PeerJ Computer Science

Continuously revised assurance cases with stakeholders' cross-validation: a DEOS experience

Kimio Kuramitsu

Graduate School of Electronic and Computer Engineering, Yokohama National University, Japan

ABSTRACT

Recently, assurance cases have received much attention in the field of software-based computer systems and IT services. However, software changes very often, and there are no strong regulations for software. These facts are two main challenges to be addressed in the development of software assurance cases. We propose a method of developing assurance cases by means of continuous revision at every stage of the system life cycle, including in operation and service recovery in failure cases. Instead of a regulator, dependability arguments are validated by multiple stakeholders competing with each other. This paper reported our experience with the proposed method in the case of Aspen education service. The case study demonstrates that continuous revisions enable stakeholders to share dependability problems across software life cycle stages, which will lead to the long-term improvement of service dependability.

Subjects Security and Privacy, Software Engineering

Keywords Assurance cases, GSN, DEOS process, Experience report, Service dependability

INTRODUCTION

Assurance cases are documentation-based engineering with structured arguments on safety and dependability. Originally, assurance cases were developed in the field of safety engineering for public transportation and industrial plants, and they have been adopted broadly as a documentation standard (*Bloomfield & Bishop, 2010*). Many regulators, especially in EU countries, are likely to validate assurance cases before the developed systems are deployed.

Recently, due to increased attention to safety and dependability in software, many developers have become interested in the application of assurance cases for software. However, software often changes over time and even needs to change after deployment. The emerging style of DevOps development suggests that it would be difficult to separate developments from service operations. This also makes it difficult for a regulator to assess assurance cases, thereby resulting in the absence of strong regulators for software in general.

To overcome these difficulties, the DEOS process (*Tokoro, 2015*) was developed with the life cycle maintenance of assurance cases. The idea is straightforward: assurance cases are revised in a synchronized way as software updates. An arising question is as follows: who validates such revised assurance cases, even in the post-development phase? The answer is still unclear in the DEOS process. In this paper, we assume that stakeholders

Submitted 13 January 2016
Accepted 14 November 2016
Published 19 December 2016

Corresponding author
Kimio Kuramitsu, kimio@ynu.ac.jp

Academic editor
Mariagrazia Fugini

Additional Information and Declarations can be found on page 15

DOI **10.7717/peerj-cs.101**

who are competing with each other are motivated to validate cases because they will suffer directly from others' faulty claims in assurance cases. However, many questions remain. Can non-expert stakeholders learn about assurance cases and then join dependability arguments? Is the validation strong enough?

To confirm these questions, we organized a case study experiment, the Aspen project, in which multiple stakeholders (e.g., developers, operators, and users) are involved in the software life cycle from development to service operation. Throughout the life cycle, all stakeholders will have participated in arguing for or against the dependability using assurance cases written in Goal-Structuring Notation (GSN) (*Kelly & Weaver, 2004*), a standard notation of assurance cases.

Unfortunately, service failures occurred during the experiment period, although all the stakeholders made extensive arguments in favor of GSN notations. The occurrence of service failure does not imply the weakness of stakeholders' cross-validation, because system failure happened in a case of regulator validations. Rather, the analysis of the failure case gives us a useful insight: the structured dependability arguments in GSNs make it easier to find and share dependability problems across organizations. Since transferring dependability problems is a missing part of software life cycle iteration, continuously revised assurance cases can be a new approach to the long-term dependability of ever-changing software.

This paper focuses on the report of the Aspen case study. The rest of the paper proceeds as follows: 'What Are Assurance Cases?' introduces assurance cases and reviews-related work; 'Argumentation Architecture' presents our basic ideas for developing assurance cases for software; 'Experimental Setup: Aspen Project' describes the Aspen project; 'Aspen Cases' examines the assurance cases that were developed in the Aspen project; 'Lessons Learned and Discussions' discusses lessons learned; and 'Conclusion' concludes the paper.

WHAT ARE ASSURANCE CASES?

