PAPER

A Case Study of Requirements Elicitation Process with Changes

Takako NAKATANI[†a], ***Member***, **Shouzo HORI**[††], ***Nonmember***, **Naoyasu UBAYASHI**[†††], **Keiichi KATAMINE**[††††], ***and*** **Masaaki HASHIMOTO**[††††], ***Members***

SUMMARY Requirements changes sometimes cause a project to fail. A lot of projects now follow incremental development processes so that new requirements and requirements changes can be incorporated as soon as possible. These processes are called integrated requirements processes, which function to integrate requirements processes with other developmental processes. We have quantitatively and qualitatively investigated the requirements processes of a specific project from beginning to end. Our focus is to clarify the types of necessary requirements based on the components contained within a certain portion of the software architecture. Further, each type reveals its typical requirements processes through its own rationale. This case study is a system to manage the orders and services of a restaurant. In this paper, we introduce the case and categorize its requirements processes based on the components of the system and the qualitative characteristics of ISO-9126. We could identify seven categories of the typical requirements process to be managed and/or controlled. Each category reveals its typical requirements processes and their characteristics. The case study is our first step of practical integrated requirements engineering.
key words: *requirements engineering, requirements elicitation process, case study*

1. Introduction

Requirements changes are considered detrimental to the success of a development project, though, requirements are often changed. However, various solutions do exist to cope with this problem resulting from requirements changes. For example, Trawling in *Volere* [1] is a method for requirements elicitation without possible omissions. Goal-oriented analysis methods explore the "why" aspect of requirements in order to find alternative requirements and forecast variability of requirements [2], [3]. Theory-W [4] tells us the risks of projects and ways to negotiate with stakeholders. In a practical sense, it is hard to elicit requirements completely in the early stage of software development.

There are several development process models designed to cope with requirements changes in the waterfall model place. The agile development process [5] recommends incremental development in short durations, in which case engineers define test specifications when they elicit requirements. If the development duration is short, the project may be free from requirements changes caused by business environmental changes. Therefore, the problem of requirements changes is beyond the scope of this technique. The Unified Process [6] proposes a four-stage development. It focuses on a framework that takes measures to meet the future requirements changes.

We need a design technique and/or framework for coping with requirements changes quickly. In component-based development, web services and SOA (Service Oriented Architecture), there are techniques with frameworks aimed at loose coupling and dynamic binding between modules. These concepts are effective in coping with requirements changes by replacing components even during a run time and, are techniques that can be considered as a method for protecting a project from requirements changes during the software evolution stage.

To deal with the volatility of requirements, it has been proposed that requirements processes should be integrated with other development processes, such as design, implementation, and testing processes [7]. This idea or concept is referred to as integrated requirements engineering. Though we agree with this idea or concept, before initiating integrated requirements engineering, we have to manage requirements changes in order to save our projects. In an adequate triage for requirements changes, we negotiate with stakeholders on quality, cost and delivery by using quantitative development data [8]. The focus of our research is to observe requirements changes quantitatively, and to visualize them for the planning of the future requirements processes as well. We must learn as much as possible from past work in order to maximize our efforts for the future.

Our case study aims to learn the real requirements changes processes following the requirements analysis phase. Therefore, we analyze the requirements process after the requirements analysis phase has been completed. The requirements changes need to be categorized in order to have applicable uses on the future projects. It means that if a manager can identify a set of requirements within the developing system as the members of a category, he/she can manage and/or control the requirements elicitation processes according to the process of the category. "Control" means that requirements engineers explain the requirements elicitation

Manuscript received February 5, 2009.
Manuscript revised October 24, 2009.
[†]The author is with the Graduate School of Systems Management, the University of Tsukuba, Tokyo, 112–0012 Japan.
[††]The author is with Quality Assurance Dept. Yaskawa Information Systems Corporation, Kawasaki-shi, 215–0004 Japan.
[†††]The author is with the Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka-shi, 819–0395 Japan.
[††††]The authors are with the Graduate School of Computer Science and Systems Engineering, the Kyushu Institute of Technology, Iizuka-shi, 820–8502 Japan.
a) E-mail: nakatani@gssm.otsuka.stukuba.ac.jp
DOI: 10.1587/transinf.E93.D.2182

process of each category to their customers and elicit requirements with their customers' cooperation to follow the ideal process of the category.

