# A Framework for Dependability Consensus Building and In-Operation Assurance*

Yutaka Matsuno† and Shuichiro Yamamoto
*Strategy Office, Information and Communication Headquarters*
*Nagoya University, Japan*
{matsu, yamamotosui}@icts.nagoya-u.ac.jp

**Abstract**

We present a framework for dependability consensus building and in-operation assurance for information systems, and show the current prototype implementation. Today, information systems are ever changing systems: they are always modified and updated to satisfy user's changing requirements and deal with environmental changes. Furthermore, information systems must cope with system failures and continue to be dependable during operational phase. Unfortunately, however, as recently many serious failures in information systems have been reported, it is becoming much difficult to assure the dependability of information systems especially in the operational phases. For this problem, we observe that there should be a framework for adapting to changes and failures that integrates conventional requirement elicitation, risk analysis, and assurance methods during the whole lifecycle of the information system. Our proposed framework consists of a process cycle for consensus building among stakeholders with conventional requirement elicitation methods, risk analysis methods, and *assurance cases*, documents for system assurance recently widely used in safety and other areas. This paper explains the process lifecycle, a case study using the cycle, and a prototype implementation for in-operation assurance.

**Keywords**: Dependability, Assurance Case, Risk Analysis, Service Continuity, Requirement Elicitation

## 1 Introduction

Today, information systems are ever changing systems: they are always modified and updated to satisfy user's changing requirements and deal with environmental changes. Furthermore, information systems must cope with system failures and continue to be dependable during operational phase. Unfortunately, however, as recently many serious failures in information systems have been reported, it is becoming much difficult to assure the dependability of information systems especially in the operational phases. For this problem, we observe that there should be a framework for adapting to changes and failures that integrates conventional requirement elicitation, risk analysis, and assurance methods during the whole lifecycle of the information system.

Our proposed framework consists of a process cycle for consensus building among stakeholders with conventional requirement elicitation methods, risk analysis methods, and *assurance cases*[2], which are documents for system assurance recently widely used in safety and other areas.

*Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, volume: 4, number: 1, pp. 118-134

*This paper is an extended version of the work originally presented at the 2nd IFIP International Workshop on Security and Cognitive Informatics for Homeland Defence (SeCIHD'12), in conjunction with the 6th International Conference on Availability, Reliability, and Security (ARES'12), Prague, Czech Republic, August 2012 [1]. We revised abstract, introduction, figures, and added demonstration detail in Section 4. Other parts are revised and simplified for readability. A part of this work was done while the first author was in Information Technology Center, the University of Tokyo.

†Corresponding author: Project Lecturer, Strategy Office, Information and Communications Headquarters, Nagoya University, Furo-cho, Nagoya 464-8601 Japan, Tel: +81-80-4172-6454

We briefly explain the process cycle in Figure 1. We call the cycle as dependability consensus building cycle. Dependability consensus building cycle consists of the following three phases: 1) requirement

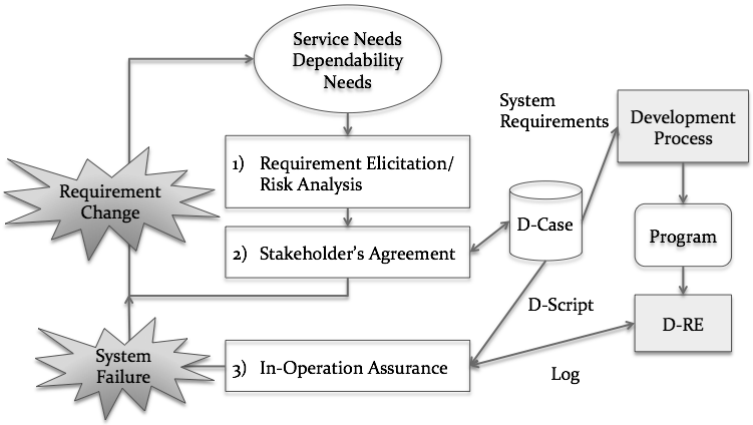


Figure 1: Consensus Building and In-operation Assurance Cycle

elicitation and risk analysis, 2) stakeholder's agreement on requirements, and 3) In-operation assurance using *D-Case*, an extended assurance cases for in-operation assurance.

Assume that a system is newly developed for some service objectives given by stakeholders. In requirement elicitation and risk analysis phase, requirements are elicited from each of the stakeholders who have their needs described in an informal way, and risks which may result in suspending the service are analyzed. In this phase, several requirement elicitation and risk analysis methods are exploited.

In stakeholders's agreement on requirements phase, these elicited requirements are discussed among the stakeholders using D-Case in order to reach agreement. If the stakeholders can not reach agreement on the requirements, some of the requirements will be returned to the first phase for revising. Once the agreement is made, programs are developed according to the D-Case description and other documents such as functional specifications. At the same time, *D-Script*, scripting codes for failure mitigation actions are extracted from the D-Case description. D-Script is used to monitor the system, to collect operational logs, and to respond to failures quickly. When the system needs to be updated due to objectives/environment changes, this cycle is re-started with new requirement being elicited and old requirements being modified.

In-operation assurance phase provides the means to assure the dependability consensus made in the previous phase by monitoring the system and managing requirements online for accountability achievement. A runtime environment called *D-RE*, monitors the system and collects logs of the system as designated by D-Scripts. If some logs show a deviation of some parameters from their thresholds, the corresponding failure responsive actions designated as D-Script codes are activated.

D-RE and D-Scripts have been developed in DEOS (Dependable Embedded Operating System) Project[3], which is a Japanese national research project funded by Japan Science and Technology Agency.

In case a need for a requirements change occurs as a result of the failure responsive actions, the above mentioned cycle is restarted with some requirements being modified. There may be a case that a failure responsive action fails to respond to the failure. Such a case may happen due to some unexpected environment changes, inadequate risk analysis, bugs of the D-Script itself, and so forth. In such a situation, the above mentioned cycle must also be restarted.

Several iterative processes for adapting to requirement and environmental changes have been pro-

posed. However, as far as we know there have been not so much studies on integrating requirement elicitation, risk analysis, and in-operation with tool implementations.

In this paper, together with the framework, we have implemented assurance case tool, called *D-Case Editor*[4] and make a tool chain with a runtime system (D-RE) and a failure mitigation scripting code (D-Script). We demonstrate the feasibility of the framework by a web server demo system. D-Case Editor is a graphical editor and viewer of assurance cases. Using D-Case Editor, the stakeholders including the developers and the operators are always able to write, edit, and observe the assurance case(D-Case) of the system.