Assurance cases are document-based engineering with structured arguments related to safety and dependability (*Avizienis et al., 2004*). In this paper, we use the term *dependability* in a broader sense related to safety. The documents of assurance cases are structured to transfer the dependability confidence of products and services to others, such as regulators and third-party organizations. To make the confidence explicit, assurance cases are usually argued in the form of *claim-argument-evidence* (CAE). Figure 1 illustrates the conceptual structure of assurance cases with the CAE arguments.

For example, consider the claim that a system is adequately dependable for operating in a given context. The argument explains the available evidence, showing how it reasonably supports the claim. The top-most claim is decomposed into a series of sub-claims until these can be solved with evidence. Since the arguments make the rationale for the claim explicit, they are rigorous, justified, defensible, and transparent.

Due to the high transparency of dependability arguments, assurance cases generally serve as an efficient risk-communication tool between organizations. However, the most practically used scenario is transferring the developer's confidence to a regulator or a

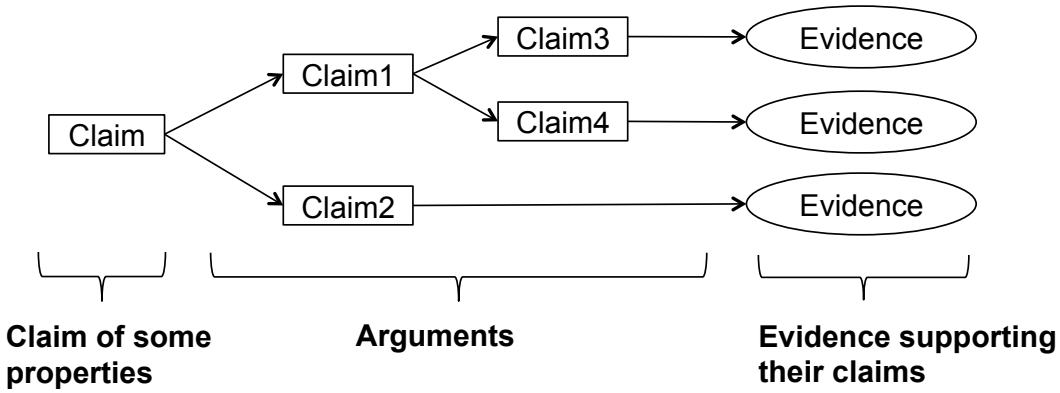

Figure 1 **Argument structure of assurance cases.**

third-party company to assess the conformance of dependability regulations (*Graydon et al., 2012*). Through this assessment mechanism, the regulator forces the developer's product to meet its regulations, and then the user trusts the developer's product due to the regulator's authority. In contrast, the self-assessment of conformance or the absence of dependability regulations makes assurance cases self-righteous and less confident.

Related work

Our work builds on many excellent existing ideas for the development of assurance cases in software, including life cycle developments (*Graydon, Knight & Strunk, 2007*), argument patterns (*Weaver, McDermid & Kelly, 2002*; *Hawkins et al., 2011*), and reviewing arguments (*Kelly, 2007*; *Yuan & Kelly, 2012*). In particular, *Graydon, Knight & Strunk (2007)* proposed a closed approach to integrating assurance into the development process by co-developing the software system and its assurance case. Hawkins et al. extensively studied the assurance aspect of the software process (*Hawkins et al., 2013*) and software evolution (*Hawkins et al., 2014*). A clear, new idea presented in this paper is the use of accountability (by dependability arguments with stakeholder identity), which allows multiple competing stakeholders to share dependability problems across the life cycle stages.

In general, assurance information needs to be kept confidential in safety-critical systems (*Bloomfield, 2013*). However, many experimental research reports have been published for researchers as in an unmanned aircraft (*Denney, Habli & Pai, 2012*), automobiles (ISO 26262) (*Ruiz, Melzi & Kelly, 2015*), autonomous vehicle and aircraft safety critical software (*Hawkins et al., 2011*), generic infusion pumps (*Kim et al., 2011*), pacemakers (*Jee, Lee & Sokolsky, 2010*), and health IT services (*Despotou et al., 2012*). In these reported areas, there exists strong regulators who validate the safety of products and services, and the development of reported assurance cases is initially motivated for their regulators. In comparison, our experience report is unique in terms of neither regulators nor safety standards. How assurance cases to be developed for improved software without regulators is an interesting open question for software practitioners (*Tokoro, 2015*).

