Tracking Requirements Evolution by Using Issue Tickets: A Case Study of a Document Management and Approval System

Shinobu Saito
Research and Development Headquarters
NTT DATA CORPORATION
Tokyo, Japan
saitousnb@nttdata.co.jp

Yukako Iimura
Software Innovation Center
NTT CORPORATION
Tokyo, Japan
iimura.yukako@lab.ntt.co.jp

Kenji Takahashi
NTT Innovation Institute, Inc.
San Mateo, CA, USA
kt@ntti3.com

Aaron K. Massey, Annie I. Antón
Georgia Institute of Technology
Atlanta, GA, USA
{akmassey, aianton}@cc.gatech.edu

ABSTRACT
Requirements evolve throughout the software life-cycle. When requirements change, requirements engineers must determine what software artifacts could be affected. The history of and rationale for requirements evolution provides engineers some information about artifact dependencies for impact analysis. In this paper, we discuss a case study of requirements evolution for a large-scale system governed by Japanese laws and regulations. We track requirements evolution using issue tickets created in response to stakeholder requests. We provide rules to identify requirements evolution events (e.g. refine, decompose, and replace) from combinations of operations (e.g. add, change, and delete) specified in the issue tickets. We propose a Requirements Evolution Chart (REC) to visually represent requirements evolution as a series of events over time, and implement tool support to generate a REC from a series of issue tickets using our rules to identify requirements evolution events. We found that the REC supports impact analysis and compliance efforts.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications

General Terms
Management, Measurement, Documentation, Legal Aspects.

Keywords
Requirements Evolution; Issue Tickets; Impact Analysis; Large Information Systems; System Compliance

1. INTRODUCTION
Requirements evolution is a fundamental challenge in software engineering because requirements evolve throughout the software lifecycle, increasing the costs of software development [5]. Managing requirements evolution is particularly challenging in large and complex systems deployed in legally regulated environments because requirements engineers are responsible for ensuring that the evolving requirements respond to stakeholders' change requests while also complying with laws and regulations.

The history of and the rationale for requirements changes provides requirements engineers important context to understand dependencies between software artifacts. This context is particularly valuable to requirements engineers brought onto the software development team after the original requirements artifacts were written. Engineers new to a project must still consider the impact that changes impose on specific requirements artifacts. Understanding requirements evolution also helps engineers assess the impact of potential requirements changes on specific software artifacts.

Issue tracking is commonly used to manage requirements changes. Many open source and commercial products use issue tracking tools for this purpose [9, 15, 17]. Whenever a stakeholder requests a change to the requirements, a requirements engineer creates an issue ticket. The requirements engineer then updates the relevant requirements artifacts to address the change request. In our study, this update specifies one or more operations (e.g., add, change, and delete) for the affected artifacts. For each issue ticket, the requirements engineer also records a rationale for the change. After the stakeholder reviews and authorizes the actions taken, the issue ticket is closed and time-stamped. Collectively, issue tickets contain valuable information for understanding the history of and the rationale for requirements evolution.

In theory, issue tracking can completely record all changes to requirements. In practice, we have not found this to be case. It is unusual for requirements engineers to accurately record all changes that take place during requirements analysis. Requirements engineers may fail to record issue tickets accurately because they are too busy or because they have too many other tasks. These "unrecorded" issue tickets are stored only in the memory of the stakeholders. Identifying unrecorded changes in a series of issue tickets is challenging, even for the requirements engineers who created the issue tickets. Just reading through all the issue tickets for a large system may take a non-trivial amount of time. Even experienced requirements engineers may fail to identify unrecorded requirements evolution events in a set of issue tickets. As a result, the impact of these changes on the software system may be overlooked or underestimated.

Our case study examines the commercial development of a large-scale document management and approval system governed by Japanese laws and regulations that uses issue tracking to manage requirements changes. The project is managed and operated by one of the NTT [14] Group companies. The system supports government approval processes in accordance with Japanese laws and regulations. Due to the proprietary nature of the system, we refer to this system as the DMAS (Document Management and Approval System) throughout the remainder of the paper. There are over 100 requirements artifacts that correspond to each of 70

ICSE Companion'14, May 31 – June 7, 2014, Hyderabad, India
ACM 978-1-4503-2768-8/14/05
http://dx.doi.org/10.1145/2591062.2591194

business processes supported by the DMAS, for a total of approximately 7,000 requirements artifacts.

For this study, we examine requirements evolution using the issue tickets created by requirements engineers in response to change requests. We create rules to identify the requirement evolution events (e.g., refine, decompose, and replace) from combinations of operations specified in the issue tickets (e.g., add, change, and delete). We also propose a Requirements Evolution Chart (REC) based on the mapping of issue ticket operations to requirements evolution events. The REC provides a visual representation of requirements evolution events over time.

In this paper, we seek to answer the following research question by means of practical evaluation within the case study:

Does the REC enable requirements engineers to identify requirements evolution events (e.g., refine, decompose, and replace) that were previously overlooked?

The remainder of this paper is organized as follows: Section 2 provides an overview of the DMAS. Section 3 describes related work with an emphasis on requirements evolution. Section 4 defines our set of requirements evolution events and presents our seven rules for mapping issue ticket operations to requirements evolution events. Section 4 also defines the REC. Section 5 describes a REC generation tool based on a software implementation of the mapping rules presented in section 4. Section 6 presents our analysis procedures and results from our case study. Section 7 describes the limitations of the case study. Section 8 discusses the implications of our findings. Section 9 summarizes the paper and presents our plans for future work.

2. DMAS OVERVIEW

The system for this study is a very large (tens of millions of SLOC) document management and approval system (DMAS) governed by Japanese laws and regulations. The DMAS supports document approvals similar to those needed for building or construction permits or drug approvals in the United States. The DMAS supports 13 high-level business process groups (BPGs), each of which is responsible for a different part of the approval process for different types of submissions. In total, there are over 70 business processes allocated to these 13 BPGs. Examples of business process activities include document filings, approvals, rejections, reviews, and appeals. On average, the DMAS supports over 1,000 daily users and nearly 500,000 documents are submitted each year, with each submission triggering a complex review and approval process. Developmental delays as little as one day might cost approximately $150,000 to $250,000. Although the DMAS is a unique system, it is similar to other large information systems that must comply with evolving laws and regulations.

One of the characteristics of this system that makes tracking requirements evolution so challenging is the crosscutting nature of the business activities that occur in each of the 13 BPGs. Some types of submissions trigger processing (e.g., reviews or approvals) in multiple BPGs, whereas others may only require processing in one BPG. In addition, each of the 50 individual requirements engineers assigned to this project is responsible for at most three of the 70 business processes. Any type of submission that triggers processing in more than two business processes requires coordination with at least one (and possibly several) additional requirements engineer to accurately assess the impact of these requirements changes. Furthermore, manually searching the entire set of artifacts to detect those affected may take as long as two or three days even for experienced requirements engineers.