After studying project records both quantitatively and qualitatively, we have categorized the requirements process according to the components contained within a certain portion of the software architecture. Each category reveals its typical requirements processes and their characteristics.

This paper introduces the case study as the first step of practical integrated requirements engineering. In Sect. 2 we give an overview of the project. Section 3 illustrates the results of our study and provides the requirements processes of the target project. In Sect. 4, we categorize the requirements processes into seven categories, followed by a discussion of the characteristics of each. In Sect. 5, we introduce related work. In the final section, we present our conclusions.

2. Overview of the Project

The target project was initiated to develop a restaurant service and order management system, named *RESORT*. *RESORT* can accept orders from the hand held terminals of staff members and from table terminals. The table terminals are situated on guests' tables to provide the food menu, recipes, knowledge of liquor and sake and, entertainment information via a wireless network from the application server. A previous system was developed by a certain company several years ago and, the client of this project decided to re-develop the system to improve its installation load and extensibility. The system was required to be able to replace its Order Entry System (OES) component with one of those provided by various companies.

The project was carried out by one client with three developer companies. One of the developer companies provided table terminals with an OS and a browser for HTML. The second company developed contents for the table terminals. The third company, the *YSKcom*, developed an application server that could communicate with the other components. The client provided a shared repository for the project to manage the schedule and problems. The developers communicated with each other via the repository and solved any problems that occurred during the development. OES providers were not members of the project, as the specifications of the OESs were made available to the project via the client.

Before starting the project, the *YSKcom* proposed the software architecture illustrated in Fig. 1 to solve the problems of the previous system. In the figure, the white icons represent components developed by the *YSKcom*, the dotted line rectangle represents the OES, and the gray colored icons represent the components developed by other developers. Our research focuses on the components in white. This architecture was elaborate and shared with the developers in order to understand the functions of the components.

The *RESORT* project was completed in four and a half months, although five months were originally budgeted. Development data were collected in 117 days, from the external design phase to completion of the project. The duration of each phase was accomplished as shown in Table 1. We calculated the duration beginning with the start of the external design phase. The duration of the integration testing phase was actually 27 days according to the recorded data. However, we counted it as seven days, since only four requirements were elicited in the first seven days of the phase. Thus, the project was completed on the 117th day.

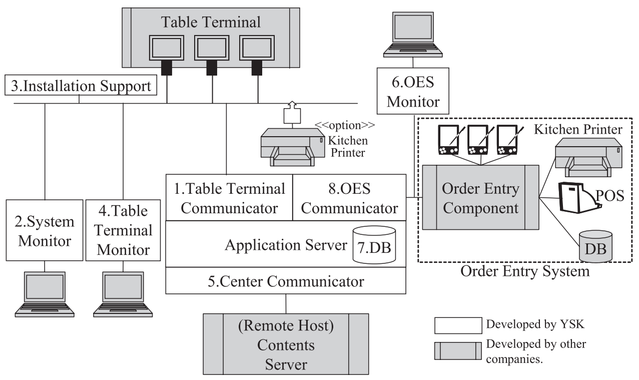

Fig. 1 The architecture of *RESORT*.

Table 1 The duration and starting date (relative to the project start) of each phase.

Phase	Day (relative to start date)	Duration
External Design	0	55 days
Internal Design	47th	40 days
Implementation	64th	45 days
Integration Test	93rd	7 days
System Test	106th	11 days
Project completion	117th	-

3. Requirements Process

3.1 A Case Study Method

The *YSKcom* provided us with their development data. The engineers from the *YSKcom* used traditional interview techniques in multiple non-scheduled reviews with their client. They managed issue records, question and answer records and decision records for each phase. From the records we can see the content of each requirement, its priority, the date of contact/recording/completion, the name of a client/an engineer in charge/the manager, agreement of stakeholders, and the reason for the change/addition/deletion of a requirement.