ARGUMENTATION ARCHITECTURE

This section describes our ideas on how to use assurance cases in software-based IT systems with no strong regulator.

Sharing dependability arguments

Our initial motivation comes from the risk of miscommunication between stakeholders, such as developers and operators, who separately act in distinct phases of the life cycle. In other words, limitations discussed during software development are quite useful for operators attempting to deliver the correct services, but they are unseen at the time of operation. On the contrary, discussions at the time of operation can provide useful feedback for further development. Sharing such discussions, which are focused on dependability, is demanded extensively to improve the long-term dependability of the products and services.

Our aim in the use of assurance cases is to share dependability arguments among stakeholders throughout the life cycle. As introduced in 'What Are Assurance Cases?', the arguments are well structured and more likely to inspire confidence in stakeholders due to the supporting evidence. This would suggest that assurance cases serve as a good foundation for sharing focused knowledge and risk communications.

The argumentation architecture needs to change slightly when we attempt to apply it between (a) one stakeholder and another and (b) many stakeholders to many others. First, the top claim must represent a common goal and assumptions that are shared among all stakeholders. We decompose the common claim into sub-claims so that each stakeholder can separately lead his or her active part of the dependability arguments.

The top claim is decomposed by *stages* in the life cycle of products and services. Then, we decompose each stage claim by stakeholders if multiple acting stakeholders exist in the same stage. Each stakeholder has to provide available evidence that supports dependability claims that are part of the common goal.

Staging in the lifecycle varies from project to project, but we refer to the following stages in this paper.

- Planning stage (requirement elicitation and architecting)
- Development stage (coding and testing)
- Operation stage (service design and failure recovery)
- Evolution stage (change accommodation).

Note that the abovementioned stage decomposition is based on the open system dependability (*Tokoro, 2015*) that we proposed in the JST/DEOS project. It is distinct in its evolution stage, when all stakeholders argue for the continuous improvement of services beyond the lifetime of a single system operation.

Accountability, rebuttals, and revision

Currently, most software-based IT services run under no regulation. As described in 'What Are Assurance Cases?', the absence of strong regulators may reduce the practicality of assurance cases. This is a challenge to be avoided in practice.

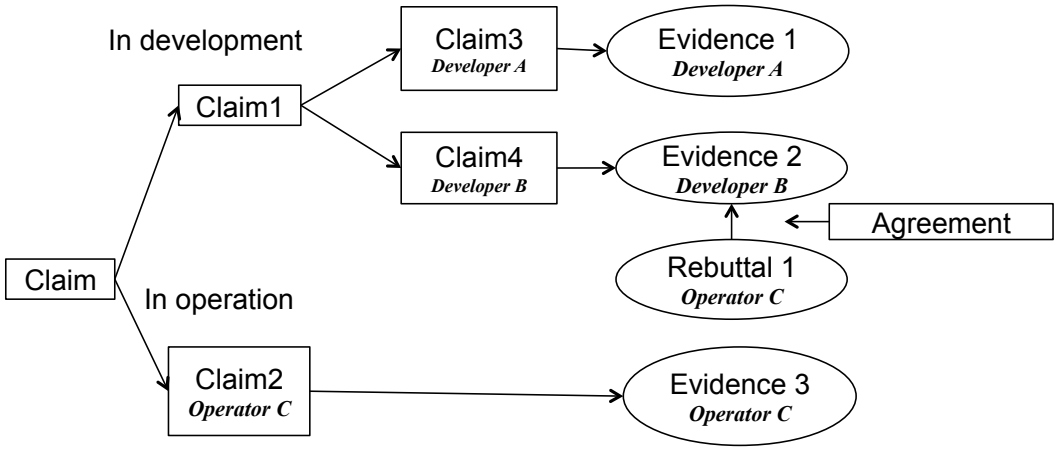

Figure 2 **Cross-Validation and Agreements on Conflicts.**

The first idea to confront this problem is the use of the concept of *accountability* (*Haeberlen, Kouznetsov & Druschel, 2007*); accountability is widely used to build trust and reputation among competing individuals and organizations by exposing failures. In the context of assurance cases, we can integrate the concept of accountability by recording the stakeholder's identity in every element of assurance cases. That is, the stakeholder's identity can be used to trace who makes a faulty claim or who gives faulty evidence when a problem occurs. In general, this results in strong incentives to avoid faulty claims and evidence.