The DMAS is a six-year development effort with two years devoted to requirements definition, two years to architectural design, and three years to implementation and testing. Note that there is some overlap between the last year of the requirements definition phase and the first year of the architectural design phase.

This project was prompted by the necessity of compliance with new laws and regulations. The need to reengineer the DMAS for legal compliance also afforded an opportunity for a general BPR (Business Process Reengineering) effort to consolidate a large set of databases and migrate the legacy mainframe system to a new client-server based system.

2.1 Requirements Artifacts

The DMAS business process is represented by a collection of business flows. Figure 1 shows an example business flow, including the relationship between the use cases and the decision table. Thus, we now define three types of requirements artifacts:

A *use case* describes a sequence of events performed by actors using natural language (e.g., input and output, pre- and post-conditions, normal and exceptional scenarios).

A *decision table* represents conditions at the end of a given use case that must exist for determining a plausible next use case in the business flow sequence.

A *business flow* consists of use cases and decision tables.

The business flow in Figure 1 comprises six use cases (Use Cases #1 - 6), and one decision table (Decision 1).

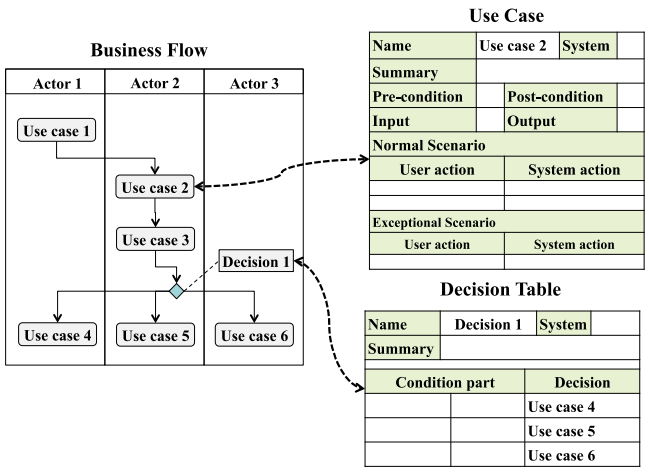


Figure 1. Requirements artifacts for a business flow.

2.2 Requirements Engineering Team

The DMAS requirements engineering team consists of approximately 50 requirements engineers, one project manager, and five middle managers. Each middle manager manages 10 requirements engineers on average. Each requirements engineer is in charge of 2-3 business processes on average. Thus, there is a many-to-many relationship between requirements engineers and business processes, which complicates requirements evolution management.

Once requirements artifacts are approved by the customer(s), the design and implementation will be contracted to another team. The planned maximum size for the implementation and testing

team is approximately 1,000 software engineers. Most original members of the requirements engineering team will be contracted out after the requirements definition phase.

2.3 Stakeholder Review Meetings

The initial input to the entire overall system development effort was a set of requirements extracted from the legacy system's specification and managed using an Excel spreadsheet. This original specification was incomplete, and we conducted stakeholder review meetings to ensure requirements coverage for the new system.

A separate requirements analysis effort was conducted for each business process. The stakeholders for each business process included members of the NTT team and approximately five members of the customers' IT staff and end users familiar with the legacy system the DMAS will replace. A few days prior to each requirements analysis meeting, an NTT middle manager proposes an agenda, which must be approved by a customer representative. Using the agenda as a basis, the customer representative selects stakeholders with appropriate or relevant experience with the aspects of the system relevant to the agenda for participation in each meeting. Two requirements review meetings were held each week. Requirements artifacts (e.g. business flows, use cases, and decision tables) were updated after each review meeting on the basis of discussions held and suggestions made during the meetings.

Because the updates are not made during the actual review meetings, each meeting began with stakeholders reviewing, correcting, and approving the minutes from the previous meeting and any issue tickets created by the engineers as a response to the previous meeting. This typically took approximately 30 minutes. During each meeting, the attendees actively reference and review all the requirements artifacts.

The stakeholders see the requirements artifacts for the first time when they arrive at the meeting. There is no a priori review. Each meeting lasts about two hours. There is no break during the meeting because they were conducted using Fagan-style [3] software inspections, which are normally limited to two hours.

2.4 Issue Tickets

During a meeting, stakeholders often request new requirements, which are then recorded on new issue tickets by the requirements engineers. These tickets reflect the new issues raised as well as the change requests made by the stakeholders during the meeting. Table 1 shows an Issue Ticket Template. Each issue ticket includes a ticket ID, a change request, rationale, update action (e.g., updated artifacts, artifact types, operation types), issue date, and close date. The operation types are Add, Change, and Delete. The artifact types are UC and DT, which are abbreviations for "Use Case" and "Decision Table".

Table 1. Issue ticket template.

Ticket ID	Change Request	Rationale	Update action			Issue date	Close date
			Updated artifact	Artifact type	Operation type		
T1	foo	bar	A	UC	Delete	2013/4/1	2013/4/8
			C	UC	Add		
			D	UC	Add		
			E	DT	Change		
…	…	…	…	…	…	…	…

After each stakeholder review meeting, requirements engineers document change requests by creating issue tickets, and determine whether the issue can or will be addressed by the requirements engineering team. The team then distributes the meeting minutes and issue tickets to the stakeholders, as well as details about the issues upon which agreement was reached. As previously mentioned, these minutes and issues tickets must then be approved at the beginning of the next review meeting.

The responsible middle manager in the requirements engineering team must carefully manage any new requirements surfaced during the meetings. For example, stakeholders discussing part of the system they have not previously examined bring a new perspective and tend to generate more requirements changes than stakeholders who have previously examined that part of the system. Due to the size of this system, the middle managers must actively manage and monitor requirements evolution to ensure that there are no conflicts between new or changed requirements and existing, unchanged requirements. In particular, they must be aware of regulatory requirements and ensure the system remains compliant. The middle managers must also attempt to minimize superfluous or "bells and whistles" requirements.

3. RELATED WORK

3.1 Software Evolution

Lehman [12] proposed three different software types: S-type, P-type, and E-type. S-type software addresses problems stated formally and completely. P-type software is for "problem-solving"; it finds solutions for addressing imprecise problems of the real world. Because the real world changes and the problems also change, P-type software is likely to evolve continuously. E-type software is embedded in the real world and becomes part of it; it must evolve to remain satisfactory to stakeholders. Herein, we focus on requirements evolution exclusively within the context of P-type software. Specifically, we are interested in ways to manage high-level requirements as they change in response to change requests from stakeholders, and in ways to trace requirements as they evolve in order to identify the subsequent impact of changing requirements.