Web server systems are typical information systems and several serious failures have been reported such as loss of clients data. The structure of the demo system is shown in Figure 2. The web server system consists of three main components: a web server, an application server, and a database server. The service level agreement document (SLA) document is assumed as in Figure 3. The service provided

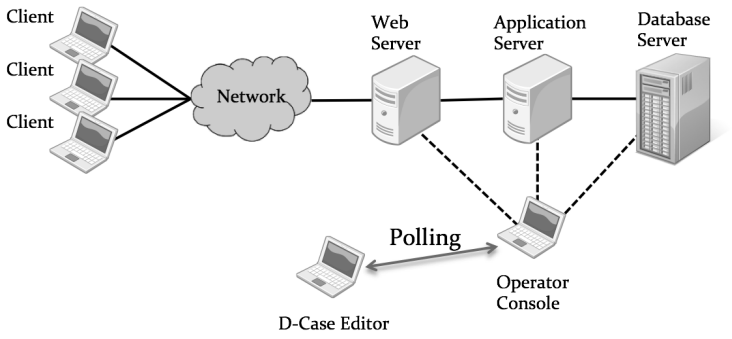


Figure 2: A Snapshot of Web Server Demo

by the web server is CD online sales.

**SLA document**

1. Continuity priority index
   (a) End user containment program : allowable down time 30 seconds, once/half of year
   (b) Main business programs : allowable down time 3 minutes, once/month
   (c) Additional programs : isolation in case of failure, twice/month
2. Performance
   (a) Access capacity : 2500 access/min
   (b) Response time : 800 msec/access
3. DB capacity
   (a) Registered end users : 1,000,000
   (b) Variety of CDs : 1,000,000

Figure 3: Service Agreement document for the web server demo system

The developer of the web server system need to assure the dependability of the system to the web service provider. In this case the dependability of the system can be considered as the SLA requirements.

The developers of the web server need to assure that the web server can continue the service satisfying the SLA even some failures occur. The detail will be shown in Section 4.

Also, in Figure 2, the monitoring information of the runtime system is always polled to D-Case Editor at regular intervals. This enables the operators of the system to check all monitoring values are within the ranges during the operational phase.

Our contributions are as follows.

- We propose a process cycle for dependability consensus and in-operation assurance.
- We implement prototype tools for realizing the process cycle: D-Case Editor, D-RE, and D-Script.
- We show a web server demo system to show how our framework works, and conducted initial evaluation of the framework.

The structure of this paper is as follows. In Section 2, we introduce our requirement elicitation and risk analysis methods. Section 3 introduces D-Case for making agreement among stakeholders and in-operation assurance. Also, we show an abstract view of the cycle in Section 3. In Section 4, we show current implementation status and an example using the techniques described in this paper. Section 5 concludes the paper.

## 2 Requirements Elicitation and Risk Analysis

The requirements elicitation starts with the service objectives. The stakeholders can be defined according to their service objectives. Requirements are elicited from each stakeholder's objectives and needs. Here, requirements include service requirements and dependability requirements. Regulations made by regulatory agencies can also be considered as a kind of requirements.

In requirements engineering, various requirements elicitation methods have been proposed such as Ethno-Methodology, Trolling, Business Modeling, Goal Oriented Analysis, Use Case Analysis, Misuse Case Analysis, and Triage.[5, 6, 7, 8]. Leveson[9] and Ericson[10] introduced methods for safety requirements analysis, such as FMEA, HAZOP, FTA, ETA. Kotonya and Sommerville showed a method for analyzing safety requirements using Hazard analysis and FTA [8]. Troubitsyna proposed Component based FMEA (Failure Mode and Effects Analysis) to analyze how component failures affect behavior of systems [11]. Sasaki and Taniyama proposed Multiple Risk Communicator to the personal information leakage problem [12, 13].

We mainly focus on dependability requirements. First, service needs are extracted from stakeholders who describe service needs informally and verbally, and from these, dependability needs are obtained. Second, "gdependability requirements" are identified through the analysis of dependability needs. Next, "service continuity scenarios" are created based on risk analysis and service requirements. More precisely, service continuity scenarios are developed by considering and determining countermeasures for each factor which cause deviations. Finally, D-Case and D-Script are made through consensus building among stakeholders based on the service continuity scenarios.

Based on previous work on requirement elicitation and risk analysis, we define several requirement elicitation and risk analysis methods which are used in our framework.

Table 1 shows management techniques used to elicit requirements and analyze risks. Service consensus building card (SCBC) is used to define service requirements and to agree on the requirements among stakeholders. Dependability Control Board (DCB) manages consensus building process with SCBC. DCB members are representatives of stakeholders. Dependability Control Map (DCMap) describes relationships among dependability goals as well as roles of stakeholders. D-Cases are stored in D-Case DB and used to achieve dependability goals for dependability requirements of services. Service Risk

Brake-down Structure (SRBS) hierarchically decomposes risks into categories. Service Fault Tree (SFT) describes the logical conditions for failures. Service Continuity Scenario (SCS) are designed to mitigate risks for dependability requirements. SCS are implemented as D-Scripts. Service Risk Management Table (SRMT) defines service risks based on probabilities and impacts of failures according to service event scenarios. Service Requirements State Management (SRSM) manages service requirements state not only during online but also offline. Figure 4 shows relationships among techniques given in Table 1. Dependability requirements in DCMap are precisely defined and agreed on using SCBC. SRBS is then used to analyze risk category. SRMT is used to identify and mitigate risks of services elicited using SCBC. SFT is developed for each scenario in SRMT to show conditions of fault occurrences. D-Cases are written based on the information of DCMap and SRMT.

Table 1: Requirement Management Table

| Techniques | | Explanation |
|---|---|---|
| SCBC | Service consensus building card | SCBC is used to define service requirements and agree on among stakeholders |
| DCB | Dependability Control Board | DCB manages consensus building process with SCBC. DCB members are representatives of stakeholders. |
| DC Map | Dependability Control Map | DC Map describes relationships among dependability goals as well as roles of stakeholders. |
| D-case DB | D-case data base | D-cases are stored to achieve dependability goals for dependability requirements of services. |
| SRBS | Service Risk Brake-down Structure | SRBS hierarchically decomposes risks into categories. |
| SFT | Service Fault Tree | SFT describes the logical conditions for failures. |
| SCS | Service Continuity Scenario | SCS are designed to mitigate risks for dependability requirements. SCS are implemented by D-Scripts. |
| SRMT | Service Risk Management Table | SRMT defines service risks based on probabilities and impacts for service event scenarios. |
| SRSM | Service Requirements State Management | SRSM manages service requirements state not at online but also offline. |