In addition to stakeholder accountability, we include a form of *rebuttal* in the dependability arguments. In the context of assurance cases, a rebuttal is a challenge to a claim or an objection to the evidence, usually noted in the review process. Rebuttals do not remain during the assessment process because they need to be solved prior to certification. In the case of the absence of a regulator, a rebuttal is not strong enough to enforce modification. Unsolved rebuttals are considered conflicts. If the conflicts remain between stakeholders, the claim containing the rebuttal also is regarded in terms of the stakeholder agreements. Note that the rebuttals are recorded with the stakeholder's identity for accountability.

Based on the recorded rebuttals, we use cross-validation between stakeholders instead of third-party reviewers, since stakeholders in part compete with each other (e.g., a developer wants reduced costs for development, but this makes improperly developed systems a potential risk). A faulty claim often becomes a potential risk for other stakeholders. Naturally, non-rebuttal claims are regarded as approved by all stakeholders with some sharable responsibility when a problem occurs. This also leads to other incentives to make rebuttals to others' claims. Figure 2 illustrates our proposed argumentation architecture with life cycle decomposition and stakeholder identities.

More importantly, recall that our aim is to facilitate sharing dependability problems between stakeholders, not to facilitate competition among them. The developers and the operators can change software or service operations if they agree on the given rebuttals. In addition, they also are allowed to revise the related assurance cases if their practice

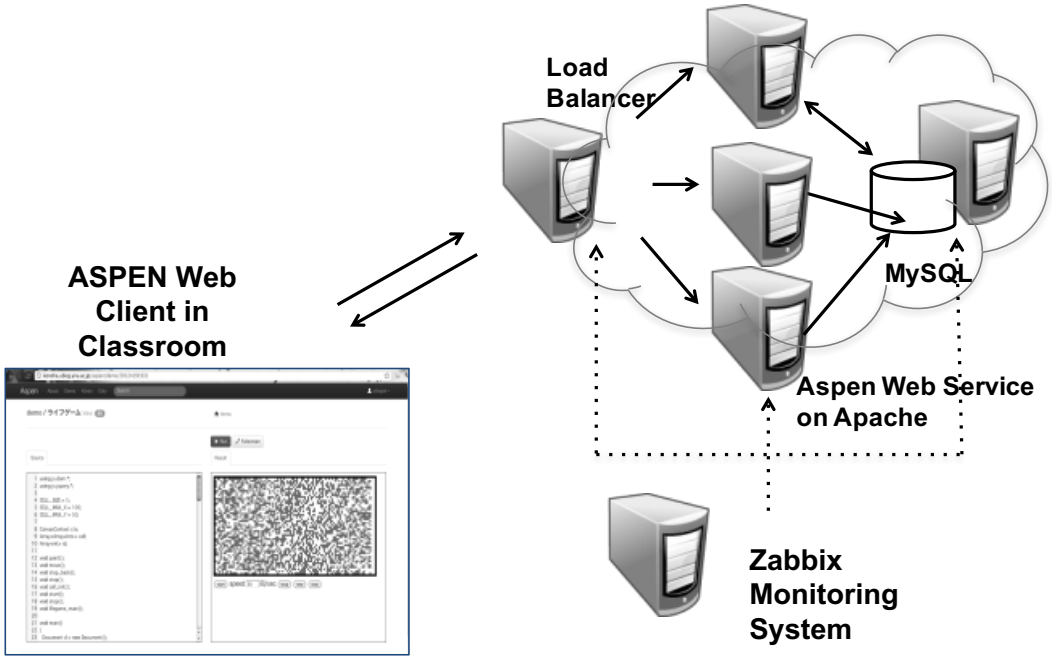

Figure 3 **Overview of Aspen system.**

is changed. This results in an iterative process that enables us to better capture the ever-changing nature of software and maintain the dependability of changing software.