The requirements specification of the *RESORT* system was composed with use cases. A requirement change had been proposed to the use case defined in the initial requirements specification. If a new requirement was brought up to the system, it was regarded as a new use case. Therefore, we could observe the requirements changes of the system based on the use cases. Each use case is a bag that receives requirements changes, which by the end of the project has matured. Recently, the use case model has been applied to

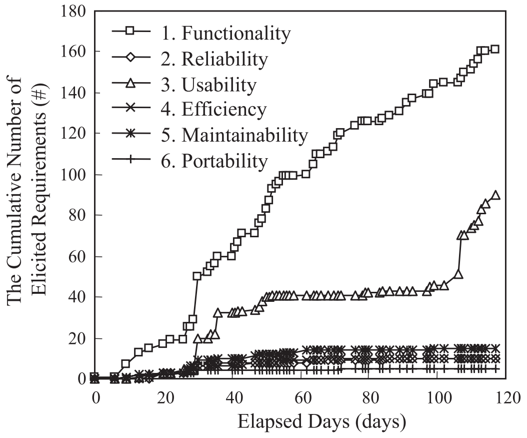

Fig. 2 Requirements process for quality characteristics.

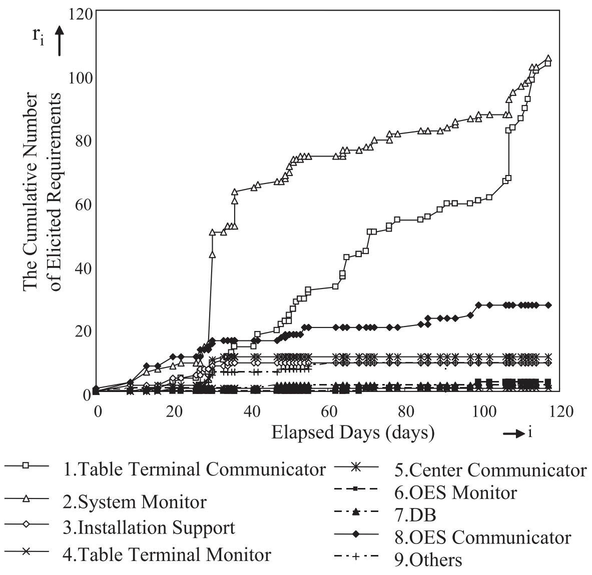

Fig. 3 Cumulative number of requirements for each component.

projects and managed as a base of requirements. Therefore, counting requirements changes based on the number of use cases is a practical application for a lot of projects.

From the estimation viewpoint, the granularity of each use case should be challenged. However, from the requirements changing observation viewpoint, the granularity of each use case is out of the scope of our research, since we only want to know how many the use case receives requirements changes during its development. In order to observe requirements changes for each use case, its characteristics for categorization and/or background for future measures are more essential than its granularity.

We analyzed these documents and counted the number of additional requirements, changed/adjusted requirements, and deleted requirements. Thus, if a newly added requirement was deleted after being changed, we counted it three times.

The quantitative data were analyzed in two ways: first, on a quality characteristics basis, and second, on an architectural basis. Furthermore, we interviewed both the project manager and the quality control manager of the project to understand the background of the records and their rationale.

3.2 From a Quality Characteristics View

Figure 2 shows the requirements processes from a quality characteristics [9] view. The x-axis represents the elapsed days of the project, while the y-axis represents the cumulative number of elicited requirements. From this graph, we can see that the requirements of 1. functionality and 3. usability were elicited continuously throughout the project, while other requirements were frozen midway through the project schedule.

The continuous process of developing the functionality requirements represents the learning curve of the engineers, as they did not have sufficient knowledge to understand the domain of *RESORT*. They had to develop the system as they acquired the knowledge in a step by step manner. For example, the requirements of managing a group of tables for which one bill was required, was elicited after the requirements analysis phase. Furthermore, there was another reason of the continuous requirements elicitation: the OES providers did not make their specifications available to the project in the early phase of development. The major provider only released his specification to the project after the implementation phase had started, i.e., on the 64th day of the project schedule.