Lehman and Ramil [13] primarily focus on "program evolution". For example, they observed and analyzed the trends of size growth in program modules by using release and revision dates. In contrast, we focus on requirements evolution using issue tickets during the early stages of software development.

3.2 Traceability and Impact Analysis

Settimi et al. [16] trace requirements to UML artifacts, source codes, and test cases for supporting software evolution. Their Information Retrieval method aids in understanding the change impact scope. However, their approach requires textually rich artifacts for improving the impact analysis accuracy. Because our approach relies on a limited set of operation types in issue tickets, it is not dependent upon the richness of textual description in the artifacts used.

Von Knethen [11] uses a fine-grained trace model to evaluate impact analysis in system requirements changes. This trace model determines documentation entity types (e.g., use case and functional requirement) and relationships (e.g., dependency, refinement) to be traced. The trace model also includes constraints on these relationships. They presented a set of process descriptions that determine how to establish traceability and how to analyze the impact of changes. These processes are semi-automated with tool support. The trace model is tailored for

embedded control systems and smaller product applications that use UML models. In contrast, our work focuses on large, complex information systems, and our artifacts are not limited to UML models.

3.3 Managing Requirements Evolution

Requirements evolution is inevitable in any software development effort. Within the context of systems that must comply with laws and regulations, managing these changes while maintaining traceability is critical because of the need to ensure due diligence while adhering to the expected standard of care.

Jones [10] introduced the term "requirements creep" to discuss the inevitable evolution of requirements. Several emerging technologies (e.g., prototypes, requirements inspections, and change-control boards) were proposed as ways to clarify early requirements and minimize the disruptive effects and costs of changing requirements later. Similarly, we leverage the history of and rationale for requirements evolution early on to allow for efficient impact analysis when stakeholders submit requirements change requests.

Carter et al. [1] proposed a tool-supported method to manage requirements creep. Their evolutionary prototyping model helps engineers prompt stakeholders to submit and clarify requests. In contrast, our approach relies only on the history and the rationale associated with requirements evolution as expressed in issue tickets. Our approach supports both stakeholders and engineers during weekly meetings by clarifying the impact of changes on requirements artifacts.

Harker et al. [5] noted the importance of considering requirements change classifications in order to manage requirements evolution. Specifically, they distinguished between stable and changing requirements (mutable, emergent, consequential, adaptive, and migration) to assess their impact scope. In the case study reported herein, most of the changing requirements were mutable and emergent requirements. The mutable requirements originated from laws and regulations, whereas the emergent requirements originated with customers attending the review meeting.

Wnuk [18] proposed a technique used to visualize a large number of requirements. Their technique, called Feature Transition Charts (FTC), enables visualization of the scope dynamics within and across multiple projects (e.g., product line projects). The technique provides a comprehensive overview of the timing and magnitude of feature transitions among multiple projects. The scope of each project is maintained in a feature list. In order to find feature transitions, they search feature identifiers in the feature lists of the projects for exactly matching names. In contrast, we focus on an issue tickets list for a single large software system.

When stakeholders request new requirements, requirements engineers make structural changes to the corresponding artifacts. Cleland-Huang et al. [7] defined seven evolutionary events of change: Create, Inactivate, Modify, Merge, Refine, Decompose, and Replace. In the next section, we discuss our adoption of these seven evolutionary events.

4. EVOLUTION TRACKING TECHNIQUE

4.1 Evolutionary Events

As mentioned above, we used Cleland-Huang's seven events [7] to record the kind of requirements evolution that takes place. Figure 2 shows the events resulting from requirements evolution. Requirements artifacts are represented by the capital letters A, B, and C. The symbol → represents a structural change from the artifacts on the left hand side to those on the right hand side.

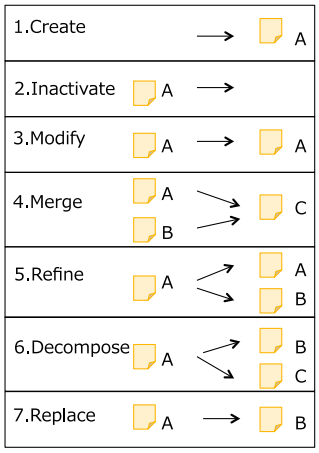


Figure 2. Evolutionary events.

Brief descriptions of the seven evolutionary events follow. "Create" produces a new artifact. "Inactivate" means an artifact is marked as deleted. "Modify" changes the value of an attribute in a given artifact. "Merge" combines two or more artifacts into a new artifact. "Refine" adds a new additional artifact to, for example, fine-tune an existing original artifact. "Decompose" takes an existing artifact and separates it into two or more artifacts. "Replace" substitutes one artifact with another.

4.2 Guidance for Recording Issue Tickets

Requirements engineers must record issue tickets according to a set of guidelines to ensure consistency. The guidelines are as follows:

- Changes affecting one artifact:

  If a single change request affects only a single artifact (e.g. UC, DT), it shall be recorded in one issue ticket.

- Changes affecting more than one artifact:

  - A single change affecting more than one artifact:

    If a single change request affects more than one artifact, it shall be recorded in one issue ticket. Additional change requests affecting the same set of artifacts shall also be recorded in this issue ticket.

  - More than one change affecting more than one artifact:

    If change requests, each of which affect multiple artifacts, do not affect exactly the same set of artifacts, they shall be recorded in separate issue tickets.

Figure 3 portrays a meta-model that represents the relationship between information about an issue ticket and evolutionary events. As described in Subsection 2.4, one issue ticket includes one or more update actions. An update action contains information about

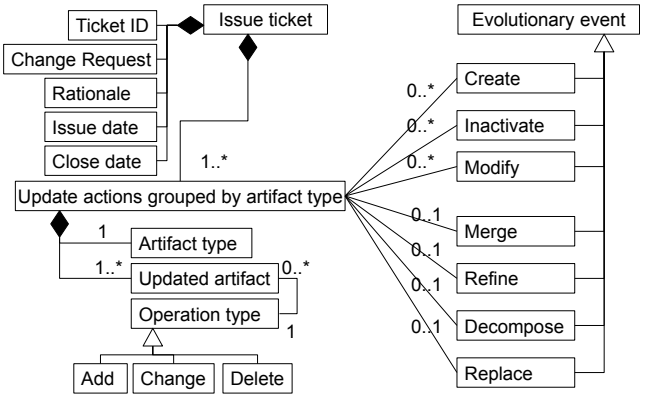


Figure 3. Meta-model of issue tickets and requirements evolution events.

artifact type, updated artifact, and operation type. In our approach, a set of update actions grouped by artifact type in an issue ticket is mapped to one of the evolutionary events. For example, when an issue ticket records two sets of update actions grouped by two artifact types (e.g. use case and decision table), each set of update actions can be mapped separately to requirements evolution events.