Note that we assume that all revised versions of assurance cases can be maintained by a proper version control system. AssureNote, which used in this experiment, has been developed for this purpose.

EXPERIMENTAL SETUP: ASPEN PROJECT

The Aspen project is our organized experiment as a part of the JST/DEOS project (*Tokoro, 2015*) to investigate the life cycle maintenance of assurance cases without any regulators. The Aspen project includes not only the development of assurance cases across different organizations but also Aspen's software development and service operation with DEOS industrial partners. This section describes the experimental design in the Aspen project.

System and service overview

Aspen is an online education system that provides exercises on programming to students via the Web. Figure 3 provides the overview of the Aspen service. The Aspen system are not so unique and consists of multiple servers, including Apache Web servers, MySQL servers, and Zabbix monitor servers, which run on Linux operating systems. All of the software that constitutes the Aspen system are written in scripting languages such as Python and JavaScript. Due to the Agile feature of these languages, the Aspen system is updated often, even after its deployment.

Aspen is a typical Web-based system, not a safety-critical system, which is the system on which assurance cases mainly have been developed so far. However, Aspen involves

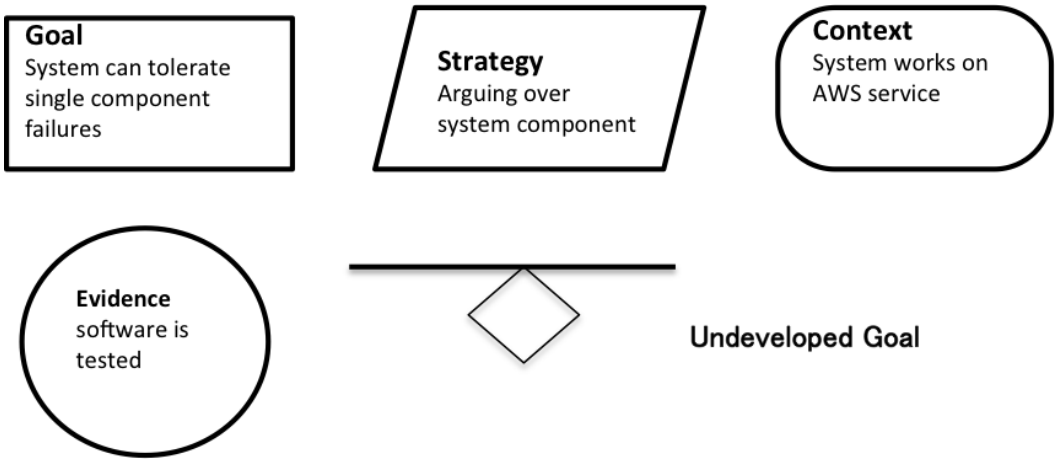

Figure 4 **GSN elements.**

several dependability attributes (*Avizienis et al., 2004*) that are commonly required in software-based IT services:

- Availability: the service always will be available to the users (i.e., students).
- Reliability: no hardware or software failures occur during provision of the service.
- Integrity: programming assignments supplied by the owner do not disappear.
- Privacy: personal information is not disclosed to unauthorized parties.

Documentation and tool supports

In the Aspen project, we use *Goal Structuring Notation* (*Kelly & Weaver, 2004*) to share assurance cases among stakeholders. GSN is a standard argumentation notation of assurance cases in safety-critical industries. GSN consists of four principal elements, goal (depicted as a rectangle), strategy (parallelogram), evidence (oval), and context (rounded rectangle), as shown in Fig. 4. The goal element is used to state a claim that a system certainly has some desirable properties, such as safety and dependability. The evidence element is some piece of material to support that the linked claim is true. The goal without linked evidence is called undeveloped.

As in the CAE notation, a goal is decomposed into sub-goals until a point is reached where claims can be supported by direct reference to available evidence. The strategy element is used to state a reason for claim decomposition. The context element is used to state an assumption (the system scope and the assumed properties). There is no special support for stating a rebuttal. We regard the context element linked to the evidence as the rebuttal element.

In the experiment, we used AssureNote (*Shida et al., 2013*), a Web-based authoring tool that allows multiple users to share a GSN document via the Web. Figure 5 is a screenshot of AssureNote. In AssureNote, all GSN elements are automatically recorded with the user identity under the version control of GSN elements.

