under remote work by fostering an atmosphere where opinions can be frankly discussed.

In the future, we are also considering holding workshops remotely so that we can continue to maintain these relationship even when members are replaced.

(3) Security

Information security risk is an important factor in remote work development. When the development was done in groups at our base, an ID card was required to enter the room, and people who were not involved in the project were not allowed in. However, when development is done at home, people who are not involved in the project, such as family members, may be able to enter and leave workrooms. Similar to physical measures, it is important to set certain rules and ensure that development team members follow those rules. The Telework Security Guidelines from the Ministry of Internal Affairs and Communications recommend the implementation of measures that balance rules, people and technology [3]. From early on in our project, we have been working with the development team to study countermeasures against information security risks, as described below.

*17 Self-organization: Instead of having a manager, team members manage projects autonomously. Members make plans themselves, share daily progress and solve problems.

(a) Formulation of rules for secure operations

In formulating the operation rules, we referred to the content of the guidelines set by the security departments of the companies involved in the project.

Specifically, work is to be done in a secure environment such as an isolated private room, the use of loaned PCs is mandatory, the use of personal PCs is prohibited, free software is not to be installed, and PCs are not to be taken outside the home, etc.

These operational rules were made into an itemized checklist. The entire team first reads the document, and then each person is asked to check it once a month to ensure that individuals regularly reacquaint themselves with it and that it is being used appropriately.

(b) Source code and documentation management

The source code is stored in the cloud. To maintain security, the development environment on the cloud is accessed via a Virtual Private Network (VPN) connection*18 after authentication with an ID, password, and client certificate assigned to each individual.

Documents such as specifications and design documents are stored on a server at the development vendor base. The information on the server can be viewed and edited using a remote desktop, which allows remote control of the PC at the base from the home. It is prohibited to physically take the information home.

(c) Device management

For remote work development, it was decided to prohibit the use of individually purchased computers and to mandate use of computers loaned by the company. The development of the docomo TV terminal app also requires verification of operations on various smartphones and tablets. This meant it was necessary for each member to also take home smartphones or other devices to check the app. To manage and ensure that these devices are in proper working order, a visual check is conducted every morning via video call to check where PCs and devices are and ensure that they can be turned on. This reduces the risk of loss or malfunction, and also enables quick response in case of loss or malfunction.

5. Productivity and Quality in Agile Remote Working

5.1 Productivity

We examined team productivity when they were developing together at the base and after the transition to remote work.

1) Velocity Trends

In Scrum development, there is a number called velocity. The average amount of work that a Scrum team completes during a sprint is expressed by a unit of “story point,” A value expressed in this way is called “velocity.” Velocity is a relative value and is used as a reference to see how much work (tickets) can be completed in the next sprint, and to understand the growth of the team.

*18 VPN connection: A method of connecting to a public network such as the Internet using a virtual dedicated line environment protected by authentication and encryption technologies.

Figure 4 shows the velocity of the docomo TV terminal app development team and the trend in corresponding story points per person. In this project, the number of people in the development team changed at the same time as the transition to remote work, so we checked the story points per person trend before and after the transition to remote work rather than the velocity of the entire team. Immediately after the switch to remote work, there was a temporary drop in the trend, but this recovered and stayed at the same level as before the switch, indicating that the team was able to operate without losing productivity.

Since we were able to operate without losing productivity, there has been no impact on release schedules or frequency of app releases.

2) Happiness Level Trends

Happiness level is a five-point score of how happy individuals feel based on their work content, role and ease of working, and is an indicator of whether the organization is a vibrant place to work. In our project, we interviewed members of the development team about their level of happiness in each sprint, calculated the average, and analyzed this trend (**Figure 5**).

Immediately after the transition to remote work, many people were confused by the change in environment and were temporarily depressed but later recovered. We heard many positive comments about working on development from home, such as having more time to spend with family. In agile development, it is said that the higher the happiness level of the team, the higher the productivity. It can be seen that the happiness level trend is similar to the velocity trend shown above.