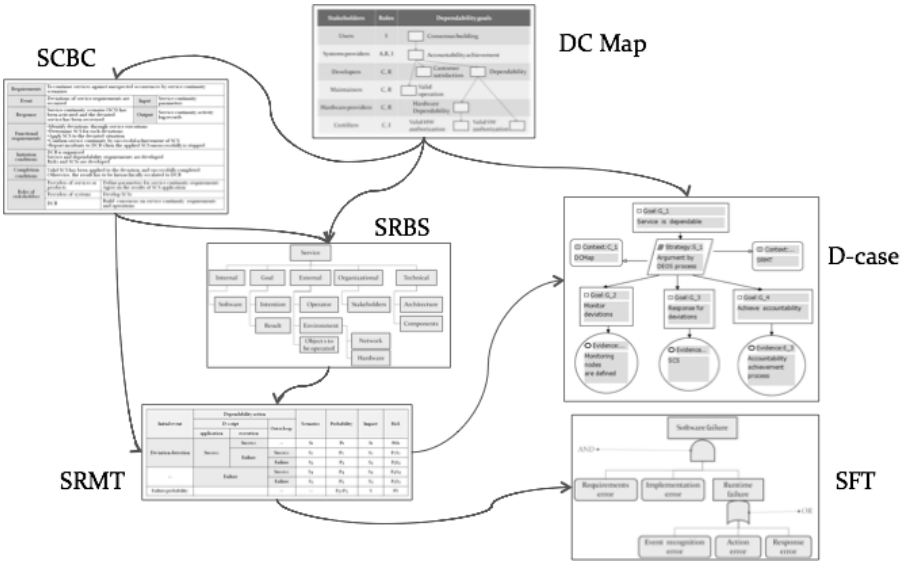


Figure 4: Relationship of RM Techniques

We describe some of the techniques in Table 1.

An example of Dependability Control Map (DCMap) is shown in Figure 5. DCMap contains three columns: stakeholders, roles, and dependability goals. Stakeholders and roles columns correspond to RACI matrix [14]. R, A, C, and I represent Responsible, Accountable, Consulted, and Informed, respectively. The dependability goals column describes goals of the stakeholders and their relationships. In Figure 5, users need to be informed that consensus has been made on service dependability. This is accomplished by accountability achievement goal of system providers. The accountability achievement goal is supported by goals of the developers and the maintainers. Dependability goals of the developers are also supported by hardware dependability and valid software authorization.

Table 2 shows an example of service consensus building card (SCBC). SCBC consists of requirements name, event, response, input, output, functional requirements steps, initiation condition, completion condition, and roles of stakeholders. This figure omits the identification of SCBC for simplicity. Figure 6 shows an example of Service Risk Breakdown Structure (SRBS). Service risks are broken down into internal, goal, external, organizational, and technical risks. A service has a goal that is the intention and result that an actor, who wants to use the service, expects to get from the system. By getting an event from actors, services will act on objects and generate a result to achieve the goal. Services will also make responses to actors. Services work on an environment including hardware and network. Deviations of these components will cause service risks. Service continuity scenarios can be constructed to mitigate these risks by considering deviations of service constituents. This risk breakdown structure is based on PMBOK.

Table 3 shows an example of Service Risk Management Table (SRMT). SRMT describes initial events, dependability actions, scenarios, probabilities, severity of impacts, and risks. The structure of SRMT is decomposed into two parts. The left part of SRMT describes scenarios using a binary tree of success and failure. The right part of SRMT describes the risk of each scenario.

## 3 D-Case: Assurance Case for Stakeholders Agreement and In-Operation Assurance

D-Case is an extension of assurance case[2]. An extension of D-Case is introducing the notion of run time system monitoring.

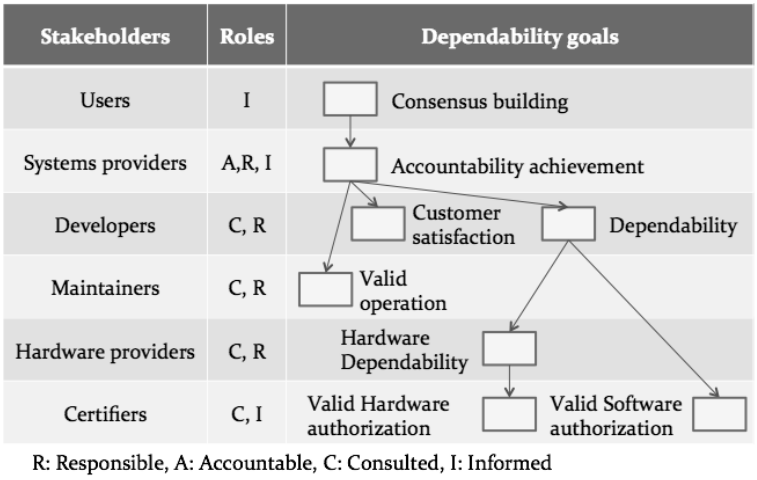


Figure 5: An Example Dependability Control Map

Table 2: An Example Service Consensus Building Card (SCBC)

| | | | |
|---|---|---|---|
| Requirements | To continue services against unexpected occurrences by service continuity scenarios | | |
| Event | Deviations of service requirements are occurred | Input | Service continuity parameters |
| Response | Service continuity scenario (SCS) has been activated and the deviated service has been recovered | Output | Service continuity activity log records |
| Functional requirements | •Identify deviations through service executions<br>•Determine SCS for each deviations<br>•Apply SCS to the deviated situation<br>•Confirm service continuity by successful achievement of SCS<br>•Report incidents to DCB when the applied SCS unsuccessfully is stopped | | |
| Initiation conditions | DCB is organized<br>Service and dependability requirements are developed<br>Risks and SCSs are developed | | |
| Completion conditions | Valid SCS has been applied to the deviation and successfully completed<br>Otherwise, the result has to be hierarchically escalated to DCB | | |
| Roles of stakeholders | Providers of services or products | Define parameters for service continuity requirements<br>Agree on the results of SCS application | |
| | Providers of systems | Develop SCSs | |
| | DCB | Build consensus on service continuity requirements and operations | |