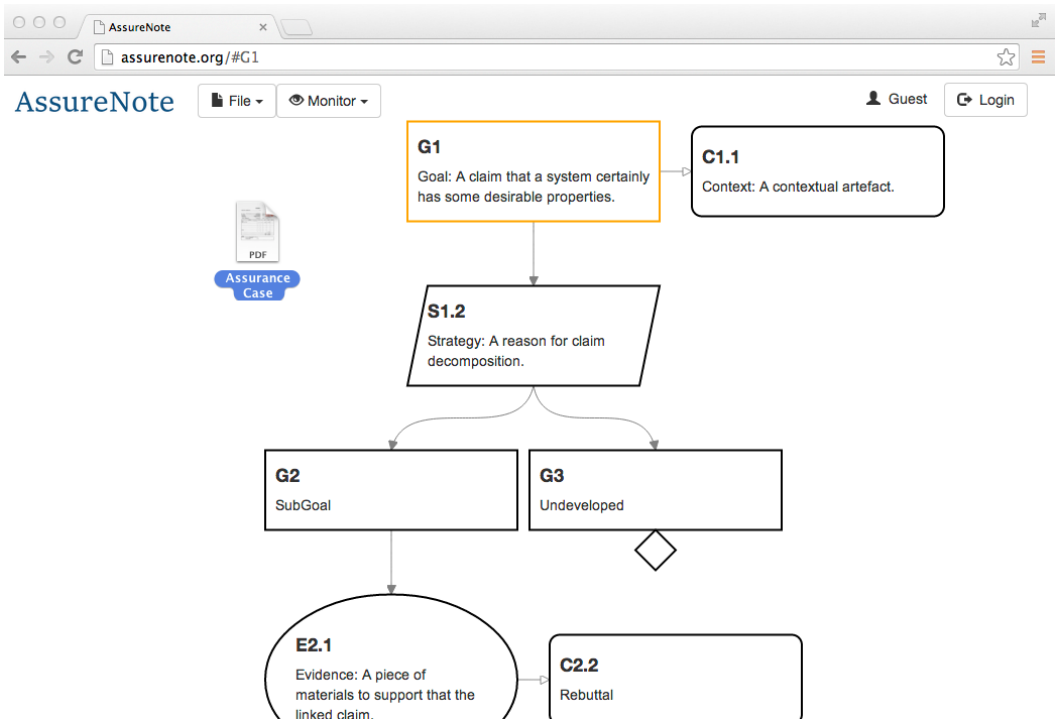

Figure 5 **AssureNote: a Web-based GSN authoring tool.**

Stakeholders

Another main aim of the Aspen project is to examine the effect of assurance cases in the absence of a strong regulator. As described in 'Argumentation Architecture', we need to set up some competing relationship between stakeholders.

All stakeholders were selected from different institutes. First, we contracted two different firms separately: one that took charge in software development and the other in service operation. Second, we asked an instructor who had adopted the Aspen system in her classroom to join the experiment as a stakeholder. The author plays a coordinator role, not a regulator role throughout the whole stage. The stakeholders involved in the Aspen project are listed below.

- Owner: The author
- Developer: A programmer working for a software development firm
- Operator: A system engineer with more than 10 years' experience with Web-based service operations
- User: An instructor who use the Aspen in her classroom.

In this paper, we identify stakeholders by role names–Owner, Developer, Operator, and User–for readability. However, on the assurance cases, stakeholders are identified by a personal name for the sake of accountability.

Development procedure

In the experiment, we attempt to develop assurance cases in parallel with the development and service operation of the Aspen system. All assurance cases and other communications between stakeholders are written and spoken in Japanese.

In the planning stage, the owner defines the top claim, which includes dependability goals and assumptions, for which the Aspen system is assumed to deliver correct services for the users. Following the planning stage, we undertake the Aspen system with the following procedures:

- The developer claims that the developed system meets the owner's dependability goals with supporting evidence.
- The operator, the user, and the owner review the developer's claims and agree or disagree.
- The operator, based on the developed system, claims that the system operation meets the owner's dependability goals with supporting evidence.
- The developer, the user, and the owner review the operator's claims and come to agreements on the conflicted claims.
- Any stakeholder can revise the assurance cases if they contain any insufficiencies or flaws.