4.3 Mapping Rules

We define seven rules that map a combination of operations in an issue ticket to an evolutionary event. Table 2 is a comprehensive list of the seven mapping rules. It shows the mapping relations between evolutionary events and combinations of operations recorded in the issue ticket. Names of the evolutionary event appear in the leftmost column. The operation types (Add, Change, and Delete) appear in the three rightmost columns. The number of artifacts updated ("One", "One or more", and "Two or more") is noted in relevant cells. We can recognize requirements evolution using the issue tickets from these rules as described below:

Table 2. Seven mapping rules.

Evolutionary event	Combination of operations		
	Add	Change	Delete
1. Create	One or more		
2. Inactivate			One or more
3. Modify		One or more	
4. Merge	One		Two or more
5. Refine	One or more	One	
6. Decompose	Two or more		One
7. Replace	One		One

4.3.1 *Create and Inactivate Events*

When an issue ticket contains information that one artifact was added or deleted, we recognize that the artifact was newly created, inactivated, respectively. Figure 4 portrays the identification of Create and Inactivate events from its corresponding issue tickets. An issue ticket for which an Add operation is recorded is mapped to a Create event. In the same way, an issue ticket in which only a Delete operation is recorded is mapped to an Inactivate event. The mapping rules for Create and Inactivate events are formally represented as below:

(No. of Add operations ≥ 1) & (No. of Change operations = 0) & (No. of Delete operations = 0) → Create event

(No. of Add operations = 0) & (No. of Change operations = 0) & (No. of Delete operations ≥ 1) → Inactivate event

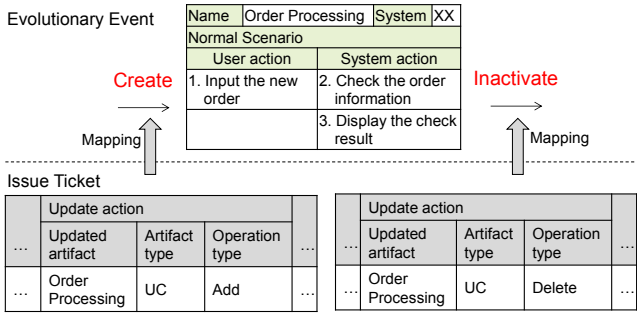


Figure 4. Create and Inactivate events and the corresponding issue tickets.

4.3.2 *Modify Event*

When an issue ticket contains information that one artifact was changed, we recognize that the value of the attribute in the artifact

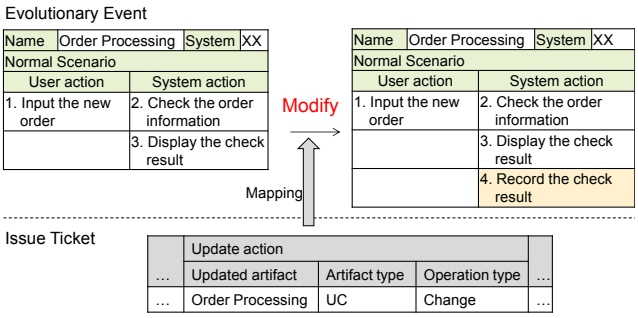


Figure 5. Modify event and the corresponding issue ticket.

was modified. An issue ticket in which only a Change operation is recorded is mapped to a Modify event. Figure 5 depicts the identification of a Modify event from its the corresponding issue ticket. In this figure, the issue ticket contains information that the use case was changed. Thus, the Modify event (i.e., scenario in the use case changed) is identified from this combination of operations in the issue ticket. The mapping rule for a Modify event is formally represented as below:

(No. of Add operations ≥ 0) & (No. of Change operations ≥ 1) & (No. of Delete operations = 0) → Modify event

4.3.3 *Merge Event*

In an issue ticket where two or more artifacts were deleted and exactly one artifact was added, we recognize that the deleted artifacts were merged into a newly added artifact. Figure 6 shows a Merge event and its corresponding issue ticket. In this figure, the issue ticket indicates that two use cases were deleted and new use case was added (i.e., two use cases were merged into one use case). The Merge event is identified from this combination of operations in the issue ticket. The mapping rule for a Merge event is formally represented as below:

(No. of Add operations = 1) & (No. of Change operations = 0) &

(No. of Delete operations ≥ 2) → Merge event

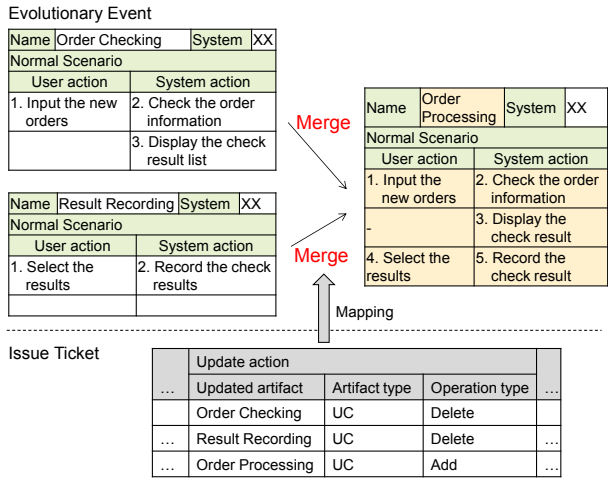


Figure 6. Merge event and the corresponding issue ticket.

4.3.4 *Refine Event*

An issue ticket containing one changed artifact with one or more added artifacts is recognized as a Refine event. Figure 7 shows a Refine event and its corresponding issue ticket. In this issue ticket, the existing use case was changed and new use case was added.

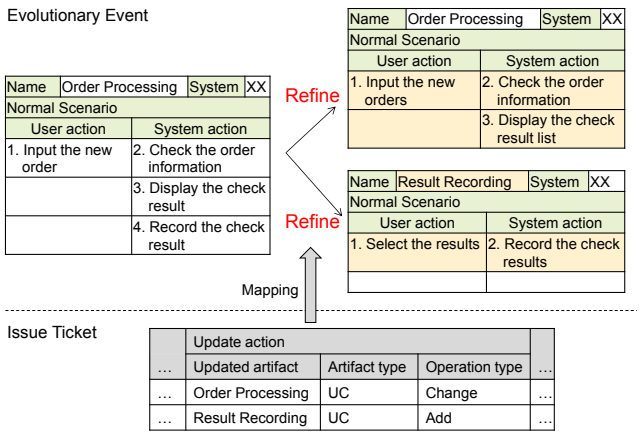


Figure 7. Refine event and the corresponding issue ticket.