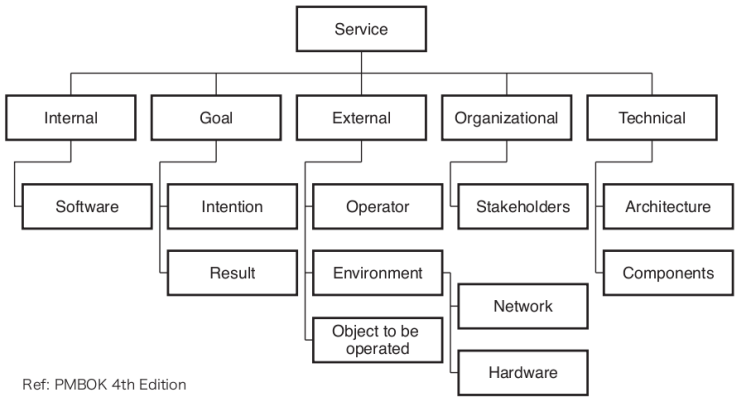


Figure 6: An Example of Service Risk Breakdown Structure (SRBS)

Table 3: An Example of Service Risk Management Table (SRMT)

| Initial event | Dependability action | | | Scenarios | Probability | Impact | Risk |
|---|---|---|---|---|---|---|---|
| | D-script | | Failure response by operator | | | | |
| | application | execution | | | | | |
| Deviation detection | Success | Success | -- | $S_1$ | $P_1$ | $S_1$ | $P_1S_1$ |
| | | Failure | Success | $S_2$ | $P_2$ | $S_2$ | $P_2S_2$ |
| | | | Failure | $S_3$ | $P_3$ | $S_3$ | $P_3S_3$ |
| -- | Failure | | Success | $S_4$ | $P_4$ | $S_4$ | $P_4S_4$ |
| | | | Failure | $S_5$ | $P_5$ | $S_5$ | $P_5S_5$ |
| Failure probability | | | -- | -- | $P_3+P_5$ | S | PS |

As systems become ever-changing, it has become difficult to sustain dependability of the systems only by conventional methods. We observe that the best way is stakeholders argue dependability of the system with evidence supported by experts, and try to reach agreement that the system is dependable through the whole system lifecycle. For the objectives, first, we need a method to describe and evaluate dependability requirements. Dependability requirements need to be understood by diverse stakeholders involved in the whole system lifecycle. Second, a mechanism should be in place that ensures traceability between dependability agreement and actual system behaviors. The mechanism not only keeps track of the development phases of a system, but also its run-time operations by constantly checking whether dependability requirements are being satisfied or not. In particular, we must update dependability agreement when changes occur.

To achieve these two goals, we have started our study with system assurance. From several methods in system assurance, we exploit assurance case [2] to describe and evaluate dependability requirements. Assurance cases are structured documents for assuring dependability/safety/reliability/etc. of systems based on evidence. This simple framework has recently been widely used for safety critical domain. This is because as systems become large and complex, only following some safety checklists does not satisfy safety requirements, but assuring safety of systems becomes crucial. Current assurance cases, however, are mostly written in weakly-structured natural languages, and it is difficult to ensure traceability between assurance cases (and associated documents) and system's actual states during the whole lifecycle.

We show a brief introduction of assurance cases. Safety cases (assurance cases for safety of systems) are required to be submitted to certification bodies for developing and operating safety critical systems, e. g., automotive, railway, defense, nuclear plants and sea oils. There are several standards, e.g., EUROCONTROL [15] and MoD Defence Standard 00-56, which mandate the use of safety cases. There are several definitions for assurance cases. We give one such definition as follows [16].

> a documented body of evidenc that provides a convincing and valid argument that a system is adequately dependable for a given application in a given environment.

Assurance cases are often written in a graphical notation. Goal Structuring Notation (GSN) is one of such notations [17]. Writing assurance cases and reusing them in a cost effective way is a critical issue for organizations. Patterns and their supporting constructs are proposed in GSN to enable the reuse of existing assurance cases, which includes parameterized expressions. Another widely used notation is Claims, Arguments and Evidence (CAE), which was developed by Adelard and City University London [18].

We show D-Case nodes (Figure 7) and an example (Figure 8). Current D-Case syntax is mostly the same as GSN except for the monitoring nodes.

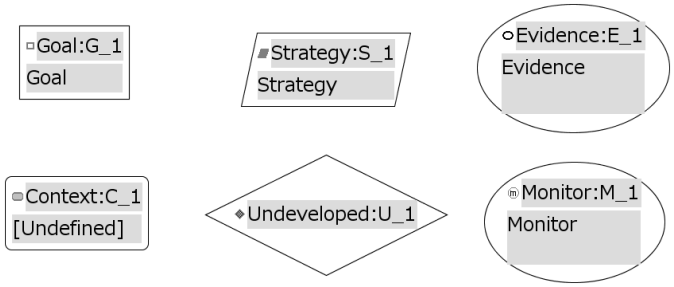


Figure 7: D-Case Nodes

We briefly explain constructs and their meanings in D-Case. Arguments in D-Case are structured as trees with a few kinds of nodes, including: Goal nodes for claims to be argued for, Strategy nodes

for reasoning steps that decompose a goal into sub-goals, and Evidence nodes for references to direct evidence that respective goals hold. Context nodes are associated to Goal and Strategy nodes to describe environmental information and assumptions for the Goal and Strategy nodes. Undeveloped nodes are attached to goals if there are no supporting arguments for the goals at that time. In D-Case, monitoring nodes are a sub-class of evidence nodes. They are intended to represent evidence available at runtime, corresponding to the target values of monitoring points of the run time system. Figure 8 is a simple example of D-Case. The root of the tree must be a goal node, called top goal, which is the claim to be argued (G1). A context node C1 is attached to complement G1. Context nodes are used to describe the context (environment) of the goal to which the context is attached. A goal node is decomposed through a strategy node S1 into sub goal nodes (G2 and G3). A strategy node contains an explanation, or reason, for why the goal is achieved when the sub goals are achieved. S1 explains the way of arguing (argue over dependability requirements (safety and reliability)). When successive decompositions reach a sub goal (G2) that has a direct evidence of success, an evidence node (E1) referring to the evidence is added. Here we use a result of fault tree analysis (FTA) as the evidence. The sub goal (G3) is supported by monitoring node M1. In this D-Case, G3 is supported by runtime log results.