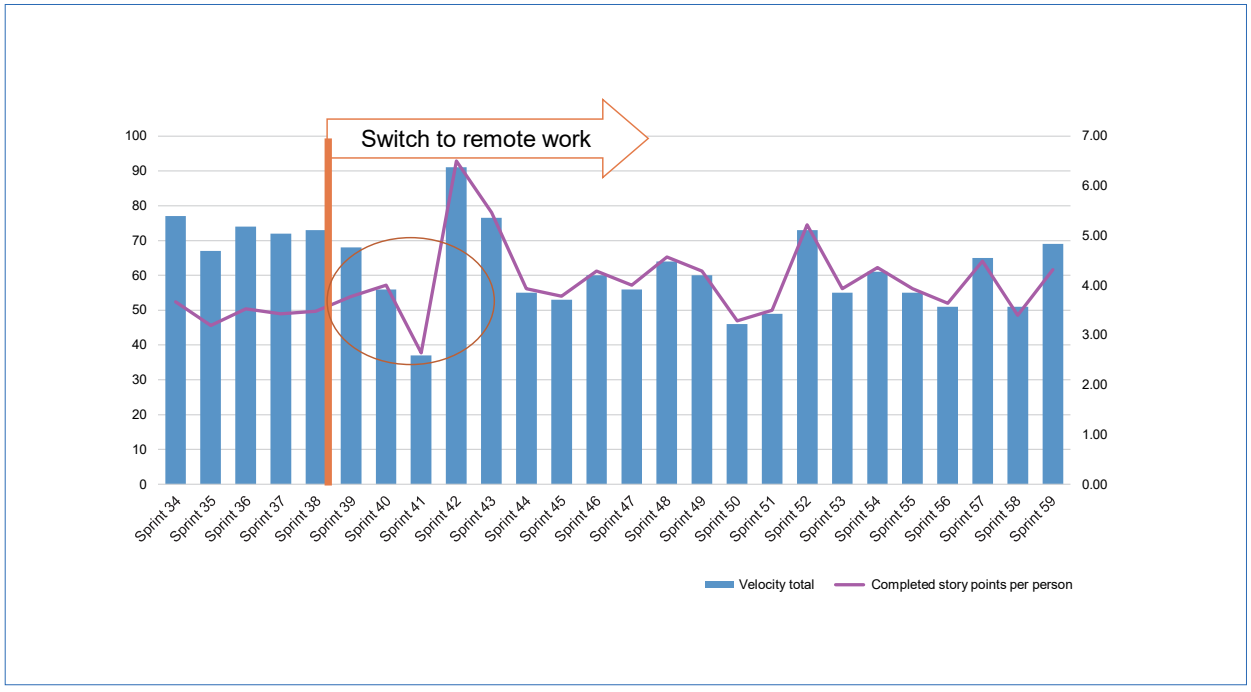

Figure 4 Trends in velocity

5.2 Quality

To verify whether there was any change in the quality of the app before and after the transition to remote work, **Figure 6** shows the bug detection rate per number of kSteps added during the development phase. The quality target for the app is less than 1 item/kStep. We achieved less than 1 item in all sprints, meaning that compared to before, there was no decline in quality after the switch to remote work.