As shown above, we focus on dependability-related issues and avoid general issues with software implementations and operating procedures. When we handle the disagreement, we ask all stakeholders to meet together at the same place to agree on the conflicts.

ASPEN CASES

The Aspen cases were developed with the method that we proposed in the 'Argumentation Architecture' and 'Experimental Setup: Aspen Project' sections. This section reports how the arguments are organized with a fragment of GSNs and excerpted from the developed assurance cases.

Overview

First, we overview the statistics of the Aspen cases. As we described in 'Stakeholders1', the Aspen cases are written in GSN. We first gave the top goal, which was decomposed into the Development stage, the Service stage, and the Evolution stage with common assumptions. The GSNs were revised 40 times throughout the Aspen project. Here we use #n to represent the nth revision. The GSN document grew from four elements (at #1) to 134 elements (at #40). We identify the major revisions as follows:

- #2 Initial requirement
- #12 Development
- #22 Development agreement
- #27 Service design

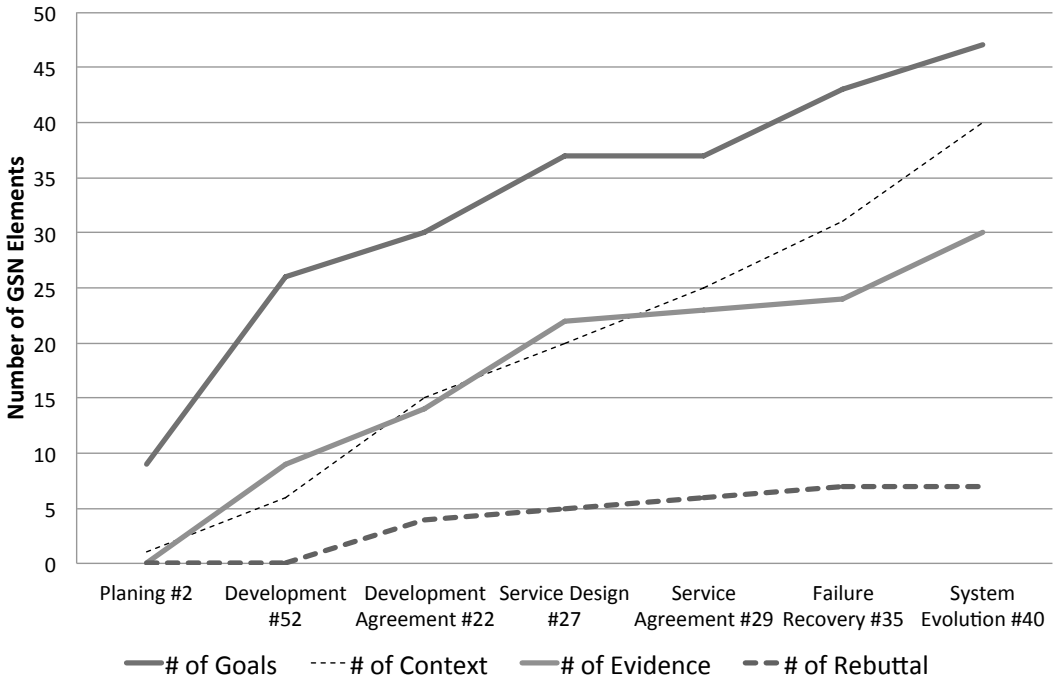

Figure 6 **Growth of assurance cases in GSN elements.**

- #29 Service agreement
- #35 Failure recovery
- #40 Service evolution.

Figure 6 shows the growth of the Aspen cases in the number of GSN elements: goals, contexts, evidences, and rebuttals. The increase in contexts suggests that the reduced ambiguity of the goals and the increase of evidence reduce the reduced uncertainty in their dependability claims.

In the remainder of this section we close up the development agreement (#2–#20), the service agreement (#27–#29), the failure recovery (#35), and the system evolution (#40).