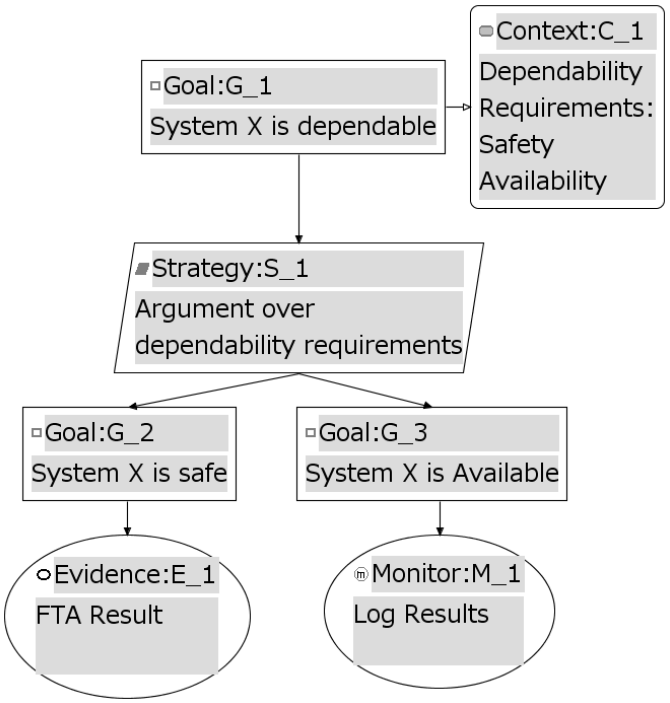


Figure 8: D-Case Example

## 3.1 In-operation Assurance

This section shows our initial idea of in-operation assurance with a reference implementation. A demo of our idea was presented in Embedded Technology 2011, one of the largest exhibitions for embedded systems in Japan. Figure 9 shows a reference system for In-Operation Assurance.

D-Case DB contains D-Case patterns for failure response action. D-Case Pattern $\Longleftrightarrow$ Module Mapping Table contains mappings between parameters used in D-Case pattern and system modules. Using the table, D-Case pattern is translated to D-Script. The right-hand side of Figure 9 is a simplified D-RE,

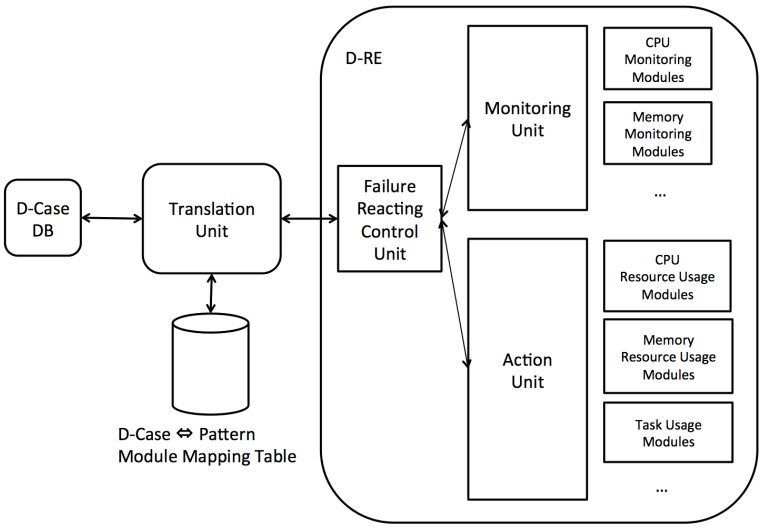


Figure 9: A Reference System for In-Operation Assurance

in which the Monitoring Unit and Action Unit have monitoring and failure response action modules, respectively. The key concept of the reference system is that only system behaviors, which are agreed upon and stored as D-Cases, can be executed. Operators of the system are able to choose appropriate action as a failure response action based on a D-Case from agreed upon D-Cases. Figure 9 shows an example of D-Case pattern for over usage of CPU resources. Argument of the D-Case pattern is that if CPU usage rate exceeds 50% (detected by monitoring), the failure recover control unit invokes CPU resource usage module to restrict CPU usage under 50%. In Figure 10, a monitoring node is exploited. Task "A", "CPU resource usage rate", and "under 50%" in those monitoring nodes are value of parameters which operators and other stakeholders agreed. For example, we can specify the name of some other CPU task instead of "A", "Memory resource usage rate" instead of "CPU usage rate", etc. Setting the values of parameters automatically generates executable codes.

## 3.2 An Abstract View of the Framework

In requirements management, as mentioned above, states of requirements are managed. There are four kinds of the states; elicited, agreed, ordinarily operated, and deviated (Figure 11). First, requirements are elicited from stakeholders. These elicited requirements may conflict with each other. By consensus-building, requirements are agreed upon among the stakeholders. Agreed-upon requirements are then implemented in ordinary operations. When objectives and environments change, some ordinarily operated requirements may become obsolete and new requirements must be elicited again.

If a requirement is not fulfilled, i.e., there is deviation from the corresponding in-operation range, it moves to the deviated state. When a responsive action is possible, it moves back to the ordinarily operated state. If the service continuity scenarios cannot work for some requirements in the deviated state, these requirements should be modified and move to the elicited state. If deviations came from the implementation problems, the corresponding elicited requirements do not need any change. But it is necessary to agree on other requirements to revise the faulty implementation. This is done by consensus-building. The elicited and agreed states of requirements are managed at offline, whereas ordinarily operated and deviated states are managed online. The state of the system is represented by a set of these requirements states. Figure 12 shows how this set of requirements are managed by requirements

management table as the system evolves.

Traditional requirements management (TRM) methods only consider states of requirements at offline [8, 5, 19, 20, 21, 22, 23]. These requirements engineering text books describes requirements management process by Change Control Board (CCB) with requirements change requests. TRM does not consider requirements deviations at runtime. Requirements management state model can take into account deviations of requirements at runtime. To detect and manage deviation, it is necessary to record the deviation situations on requirements with identifications, events, inputs, outputs, and responses. Otherwise, there is no evidence on deviations and it is impossible to analyze failures.