The Refine event is identified from this combination of the operations in the issue ticket. The mapping rule for a Refine event is formally represented as below:

(No. of Add operations ≥ 1) & (No. of Change operations = 1) & (No. of Delete operations = 0) → Refine event

4.3.5 Decompose Event

An issue ticket containing a deleted artifact with two or more newly added artifacts is recognized as a Decompose event; the newly added artifacts actually decompose the deleted artifact. Figure 8 shows a Decompose event and its corresponding issue ticket. In this figure, the issue ticket contains information that

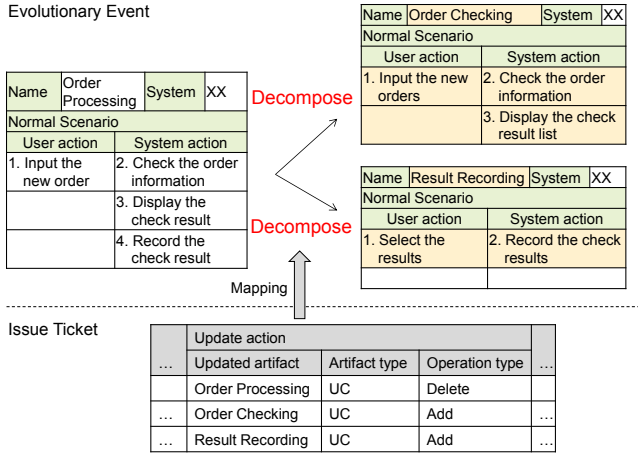


Figure 8. Decompose event and the corresponding issue ticket.

artifact the existing use case was deleted and two use cases were added (i.e., one use case was divided into two use cases). The Decompose event is identified from this combination of the operations in the issue ticket. The mapping rule for a Decompose event is formally represented as below:

(No. of Add operations ≥ 2) & (No. of Change operations = 0) & (No. of Delete operations =1) → Decompose event

4.3.6 Replace Event

An issue ticket in which one artifact was deleted and one artifact was added is recognized as a Replace event. Figure 9 shows a Replace event and its corresponding issue ticket. In this issue ticket, the use case was deleted and new use case was added. The Replace event is identified from this combination of the operations in the issue ticket. The mapping rule for a Replace event is formally represented as below:

(No. of Add operations = 1) & (No. of Change operations = 0) &

(No. of Delete operations = 1) → Replace event

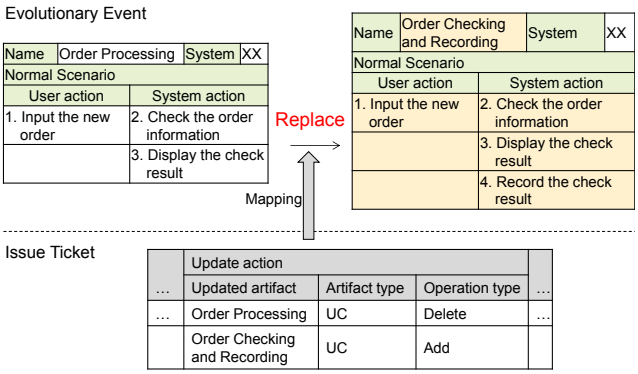
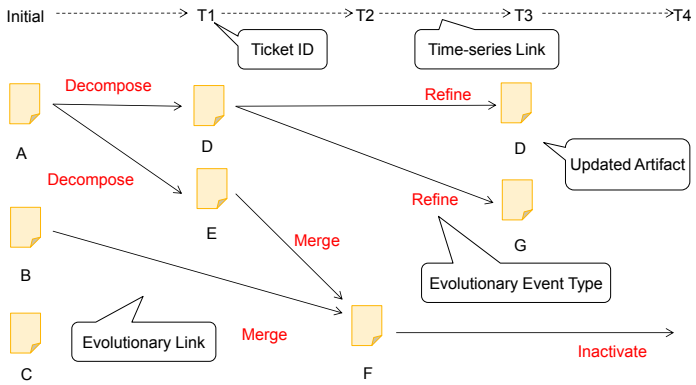


Figure 9. Replace event and the corresponding issue ticket.

4.4 REC: Requirements Evolution Chart

We now introduce a Requirements Evolution Chart (REC) to visualize a time series of events of requirements evolution. Figure 10 shows a sample REC with corresponding issue tickets. The REC includes four evolutionary events: Decompose, Merge, Refine, and Inactivate. The corresponding four issue tickets appear on the right side; the four columns (Ticket ID, Updated artifact, Artifact type, and Operation type) are taken from the issue ticket template. The ticket IDs of the issue tickets are T1, T2, T3, and T4. In this figure, artifacts A, B, and C are found in the initial condition. The dotted lines are labeled time-series links;



Ticket ID	Update action		
	Updated Artifact	Artifact type	Operation type
T1	A	UC	Delete
	D	UC	Add
	E	UC	Add
T2	B	UC	Delete
	E	UC	Delete
	F	UC	Add
T3	D	UC	Change
	G	UC	Add
T4	F	UC	Delete

Figure 10. REC and corresponding issue tickets.

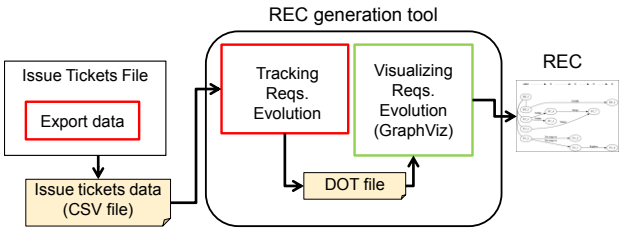


Figure 11. Overview of REC generation tool.

each represents a sequence of the issue tickets. The initial condition is the starting point and it leads directly to the first issue ticket (T1). From there, the second time-series link starts at T1 and ends at T2, which is the ticket ID of the second issue ticket. In this way, both ends of each link represent consecutive ticket IDs of two issue tickets. The solid lines are labeled evolutionary links; each represents a change transition of an artifact in the requirements evolution. In this way, these links and labels visually represent a time series of requirements evolution events. On the other hand, as shown in Figure 10, artifact C is not connected to any evolutionary links. The reason is that any issue tickets for artifact C are not recorded, as shown in the issue tickets listed in the figure.

The REC seeks to help requirements engineers understand requirements evolution. Issue tickets are mapped to evolutionary events in the REC. They also contain information about change requests made by stakeholders and the rationale for the update actions to artifacts for addressing a change request.