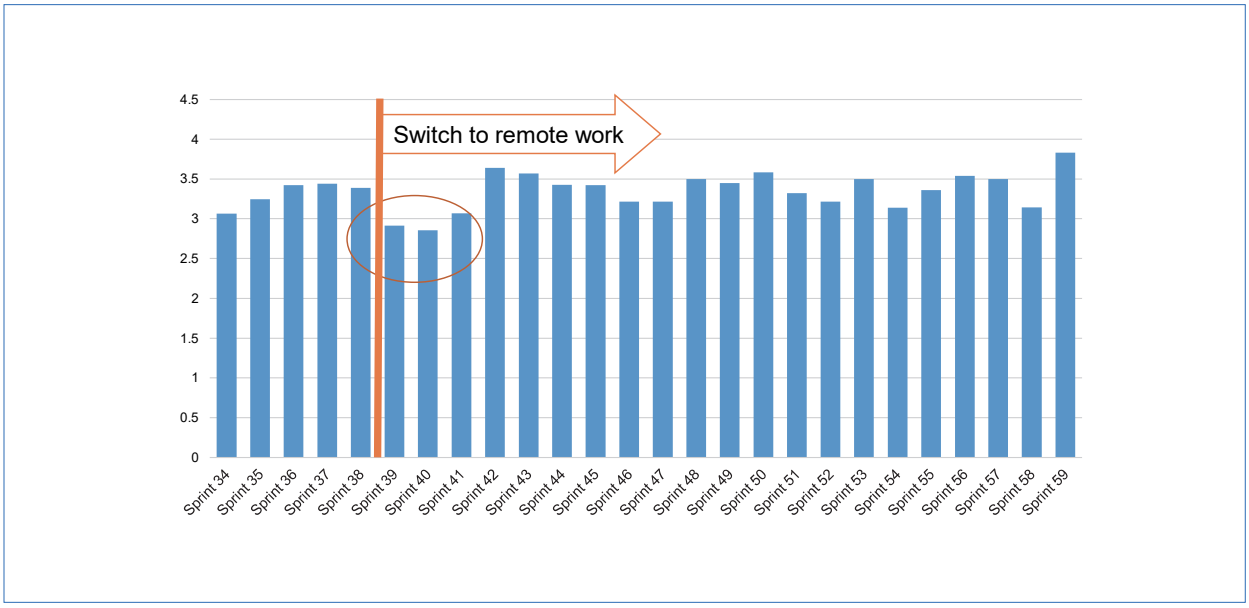

Figure 5 Trends in happiness level

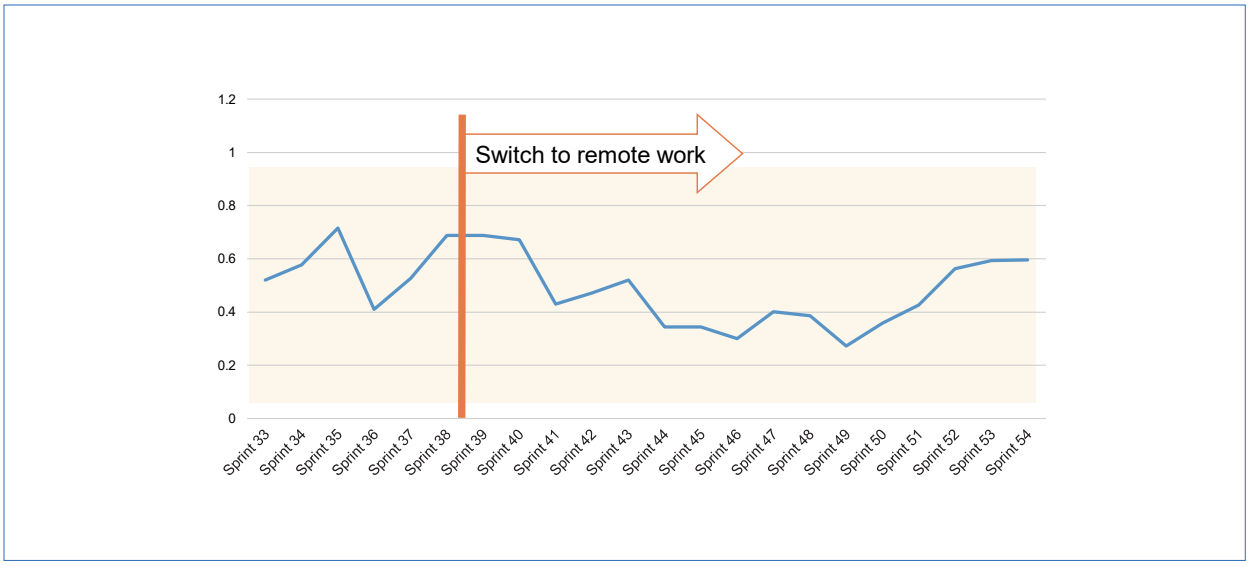

Figure 6 Trends in bug detection rate per kStep added in the development phase

6. Conclusion

In this article, we described a case study of an agile development project's transition to efficient remote development. With remote development, we were able to achieve short release cycles while maintaining quality and productivity. Agile development is a method that involves continuous improvement. Hence, we will continue our efforts to improve it and achieve more efficient development going forward.

REFERENCES

[1] K. Beck et al.: "Manifesto for Agile Software Development," 2001.
https://agilemanifesto.org/iso/en/manifesto.html

[2] K. Schwaber and J. Sutherland: "The Definitive Guide to Scrum: The Rules of the Game," Nov. 2020.
https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

[3] Ministry of Internal Affairs and Communications: "Telework Security Guidelines" (In Japanese).
https://www.soumu.go.jp/main_content/000545372.pdf