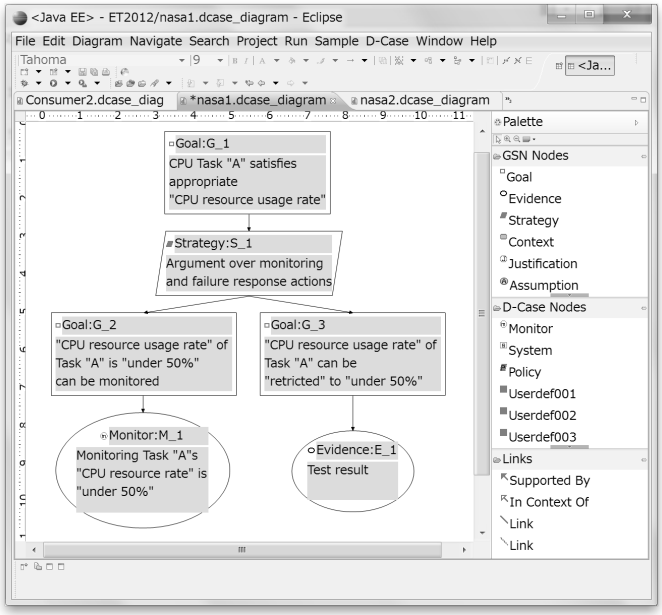


Figure 10: An Example of D-Case Pattern

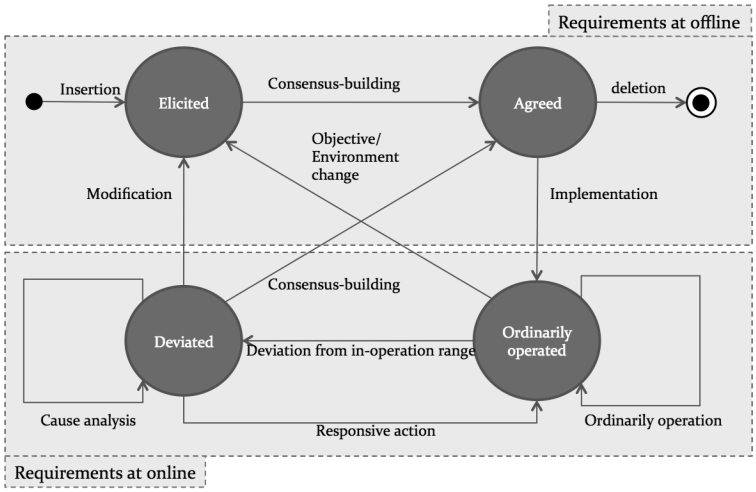


Figure 11: Requirements State Management Model

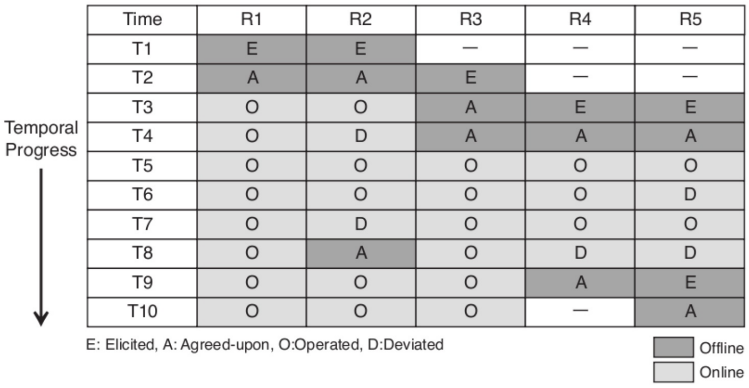


Figure 12: System Requirement State Management Table

# 4 Implementation Status and an Example Using our Techniques

Currently the requirement elicitation and risk analysis methods have been designed. Also, we have been developing "D-Case Editor" [24], which is a tool to support stakeholders' agreements, and "D-Case Viewer", which is a tool to monitor whether stakeholders' agreements are satisfied or not. D-Case Editor is a graphical editor as an Eclipse plug-in. Figure 10 is also a snapshot of D-Case Editor.

The prototype monitoring mechanism is achieved by embedding D-Script information files into D-Case XML files. A D-Script code (for monitoring or failure recovering actions) corresponds to a monitoring node in D-Case. In a D-Case XML file, nodes and link information among the nodes are defined. An example of D-Case XML embedding D-Script information in a monitoring node is shown in Figure 13 (only the part of the monitoring node is shown). In the XML file, several values for monitoring settings are defined in the `<dcase:d-script>` part.

In the web server system shown in Introduction, several risks can be considered (Figure 14).

For these risks, a dependability control map can be defined as in Figure 15. A consensus building card for the SLA document is defined as in Figure 16. Furthermore, the service risk breakdown structure for the web server demo system is described in Figure 17.

Based on these analysis, a D-Case for the web server demo system was written. The part of D-Case for web server transaction overflow failure is shown in Figure 18. In the D-Case, the mitigation of web server transaction overflow is argued by separating monitoring and mitigation actions. The mitigation actions are designed according to a service continuity scenario: first, try to restrict the access numbers from clients; second, undo the previous update immediately before the failure. In this case, the previous update was installing a new program; third, the result of the failure mitigation will be reported to the stakeholders: End users, system providers, developers, and operators if necessary. The D-Script code consists of three units, and D-Case nodes for failure mitigation actions exactly corresponds to the three units.

We have implemented a prototype demo system according to the above analysis and D-Case. Figure 19 shows a snapshot of the demo[1]. Monitoring information of the system is constantly polled to D-Case Viewer.

We show a preliminary result of the demo system in Table 4. Table 4 shows how CPU and memory resources are consumed by the prototype system. Note that this experiment is done with two-core CPU, so the total CPU resource is 200%. Only 0.1% of memory resources of the target system were consumed

[1] A Demo movie can be obtained from https://dl.dropbox.com/u/13455869/D-CaseMovie.mp4

```
<?xml version="1.0" encoding="UTF-8"?>
<dcase:dcase xmlns:dcase="http://www.dependable-os.net/2010/06/dcase" id="m11">
<dcase:description/> <dcase:properties/> <dcase:nodes>
:
:
<dcase:node type="Monitor" id="_un-SwKuiEeGUa93z1Rd6MQ" name="M_11">
<dcase:description>Monitor at the entry to Web Server</dcase:description>
<dcase:d-script>
<dcase:full-name>www.dependable-os.net/dre/apache.Monitor
</dcase:full-name>
<dcase:values>
<dcase:value name="interval">2</dcase:value>
<dcase:value name="host">sys-apa</dcase:value>
<dcase:value name="crit">1500</dcase:value>
<dcase:value name="warn">1250</dcase:value
<dcase:value name="rrd-dir">/var/lib/dre-monitor/rrd/sys-apa/apache</dcase:value>
<dcase:value name="graph-dir">/var/lib/dre-monitor/chart/sys- apa/apache
</dcase:value>
</dcase:values>
</dcase:d-script>
</dcase:node> :
:
</dcase:nodes> :
:
</dcase:dcase>
```

Figure 13: An Example of D-Case XML file

- Web access transaction overflow in the web server
- Application throughput down in the application server
- Operation monitoring failure in the operator console, and
- Database transaction overflow in the database server.