The continuous process for the usability requirements represents not only a process of trial and error to achieve better usability, but also a process of eliciting undefined requirements by evaluating the system adequacy. In fact, although the project was preceded by the waterfall model, the part relating to the usability or user interfaces was repeatedly developed and validated. If a usability specialist or domain specialist had been available to join or advise the project, the process might be represented by a different curve. The requirements elicited in the system testing phase were transferred to the next version of the system.

Regarding the non-functional requirements, these were defined as a solution to the problems of the previous system. The client recognized the quality problems of the previous system based on their business objectives. Therefore, these processes could be frozen before the internal design started.

3.3 From the Component View

The functionality and usability requirements depend on components. Figure 3 illustrates a requirements process for each component. In the figure, the x-axis represents elapsed days of the project, while the y-axis represents the cumulative number of elicited requirements. The results show that, 1. the Table Terminal Communicator (*TT_C*) and 2. the System Monitor (*Sys_M*) received many requirements throughout the project. These components are situated in the user interface area. 8. the OES Communicator (*OES_C*)

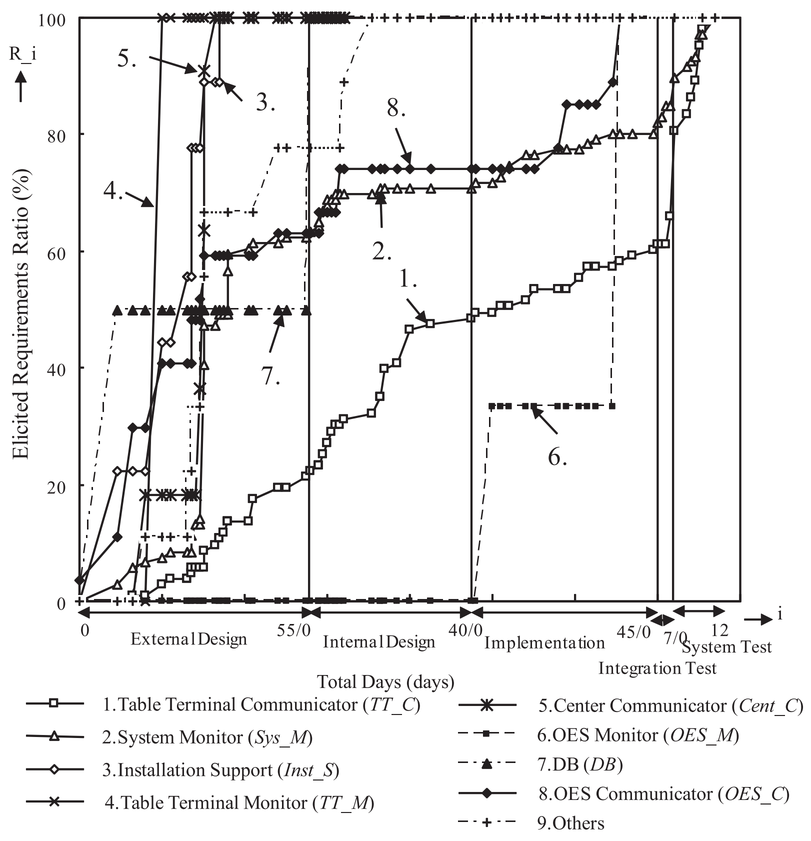

Fig. 4 Requirements elicitation ratio.

also shows changes to its requirements in the latter part of the development. This component depends on the specifications of OESs. The system was required to connect to multiple vendors' OESs, the specifications of which were, in fact, provided after the implementation had started. Until then, the engineers had collected product information on OESs informally.

We focused on the requirements elicitation ratio according to the project schedule. It provided us with valuable information, such as when and which requirements were frozen, in other words, that one hundred percent of the requirements of the developing system had been reached. Let r_i be the number of elicited requirements on project day i. Thus, the sum of a set of elicited requirements in n days can be written $\sum_{i=0}^{n} r_i$. The ratio of the accumulated requirements R_i on the ith day can be defined as

$$R_i = \sum_{j=0}^{i} r_j / N$$

Here, N is the sum of the number of requirements elicited throughout the project. Hence, R_0 is equal to 0% and R_{end} is equal to 100%.