5. SOFTWARE IMPLEMENTATION

We implemented a REC generation tool to demonstrate the ability to track and visualize requirements evolution using issue tickets. Figure 11 provides an overview of the tool. The tracking function automatically applies the mapping rules described in section 4.4 to identify evolutionary events using issue tickets. The visualization function generates the REC image based upon the results of the rule mapping. The REC tool uses an open source software package, GraphViz [4], to generate the visualizations.

The tool takes as its input a comma-separated values (CSV) file that includes issue tickets data. The upper side of Figure 12 shows a screen image of the file; as shown, it holds information for four issue tickets. The ticket IDs are numbered from t1 to t4. The "Artifact type" column shows two types: UC and DT, which are abbreviations for "Use Case" and "Decision Table." By applying the mapping rules, the function for tracking requirements evolution creates a DOT file that describes the graph. From the created DOT file, GraphViz generates an image of the REC. The lower half of Figure 12 shows an output image of the REC generated from the issue tickets file shown in the top half of the figure. It contains five artifacts (i.e., UC_# 1 to 3 and DT_# 1 to 2) in the initial condition. From these four issue tickets, the tool was able to track five evolutionary events (e.g., Refine, Decompose, Merge, Modify, and Replace) and visualize the requirements evolution history, using the sequence of changes described by the issue tickets.

This tool can support changes affecting different artifact types in one issue ticket. As shown in Figure 12, issue ticket t4 includes update actions regarding two artifact types, UC and DT. In the UC part, artifact UC_5 was deleted and artifact UC_8 was added. The tool recognizes this combination of operations as a Replace event by executing the mapping rule for a Replace event. Moreover, artifact DT_1 was changed in the DT part. The tool also

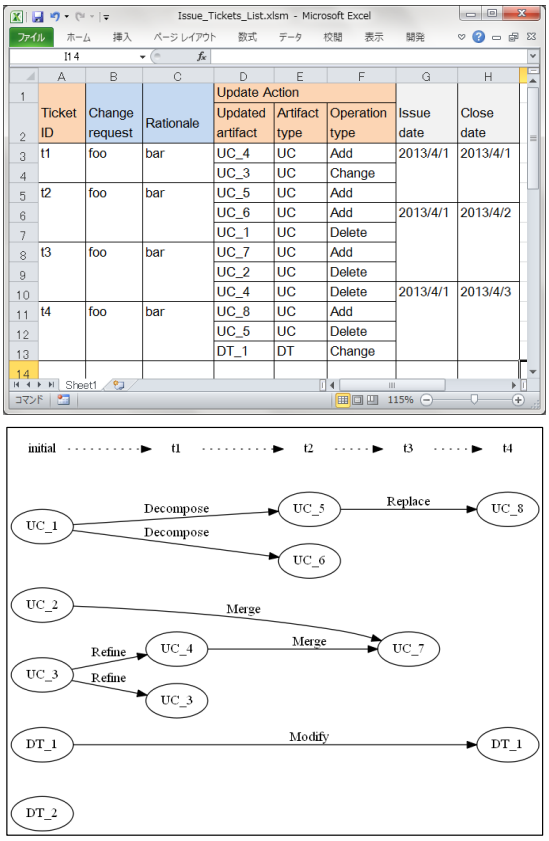


Figure 12. Screenshot of REC and issue tickets file.

recognizes this operation as a Modify event by executing the mapping rule for a Modify event. As shown in Figure 12, Replace and Modify events are visualized at the lower part of ticket t4. In the issue tickets list of the figure, any issue tickets on artifact DT_2 are not recorded. The REC shows that artifact DT_2 is not connected to any evolutionary links.

6. CASE STUDY

To answer the research question given in Section 1, we used the DMAS to conduct a case study.

6.1 Data Collection and Participants

In this case study, the requirements definition phase for one business process continued for nine weeks: three weeks of initial creation of requirements artifacts and six weeks of review with stakeholders. The stakeholders started by creating 16 requirements specifications, in which BPR and compliance needs were written in natural language. Using these specifications as a basis, two requirements engineers then created a set of 79 requirements artifacts. By the time the review was completed, the total number of artifacts had increased to 109 and a total of 61 issue tickets were created. These tickets included 44 use case tickets, 14 decision table tickets, and 3 tickets for both use cases and decision tables. The two requirements engineers who created the requirements were the participants in the case study. Both were senior-level engineers with over 10 years of experience. They were also primarily responsible for the business process of DMAS.

6.2 Analysis Procedure

The analysis procedure contains four steps, as described by Figure 13. In the first step, we collect an issue tickets file and extract issue tickets data (e.g. ticket ID, artifact types, updated artifacts,

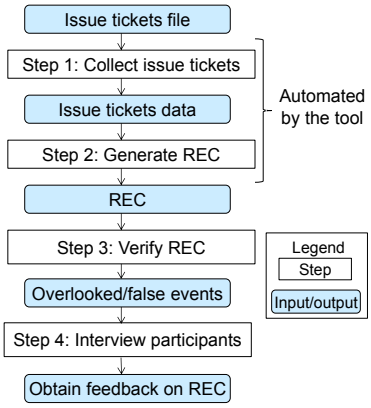


Figure 13. Analysis Procedure.

operation types) to generate the REC. In the second step, the REC is generated from the issue tickets data. Steps 1 and 2 are conducted automatically using the tool described in Section 5. In step 3, the two requirements engineers verify that the generated REC is accurate. During this step, if they identify overlooked or false events in the REC, they correct the corresponding issue tickets or create new issue tickets to resolve the discrepancy. In the final step, the engineers are individually interviewed on the effectiveness of the REC for identifying overlooked events.

6.3 Results

6.3.1 Identifying Overlooked/False Events

In steps 1 and 2, we collected an issue tickets file and extracted issue tickets data. From the issue tickets data we generated the REC using the tool. The REC includes 64 evolutionary events: 47 (=44+3) use case events and 17 (=14+3) decision table events. In step 3, the two requirements engineers verified the generated REC for approximately one hour.

Table 3 shows the verification results by evolutionary event. It shows (I) the number of events generated from issue tickets, (II) the number of overlooked events, and (III) the number of false events. By referring to the REC, the requirements engineers identified 48 overlooked events and three false events. Next, they created 48 new issue tickets corresponding to the overlooked events, and removed the three false issue tickets. As a result, they compiled a total of 109 (=64+48-3) events. The REC enabled the requirements engineers to recover the overlooked events. During the case study, the number of "corrected" issue tickets increased by 78.9% (=48/ (64-3)) – a recovery rate of roughly 80%. As described in Section 2, it may take two or three days to search the entire range of requirements artifacts in order to detect affected artifacts, even if the searching is done by experienced requirements engineers.