Figure 14: Analyzed Risks for the Web Server System

| Stakeholders | Roles | Dependability claims |
|---|---|---|
| End user | I | Containment Program: allowable down time 30 seconds, once/half of year |
| System provider | A,R, I | Service continuity claim |
| developer | C, R | Resource claim; Performance claim |
| operator | C, R | Main Business Programs: allowable down time 3 minutes, once/month |

R: Responsible, A: Accountable, C: Consulted, I: Informed

Figure 15: Dependability Control Map for the Web Server

Table 4: CPU and Memory Usage of Polling to D-Case Viewer

| Polling Cycle | CPU usage (%) | Memory usage (%) |
|---|---|---|
| None | 0.0 | 0.1 |
| 1 second | 2.8 | 0.1 |
| 5 seconds | 0.5 | 0.1 |

by this mechanism in all three cases: no polling, polling for every 1 second, and for every 5 seconds. CPU resources of the target system were consumed 2.8% when polling cycle is 1 second, which is relatively high. We are now checking the cause, but considering that this is a prototype implementation, CPU usage of this mechanism should be improved in more elaborate implementation.

## 5 Concluding Remarks

This paper has reported our initial ideas and implementation for consensus building and in-operation assurance for service dependability. We have presented several methods for requirement elicitation and

| | | | |
|---|---|---|---|
| Requirements | To continue services against unexpected occurrences by service continuity scenarios | | |
| Event | Deviations of service requirements are occurred | Input | Continuity Priority Index<br>Performance Index<br>DB capacity |
| Response | Service continuity scenario (SCS) has been activated and the deviated service has been recovered | Output | Service continuity activity monitoring log records |
| Functional requirements | •Identify deviations through service workflow using monitor nodes<br>•Determine SCS for each deviations<br>•Apply SCS to the deviated situation<br>•Confirm service continuity by successful achievement of SCS | | |
| Initiation conditions | Service and dependability claims are developed for each servers<br>Risks are identified and SCSs are developed to mitigate these risks | | |
| Completion conditions | Valid SCS has been applied to the deviation using monitoring nodes and successfully completed | | |
| Roles of stakeholders | Web service provider | Define parameters for service continuity requirements<br>Agree on the results of SCS application | |
| | System developer | Develop service and SCSs | |
| | Operator | Operate and recover web service | |

Figure 16: SLA Consensus Building Card

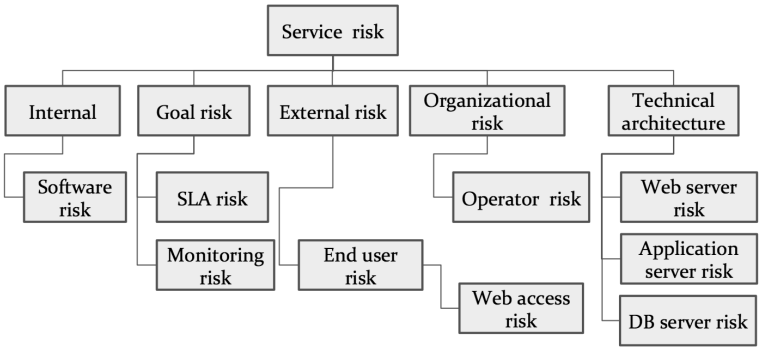


Figure 17: Service Risk Breakdown Structure

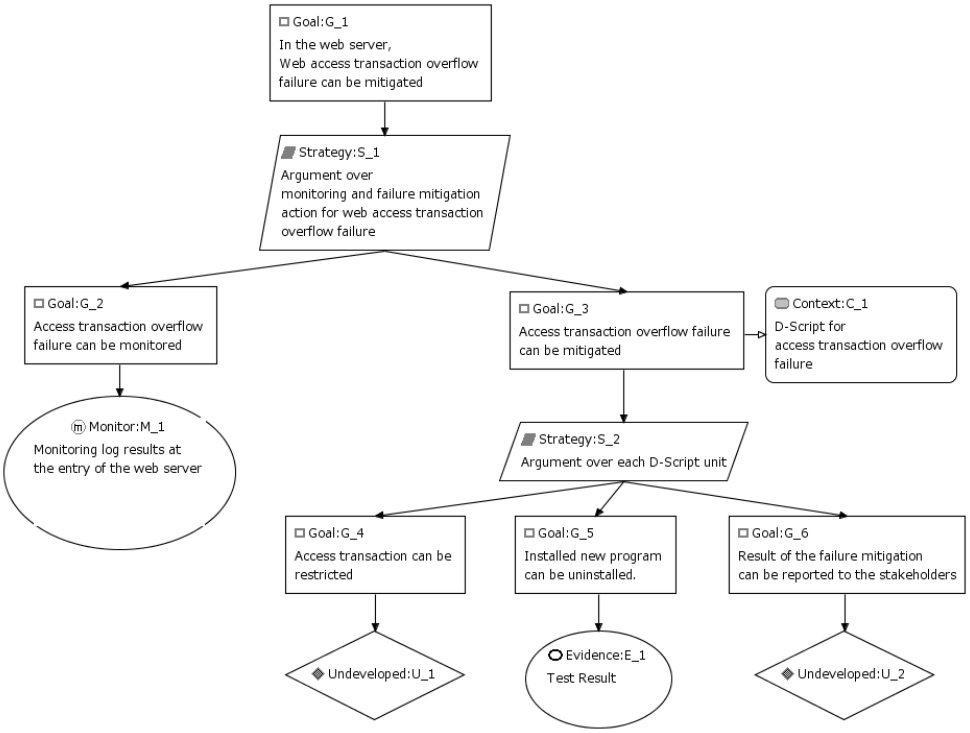


Figure 18: A Part of D-Case for the web server demo system (for web server transaction overflow failure)

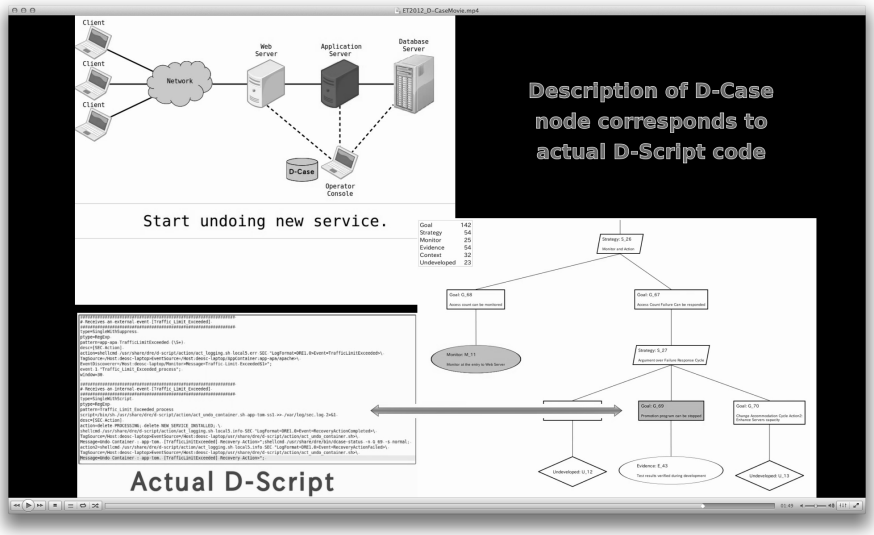


Figure 19: A Snapshot of Web Server Demo System

risk analysis. Also we have presented D-Case, which is an extension of assurance case for in-operation assurance. We have done a case study on a web server demo system to show how our techniques described in this paper work. Currently, the basic framework has been designed and a few techniques in the framework has been implemented as a prototype system. Several requirement elicitation and risk analysis methods in this paper have still not been demonstrated. As far as we know there have been not so much frame works with some implementations that demonstrate requirement management considering system failures. Therefore we believe that our works have some contributions to the research field.