Figure 4 shows the ratio of elicited requirements R_i versus the project schedule. The project schedule in this figure represents total days. For example, the internal design phase was planned after the external design phase, which took 40 days. Its process appears after the external design phase in Fig. 4. As a result, the total number of days of the project is shown as 154 days, rather than the actual development duration of 117 days. The vertical line in Fig. 4 represents the end of each phase.

To clarify the reason and rationale of the requirements processes, we conducted interviews with both the project manager and quality control manager. Then, we analyzed the rationale of the requirements processes according to the components. The numbers in the following list correspond to the numbers shown in Fig. 4.

- 1. Table Terminal Communicator (*TT_C*) and 2. System Monitor (*Sys_M*)
  The requirements have been elicited continuously from an early phase to the end of the project. These components are in the user interface area. For example, the *TT_C* depended on the Table Terminal (*TT*) used by guests of the restaurant.
  The *TT* was equipment that provided a lot of information to the customers of the restaurant and, which sent customers' requests to the *TT_C*. It means that the functionalities of the *TT_C* to manipulate and comply with the customers' requests were the keys to making *RESORT* a competitive system.
  The *Sys_M* is for use by the manager of a restaurant. It provides information on the active or inactive status of the other equipment of the system, which decreases the manager's labor involvement in the system. There-

fore, its functionality and usability were the keys in enabling *RESORT* to compete with similar products of other companies.
Figure 4 represents that the engineers have elicited requirements for these components repeatedly to improve its functionality and usability.

- 3. Installation Support (*Inst_S*)
The previous system of *RESORT* required users to make a lot of effort to install it in a restaurant. The major goal of *RESORT* was to lighten the installation effort. This requirement was essential for *RESORT*, from a product business view. Therefore, the client could clearly define their requirements for the *Inst_S* early in the development.
Figure 4 represents that the engineers have elicited requirements for the component completely in the external design phase.
- 4. Table Terminal Monitor (*TT_M*) and 5. Center Communicator (*Cent_C*)
These components depend on external interfaces provided by the co-developers. Figure 4 represents that the engineers have elicited requirements for the component completely by the end of the external design phase.
Its requirements elicitation process was similar to the process of the *Inst_S*, however, its context was different. The members of the development team highly prioritized the review of these specifications to discover incomplete requirements and, could thus freeze the requirements in the early phase. They were able to collaborate well by sharing the problems of the project via the shared repository. The process represents that, the members have cooperated on the project to keep the interface specifications.
- 7. *DB*
The requirements elicitation ratio could have reached 100% by the end of the external design phase. Figure 4 represents a similar shape as the *TT_M* or the *Inst_S*. However, its context was not the same as others.
The project reused the specifications of the former system with some minor changes, since the engineers were able to identify the insufficient schema of the existent database when the project was initiated.
- 6. OES Monitor (*OES_M*) and 8. OES Communicator (*OES_C*)
These components depend on other components provided by outside companies. Furthermore, two developers with insufficient knowledge of OESs were assigned the task of managing the integration of the system with respect to multiple OESs. Given this situation, the developers needed to study the OES domain based on the most popular OES product. After designing the component to fulfill these specifications, they then proceeded to adopt other OES products and change the design. Because of resource problems, it was difficult for them to fulfill every OES specification at one time.
Figure 4 shows that the requirements have been elicited continuously from an early phase to the end of the unit testing phase.
The *OES_M* was a hazardous component. Figure 4 shows that its requirements have been elicited accidentally in the implementation and unit testing phase. The developers realized that several interfaces were not available for one of the target OESs. They had to change the requirements of the *OES_M*. Such requirements changes can be detrimental to the project schedule. Fortunately, the *OES_M* had fewer functions and the changes to the requirements were the deletions of functions developed for the most popular OES, although the changes could possibly cause several inconsistencies in the system. The accidental requirements changes were caused by the non-cooperative company which did not provide the proper specifications of the OES to the project. It tells us that we cannot always expect the cooperation of other organizations, and thus we must always be prepared to deal with such situations.