Table 3. Verification Results.

	(I) Generated event	Identified event	
		(II) Overlooked	(III) False
1. Create	10	5	2
2. Inactivate	1	3	1
3. Modify	29	21	0
4. Merge	0	0	0
5. Refine	8	9	0
6. Decompose	2	1	0
7. Replace	14	9	0
Total	64	48	3

6.3.2 Feedback from Requirements Engineers

We asked the participants the following two questions:

- How did you identify overlooked events by using the REC?
- If overlooked or false events were not identified, what kinds of increased project risks would occur in the future?

Each engineer was interviewed for approximately half an hour. The interviews focused on the top three overlooked events (e.g., Modify, Refine, and Replace). As shown in Table 3, these events covered approximately 80% of all overlooked events.

6.3.2.1 Overlooked Modify Event

Figure 14 shows an image of the overlooked Modify event in the REC. The dotted box on the right side of the figure is not described in the REC because the corresponding issue ticket on the Modify event of artifact X was overlooked and therefore not recorded.

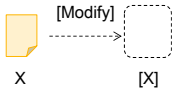


Figure 14. Overlooked Modify event.

How did they identify overlooked events?

Using the REC, the requirements engineers recognized that artifact X was not changed. The REC prompted them to reconsider whether artifact X had actually been changed. The engineers recalled past events related to artifact X. This enabled them to identify that artifact X had indeed been changed. They then created a corresponding issue ticket recording the Modify event of artifact X.

What kinds of increased project risks would occur in the future?

As mentioned in Section 2, the DMAS must comply with relevant laws and regulations. Therefore, during the case study the requirements engineering team updated a number of requirements artifacts specifically to address revisions of laws and regulations. However, most of the original team members were contracted out after the requirements definition phase in the project (see Subsection 2.2). Therefore, if the laws and regulations are revised again in the future, the new team members will have to detect artifacts affected by the new revisions using only information provided by the issue tickets recorded by the original requirements engineers. For example, consider the overlooked Modify event of artifact X (shown in Figure 14) that occurred due to a revision in regulation A. If this regulation were revised again and a corresponding issue ticket was not recorded, identifying that artifact X had been changed previously as a result of regulation A challenging and time-consuming. If the issue ticket remains unrecorded, the risk of non-compliance will increase.

6.3.2.2 Overlooked Refine Event

Figure 15 shows an overlooked Refine event in the REC. Once again, the dotted boxes on the right side set of the figure are not described in the REC because the corresponding issue ticket on the refine event (X → X + Y) was not recorded.

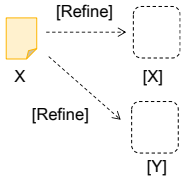


Figure 15. Overlooked Refine event.

How did the engineers identify overlooked events?

By referring to the REC, requirements engineers recognized that artifact Y was not described in the REC although they created it as a part of their requirements analysis. The REC allowed them to recognize that they did not record the issue ticket for artifact Y. This enabled them to identify that artifact X was changed and artifact Y was newly added. They corrected the error in the corresponding issue ticket by recording the refine event for artifacts X and Y.

What kinds of increased project risks would occur in the future?

In the DMAS project, stakeholders requested a number of requirements to support system compliance. In addressing these requirements, the regulatory nature of the existing artifact was

emphasized. As shown in Figure 15, it was often the case that the new artifact Y was created from the existing artifact X. In this situation, the description of artifact Y might be significantly similar to that of artifact X. For example, let us assume that artifact X was related to regulation B. If artifact X needs to be changed due to a regulatory change, requirements engineers should consider whether artifact Y also needs to be changed due to the revision. However, if the corresponding issue ticket of the Refine event is not recorded, determining that artifact Y was previously created from artifact X will be challenging and time-consuming. The requirements engineers may not be able to detect that artifact Y might have also been affected by the change required for artifact X. Once again, risk of system non-compliance will be increased if the issue tickets are not accurate.

6.3.2.3 Overlooked Replace Event

Figure 16 shows an image of an overlooked Replace event in the REC. The dotted box on the right side set of the image is not described in the REC because the corresponding issue ticket on the Replace event (X → Y) was not recorded.

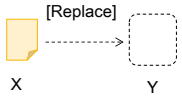


Figure 16. Overlooked Replace event.

How did they identify overlooked events?

By referring to the REC, requirements engineers recognized that artifact X was not changed. They also recognized that artifact Y was not described in the REC although they it was created during their requirements analysis. The REC prompted them to recognize that they did not accurately record issue ticket on artifact X and Y. This enabled them to identify that artifact X was replaced by artifact Y. They corrected the corresponding issue ticket to record the Replace event of artifacts X and Y.

What kinds of increased project risks would occur in the future?

Some of the laws and regulations to which the DMAS must comply have been in effect for over 10 years. They have been revised periodically since going into effect. Often, the name of the relevant artifact was changed as a result of revisions in the laws and regulations. For example, let us assume that artifact X was previously changed by a revision of regulation C, and an issue ticket regarding the event (i.e., a Modify event of artifact X) was recorded. If the regulation is revised in the future, requirements engineers must be able to know that artifact X was previously changed using only the issue tickets. If an issue ticket for the replace event (X → Y) was not recorded, identifying artifact Y as affected by the change to regulation C will be challenging and time-consuming. Errors in the issue tickets will result in increased risk of system non-compliance.

7. CASE STUDY LIMITATIONS

When designing any case study, care should be taken to mitigate threats to validity [19]. We make no causal inferences as a result of our study, so internal validity is not a concern.

External validity is the ability of a case study's findings to generalize to broader populations. A possible threat to external validity is the fact that we only analyzed one project: the DMAS. However, the system is substantially similar to other document approval systems, such as those that manage building or construction permits or drug approvals in the United States. In addition, the requirements engineering team created requirements artifacts that are not domain specific (e.g., business flow, use case and decision table). These types of software artifacts are used widely by other systems. Issue tickets are also used widely by other systems. We believe these facts reinforce the external validity of our case.

Construct validity addresses the degree to which a case study is in accordance with the theoretical concepts used. Three ways to reinforce construct validity are: using multiple sources of reliable evidence, establishing a chain of evidence, and having key informants review draft case study reports. By collecting issue tickets from 13 review meetings over a total of six weeks with two different types of stakeholders (Information Technology department staff members and end users), we used multiple sources for our study. To establish a chain of evidence, we carefully maintained a record of all issue tickets created and the relationship between issue tickets and corresponding requirements artifacts. Finally, other members of NTT reviewed our draft case study report [14].