Our project, DEOS project [3] is now focusing on evaluating our framework in real systems. We would like to present our progress with some evaluation in a new future.

## References


[1] Y. Matsuno and S. Yamamoto, "Consensus building and in-operation assurance for service dependability," in *Proc. of the 2nd Multidisciplinary Research and Practice for Information Systems - IFIP WG 8.4, 8.9/TC 5 International Cross-Domain Conference and Workshop on Availability, Reliability, and Security (CD-ARES'12), Prague, Czech Republic, LNCS*, vol. 7465. Springer-Verlag, August 2012, pp. 639–653.

[2] C. C. Howell, S. Guerra, S. L. Pfleeger, and V. Stavridou-Coleman, "International Workshop on Assurance Cases: Best Practices, Possible Obstacles, and Future Opportunities," in *Proc. of the 2004 International Conference on Dependable Systems and Networks (DSN'04), Florence, Italy*. IEEE, July 2004.

[3] M. Tokoro, "White paper: Dependable embedded operating system for practical use (DEOS) project, version 3," 2011.

[4] Y. Matsuno, H. Takamura, and Y. Ishikawa, "A dependability case editor with pattern library," in *Proc. of the 12th IEEE International Symposium on High-Assurance Systems Engineering (HASE'10), San Jose, California, USA*. IEEE, November 2010, pp. 170–171.

[5] A. M. Davis, *Just Enough Requiremtns Management- Where Software Development Meets Marketing*. Dorset House Publishing, 2005.

[6] D. Zowghi and C. Coulin, *Requirements Elicitation: A survey of Techniques, Approaches, and Tools*. Springer, 2010, ch. 2, pp. 19–46.

[7] A. Aurum and C. Wohlin, Eds., *Engineering and Managing Software Requirements Engineering and Managing Software Requirements*. Springer, 2010.

[8] G. Kotonya and I. Sommerville, *Requirements Engineering-Process and Techniqeus*. John Wiley and Sons, 2002.

[9] N. G. Leveson, *Safeware: System Safety and Computers*. Addison-Wesley, 1995.

[10] C. A. Ericson, *Hazard Analysis Techniques for System Safety*. John Wiley and Sons, Inc., 2005.

[11] E. Troubitsyna, "Elicitation and specification of safety requirements," in *Proc. of the 3rd International Conference on Systems (ICONS'08), Cancun, Mexico*. IEEE, April 2008, pp. 202–207.

[12] R. Sasaki, Y. Hidaka, Y. Murayama, S. Ishii, H. Yoshiura, and H. Yajima, "Development concept for and trial application of a multiplex risk communicator," in *Proc. of the 5th IFIP conference on e-Commerce, e-Business, and e-Government (I3E'05), Posnan, Poland, IFIP*, vol. 189. Springer-Verlag, October 2005, pp. 607–621.

[13] M. Taniyama, Y. Hidaka, M. Arai, S. Kai, H. Igawa, H. Yajima, and R. Sasaki, *Application of Multiple Risk Communicator to the Personal Information Leakage Problem*. World Academy of Science, 2008, pp. 284–289.

[14] I. I. of Business Analysis(IIBA), *BABOK 2.0*. International Institute of Business Analysis(IIBA), 2009.

[15] European Organisation for the Safety of Air Navigation, "Safety case development manual," european Air Traffic Management, 2006.

[16] Centre for Software Reliability, City University London, "Assurance cases and argumentation," http://www.csr.city.ac.uk/research.html.

[17] T. Kelly and R. Weaver, "The goal structuring notation - a safety argument notation," in *Proc. of the 2004 IEEE DSN Workshop on Assurance Cases, Florence, Italy*, 2004.

[18] Adelard, "Claims, arguments and evidence(CAE)," http://www.adelard.com/web/hnav/ASCE/choosing-asce/cae.html.

[19] I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide*. John Wiley and Sons, 1997.

[20] D. Leffingwel and D. Widrig, *Managing Software Requirements A Unified Approach*. Addison-Wesley Professional, 1999.

[21] K. Wiegers, *Software Requirements- Practical techniques for gathering and managing requirements through the product development cycle*. Microsoft Corporation, 2003.

[22] K. Pohl, *Requirements Enginerring Fundamentals, Principles, and Techniques*. Springer-Verlag, 2010.

[23] B. Berenbach, D. Paulish, J. Kazmeier, and A. Dudorfeer, *Software and Systems Requirements Engineering In Practice*. McGraw Hill, 2009.

[24] Y. Matsuno, "D-Case Editor Web Page," http://www.dependable-os.net/tech/D-CaseEditor.

**Yutaka Matsuno** received the B.S., M.S., and Ph.D. degree from the University of Tokyo in 2001, 2003, and 2006, respectively. Currently he is Project Lecturer in Strategy Office, Information and Communications Headquarters, Nagoya University, Japan. His main research interests include system assurance, assurance case, dependability, and programming languages.

**Shuichiro Yamamoto** received the B.S. degree from Nagoya Institute of Technology and the M.S. and Dr.Eng. degrees from Nagoya University in 1977, 1979, and 2000, respectively. He joined Nippon Telegraph and Telephone Public Corporation (now NTT) in 1979 and engaged in the development of programming languages, CASE tools, network-based smart card environments, and distributed application development platforms. His research interests include distributed information systems, requirements engineering, ubiquitous computing, knowledge creation, dependability engineering, and Enterprise Architecture. He moved to NTT DATA in 2002. He became the first Fellow of NTT DATA Research and Development Headquarters in 2007. He moved to Nagoya University as a professor in 2009. He is currently the leader of working group for the requirements evolution based system development method that is a research project of the Software Engineering Center of IPA ( Information technology Promotion Agency, Japan). He is the chairman of AEA (Association of Enterprise Architects) Japan. He is the chairperson of SIG Knowledge Sharing Network of The Japanese Society for Artificial Intelligence.