In the real world, a requirements process expresses mixed processes, since a requirement is constrained by an architectural aspect, as well as the context of the project. The *OES_C* and *OES_M* are examples of this. They are affected by the level of cooperation of the OES companies, the engineers' knowledge, and the limitation of project resources, as well as the client's business strategy. The client believed that *RESORT* should be able to integrate with most available OESs in order to compete with similar products.

In the next section, we build a requirements process model based on our case study.

4. Discussion

In defining the requirements process model, we decompose the requirements process characteristics into seven categories and simplify realistic situations. If a manager can identify a set of requirements of the developing system as the member of a category, he/she can control and/or manage the requirements elicitation processes according to the process model of the category. Of course, the model needs more case studies in order to apply it to an actual project. The method and the complete process model will result from our future work. However, the model derived from this case study offers implications in controlling and/or managing requirements changes.

Figure 5 depicts graphs showing each category of the requirements process. The *x*-axis represents the elapsed time of the project, while the *y*-axis represents the requirements elicitation ratio. We define each category as follows:

- Reused type (*TypeRu*)
An example of this type as observed in the case study is *DB*. The component was reused from the previous system, as the developers identified its reusability in the early stage of the development. This is the reason

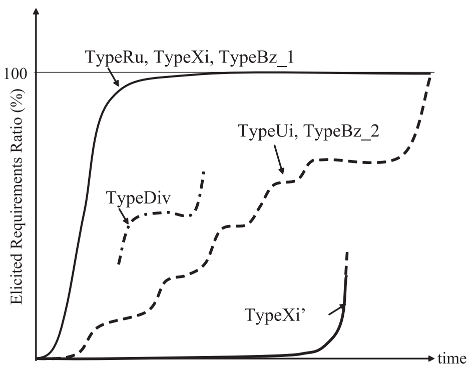

Fig. 5 Types of requirements and their observed processes.

why its requirements elicitation ratio can reach 100% by the end of the external design phase.
Therefore, when the project is able to identify the sufficient and insufficient requirements of an existent component, the project can decide to reuse the component with clearly defined changes. Thus, based on our case, the requirements of components belonging to *TypeRu* should be elicited in the early development phase.

- External interface dependent type (*TypeXi* and *TypeXi'*)
This type is for those components that depend on external components. The *TT_M* and *Cent_C* are examples of this type. The developer teams successfully procured their specifications in the early development phase through cooperative work.
On the other hand, *OES_M* and *OES_C* are also components that depend on external components. Their specifications were not available until the implementation had started, because the vendors of these components did not cooperate with the project.
Thus, if the vendor of an external component cooperates with the project, those requirements should be frozen in the early development phases. It is modeled as *TypeXi* in Fig. 5. Basically, external interfaces can be defined in the early development phase if their specifications are available.
If a vendor does not cooperate with the project, the situation can be made worse. A manager should negotiate with vendors to get their cooperation. When it is not possible, the manager should ensure the project's resources to cope with the accidental requirements changes of external interfaces of uncooperative vendors. Such a case is shown as *TypeXi'* in Fig. 5. This type is a problematic type.
- Diversity type (*TypeDiv*)
An example of this type can be observed in the case study as *OES_C*. The requirements for this were elicited continuously, because of the lack of knowledge of the developers.
This type needs an appropriate design to deal with the variability of components. Developers can design adequately if, and only if, they have enough knowledge of the variations [10]. Therefore, if developers do not have an adequate knowledge to allow them to design the internal system for the external design phase, the manager should decide to select incremental development and/or trial and error based development. When the incremental development is selected, the project can elicit requirements according to the development targets. On the other hand, if developers can have enough knowledge to understand the diverse external interfaces, they should acquire information of every external interface in the early development phase.
- User interface dependent type (*TypeUi*)
Examples of this type as observed in the case study are the *Sys_M*, *TT_C*, *TT_M*. These components depend on user interfaces. According to the case, we could clearly observe that the components' requirements tend to be added and/or changed even in the validation testing phase. For such a component, therefore, the project should plan incremental requirements elicitation and also apply incremental development by prototyping [11], [12], if the project cannot employ a usability specialist.
- Business requirements dependent type (*TypeBz*)
There were two kinds of components related to business requirements. One is observed in *Inst_S*. Requirements of this component were elicited completely by the end of the external design phase, since its requirements motivate the client to initiate the project. The other is observed in *Sys_M* and *TT_C*. Requirements of these components were elicited continuously throughout the end of the project, since the project had to improve the requirements in order to compete with similar products provided by other companies.
Therefore, we have to divide *TypeBz* into two subtypes. *TypeBz_1* relates to business improvement. Requirements for this type should be clearly defined before the development starts. To define the requirements for *TypeBz_2*, prototypes are useful tools to validate the requirements in order to compete with other products. The project should require their clients to cooperate closely with the developers [13].