Reliability is the ability to repeat a study and observe similar results. To reinforce our study's reliability, we developed the REC generation tool. This tool enabled us to conduct steps 1 and 2 of the case study automatically. By using the tool, other researchers and case study participants will be able to rigorously follow the steps of the case study.

8. DISCUSSION

8.1 Acquiring Implicit Knowledge

In the case study, a number of changes were not accurately recorded in the issue tickets. After the requirements definition phase, even when requirements engineers examined the contents of the issue tickets list written in natural language, they rarely noticed that there were "unrecorded" changes. To address this, we generated the REC, which displays the contents of the recorded issue tickets graphically. This representation helped requirements engineers identify which artifacts were not accurately recorded in the issue tickets. As a result, they were able to identify overlooked events and create the corresponding issue tickets with an approximately 80% increase from the time of the completion of the requirements definition phase. By providing the graphical representation of existing explicit knowledge (i.e., recorded issue tickets), the REC supported requirements engineers in their acquisition of implicit knowledge (i.e., unrecorded issue tickets).

8.2 Knowledge Transformation

When requirements engineers leave a project, their expertise and knowledge is sorely missed. This is particularly challenging when the rationale for changes in the requirements is extensive or subtle. In these cases, the REC can serve as a training aid when assimilating new project members. The REC enables understanding of requirements evolution and the history of software artifacts throughout the software project. The REC is a knowledge transfer tool because it documents the provenance of requirements for the newer project members. Generally, in large-scale system development projects, it takes a lot of time for new project members to understand the overall structure of requirements artifacts and the related history of and rationale for the requirements. The REC is a new tool for project members seeking to understand the relationship between evolutionary events and issue tickets and the rationale for these changes.

8.3 Effectiveness of Impact Analysis

When stakeholders request new requirements, requirements engineers usually rely on full-text searches of the artifact repository to identify the affected artifacts. For example, when a regulation is changed, they try to identify the affected artifacts using keyword searches of terms relevant to the regulation.

However, if the original artifact that complied with the regulation has evolved (e.g., through a Decompose, Merge, or Replace event), derivative artifacts that are generated from the original artifact during requirements evolution must also maintain compliance. However, those requirements may not be identifiable simply by searching for keywords from the regulation. Without historical requirements evolution information, engineers may not be able to identify the scope of the derivative artifacts when searching the latest set of artifacts. The REC provides requirements engineers with another way to identify the historical evolution and rationale for changes in system requirements.

8.4 Using the REC at Scale

Visual representations of software systems must scale to be useful. Large software systems may have thousands or tens of thousands of software artifacts. The DMAS has on the order of 8,000 issue tickets in total at a rough estimate, but because the system is decomposed into 70 business processes, each of which has a manageable amount of issue tickets, we were able to use the REC effectively. The REC described herein may not scale to systems that cannot be easily modularized.

9. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an approach to track requirements evolution using issue tickets. We first defined the issue ticket template, which contains specific operations to requirements artifacts for addressing change requests from stakeholders. We then provided seven rules that describe identification of requirements evolution events based on combinations of operations in the issue tickets. By applying these rules we can recognize requirements evolution events using a series of issue tickets. We also defined our Requirements Evolution Chart (REC), which is a graphical representation of requirements evolution, and briefly described an REC generation tool. We evaluated the effectiveness of our approach and tool by conducting a case study within the context of a large-scale document management and approval system development project. Our study offers two important insights. First, our tracking technique can enable requirements engineers to identify overlooked requirements evolution events. Second, our approach helps requirements engineers conduct impact analysis in the context of system compliance. In future work, we plan to conduct an economic analysis to measure the effectiveness of tracking the requirements evolution using the REC.

10. ACKNOWLEDGMENTS

The authors are grateful to T. Maeyama, R. Matsumoto, S. Oyama, M. Fukunaga, T. Shinya, and T. Kusano of NTT DATA for their assistance in the case study.

11. REFERENCES

[1] R.A. Carter, A.I. Antón, A. Dagnino, L. Williams, “Evolving Beyond Requirements Creep: A Risk-Based Evolutionary Prototyping Model”, 5th IEEE Intl. Requirements Engineering Conf., 2001, pp. 94-101.

[2] A. Cockburn, Agile Software Development, Addison Wesley, 2001.

[3] M.E. Fagan,”Advances in Software Inspections”, July 1986, IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, pp.744-751.

[4] Graphviz, http://www.graphviz.org/

[5] S.D.P. Harker, K.D. Eason, J.E. Dobson, “The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering”, IEEE Intl. Symposium on Requirements Engineering, 1993, pp. 266-272.

[6] J. Highsmith and A. Cockburn, Agile Software Development: the Business of Innovation, IEEE Computer, Vol. 34, No. 9, Sep. 2001, pp. 120-122.

[7] J. Cleland-Huang, C. K. Chang, and M. Christensen, “Event-Based Traceability for Managing Evolutionary Change”, IEEE Transactions on Software Engineering, vol. 29, no. 9, 2003, pp. 796-810.

[8] ISO 5806:1984, Information processing -- Specification of single-hit decision tables, ISO, 1998.

[9] JIRA, http://www.atlassian.com/en/software/jira

[10] C. Jones, “Strategies for Managing Requirements Creep”, IEEE Computer, 29(6), 1996, pp. 92-94.

[11] A. Von Knethen, “A Trace Model for System Requirements Changes on Embedded Systems”, 4th Intl. Workshop on Principles of Software Evolution, 2001, pp. 17-26.

[12] M. M. Lehman, “Programs, Life Cycles, and Laws of Software Evolution”. IEEE, vol 68, no 9, 1980.

[13] M. M. Lehman, J. F. Ramil, “Evolution in Software and Related Areas”, 4th Intl. Workshop on Principles of Software Evolution, 2001, pp.1-16.

[14] NTT, http://www.ntt.co.jp/index_e.html/

[15] Redmine, http://www.redmine.org/

[16] R. Settimi, J. Clelena-Huang, and O. Ben Khadra, “Supporting Software Evoution through Dynamically Retrieving Traces to UML Artifacts”, 7th Intl. Workshop on Principles of Software Evolution, 2004, pp. 49- 54.

[17] The Trac Project, http://trac.edgewall.org/

[18] K. Wnuk, B. Regnell, and L. Karlsson, “Feature transition charts for visualization of cross-project scope evolution in large-scale requirements engineering for product lines”, Requirements Engineering Visualization (REV), 2009, 4th Intl. Workshop, pp. 11–20.

[19] R.K. Yin, Case Study Research: Design and Methods, in Applied Social Research Methods Series, Vol. 5, 2003, 3rd ed.