New requirements or requirements changes sometimes delay the project schedule. To save our project, we must identify the influential components that form the architecture, as well as plan and monitor the requirements processes of these components carefully. In our future work, we intend studying other cases to evaluate the adequacy and/or inadequacies of these types.

5. Related Work

Lehman mentioned that "there is no room in programming for imprecision, no malleability to accommodate uncertainty or error" in his Low of Program Evolution [14]. Therefore, we must focus on requirements processes. The problem is that it sometimes causes difficulties when software is changed in order to fulfill the requirements changes.

The requirements process during development has been extensively researched in various literatures. Aoyama and his colleagues have studied the kinds of requirements that were changed and/or added after the requirements analysis phase. They then introduced an interview guide to clarify unstable requirements, and further, evaluated the guide to elicit those requirements completely before the design phase begins [15]. We do not think that it is possible to elicit requirements completely in the requirements phase, because our clients sometimes do not know their requirements. Our approach is that the engineers should ascertain requirements together with their clients during the system development. To do this, we must manage the requirements process according to the type of each component and the project situation.

Arkley and his colleagues described an application of traceability by a company. They classified the project requirements and showed requirements changes in the development processes. In their article, the specification and requirements were frozen before the software design review [16]. Sankar and Venkat focused on a way to control requirements by showing the percentage of requirements frozen in the development processes. According to the article, 70% of all requirements were frozen during the requirements gathering [17]. If most requirements could be frozen in the early development phase, we would be happy. One of the real problems is that many requirements sometimes remain undefined until the design phase. We studied a case in which the engineers elicited requirements throughout the development to clarify the practical situation. In practice, the client cannot show all the requirements before the design phase has started.

Houdek and Pohl studied the requirements engineering process of Daimler Chrysler. They mentioned that 50% to 60% of requirements changes are in the interface area [18]. We categorized this type of component as *TypeUi*. They also mentioned that the requirements engineering activities are heavily intertwined. Sommerville referred to the intertwined requirements engineering process as integrated requirements engineering [7]. From our study, we can conclude that an integrated requirements engineering process needs requirements elicitation scheduling techniques, as well as elicited requirements management.

6. Conclusions

We studied the requirements processes of *RESORT*. From its development records, we could categorize the requirements processes into seven types. These types have been observed from a type-specific requirements process view. Our aim was to identify the type of each component in order to plan the requirements processes in the project. The results indicate that we would be able to schedule the requirements process according to the type of each component and the project situation.

Acknowledgments

This work was funded by the Joint Forum for Strategic Software Research (SSR). We would like to thank our colleagues at SSR. We would also like to thank Mr. Masao Shimoji and his colleagues at the Yaskawa Information Systems Corporation who cooperated on our study as interviewees.

References

[1] S. Robertson and J. Robertson, Mastering the Requirements Process, Addison-Wesley, 1999.

[2] A.I. Anton, "Goal-based requirements analysis," Proc. ICRE'96, pp.136–144, IEEE, 1996.

[3] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," Science of Computer Programming, vol.20, pp.3–50, 1993.

[4] B. Boehm, P. Bose, E. Horowitz, and M.J. Lee, "Software requirements negotiation and renegotiation aides: A theory-w based spiral approach," Proc. ICSE'95, pp.243–253, ACM, 1995.

[5] K. Beck, et al., "Manifesto for agile software development (http://agilemanifesto.org/)," 2001.

[6] I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999.

[7] I. Sommerville, "Integrated requirements engineering: A tutorial," IEEE Softw., vol.22, no.1, pp.16–23, IEEE, Jan./Feb. 2005.

[8] A.M. Davis, Just Enough Requirements Management: Where Software Development Meets Marketing, Dorset House, 2005.

[9] ISO/IEC, "9126: Software engineering–product quality," 2001.

[10] M. Jarke, "Requirements tracing," Commun. ACM, vol.41, no.12, pp.32–36, 1998.

[11] N. Deshmukh and S. Wadhwa, "A meta model for iterative development of requirements leveraging dynamically associated prototyping and specification artifacts," Proc. 15th IEEE International Requirements Engineering Conference (RE'07), pp.343–349, IEEE, 2007.

[12] A.K. Thurimella and B. Bruegge, "Evolution in product line requirements engineering: A rationale management approach," Proc. 15th IEEE International Requirements Engineering Conference, pp.254–257, IEEE, 2007.

[13] M. Haglind, L. Johansson, and M. Rantzer, "Experiences integrating requirements engineering and business analysis," Proc. IEEE Third International Conference on Requirements Engineering, pp.108–117, IEEE, 1998.

[14] M. Lehman, "Programs, life cycles, and laws of software evolution," Proc. IEEE, pp.1060–1076, IEEE, 1980.

[15] K. Aoyama, T. Ugai, S. Yamada, and A. Obata, "Extraction of viewpoints for eliciting customer's requirements based on analysis of specification change records," APSEC, pp.33–40, 2007.

[16] P. Arkley and S. Riddle, "Tailoring traceability information to business needs," Proc. 14th IEEE International Requirements Engineering Conference (RE'06), pp.239–244, IEEE, 2006.

[17] K. Sankar R and R. Venkat, "Total requirements control at every stage of product development," Proc. 15th IEEE International Requirements Engineering Conference, pp.337–342, IEEE, 2007.

[18] F. Houdek and K. Pohl, "Analyzing requirements engineering processes: A case study," Proc. 11th International Workshop on Database and Expert Systems Applications (DEXA'00), pp.983–987, IEEE, 2000.

Takako Nakatani graduated from Tokyo University of Science, she worked for several software development companies. She received her Ph.D from the University of Tokyo in 1998. She has been a board member of S-Lagoon since 1995, and an associate professor at the Graduate School of Business Sciences, University of Tsukuba since 2006. Her interest area is software engineering, requirements engineering, and modeling techniques. She is a member of ISS, IPSJ, IEEE CS, ACM, and the Society of Project Management.

Shouzo Hori graduated from the Kyushu Institute of Technology Information Engineering in 1980, he began working for the Yaskawa Information Systems Corporation. He has performed management duties and undertaken embedded software development for 20 years. He has been engaged in the process improvement of organizations using ISO9001 and/or CMM since 1998. His interest area is software engineering, requirements engineering, and project management.

Naoyasu Ubayashi is a professor at Kyushu University. He received his Ph.D. from the University of Tokyo in 1999. He is a member of ACM SIGPLAN, IEEE CS, and IPSJ. His current research interests include aspect-oriented programming, extensible languages, and model-driven development.

Keiichi Katamine received the Ph.D degree in engineering from Kyushu Institute of Technology, Japan. He is now an assistant professor of computer science and systems engineering at Kyushu Institute of Technology. He is a PSP instructor of SEI of CMU, and a trainer of S-DBR and CCPM of Goldratt School. His research interests include software engineering, software and knowledge modeling techniques, and project management. He is a member of the Information Processing Society of Japan and the Society of Project Management.

Masaaki Hashimoto graduated from Kyushu University, he worked for NTT (Nippon Telegraph and Telephone Corporation) for 23 years and for Kyushu Institute of Technology for 16 years. He received Doctorate Degree of Engineering from Kyushu University in 1987. He has been a professor of Kyushu Institute of Technology since 1993. His interest area is software engineering, modeling technique, project management, and TOC (Theory of Constraint). He is an instructor of PSP and a trainer of TOC.