Development agreement

The developer started the argument with the claim "Aspen is dependable" and decomposed it into sub-claims by dependability attributes (cf. Aspen is available, integral, safe, etc.). The forms of evidence that the developer provided were mostly other external experience reports (collected from the Internet). Some goals included a lack of evidence. Note that a lack of evidence does not directly imply a lack of dependability, but it does indicate uncertainty about dependability.

Figure 7 illustrates the fragment with the developer's claim that the software is integral in data storage. One could consider that Evidence E22.1 was not reasonable evidence to support Claim G22. Likewise, the operator pointed out a risk of hardware failure in the disk storage. However, there was not enough time to change the storage system. Instead,

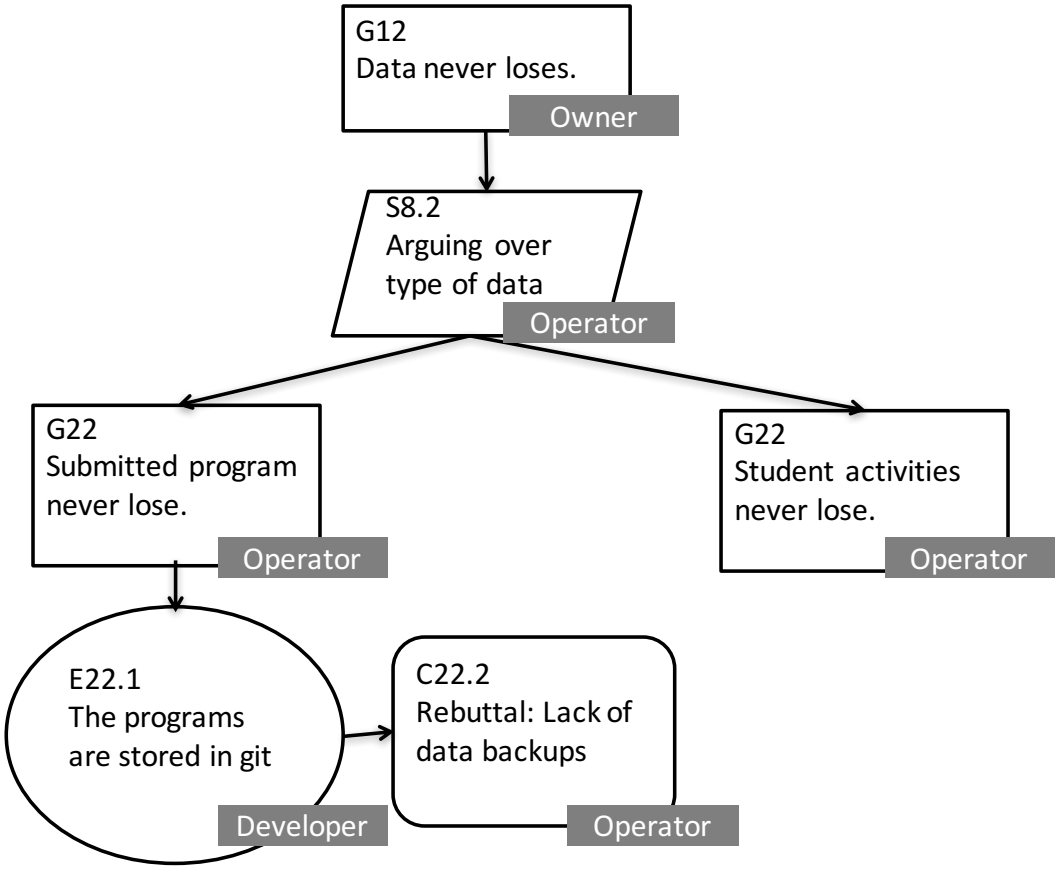

Figure 7 **Example of a rebuttal by the operator's review.**

the operator agreed to fault tolerance at the time of operation. In the end, the operator made nine rebuttals to the developer's claim prior to the operations.

Service agreement

In the operation stage, we focus only on fault mitigation and failure recovery. The operator led the arguments of this part using the fault-avoidance pattern, which consists of the following two parts:

- The symptom of a failure (an error) is detectable (by monitoring).
- The detected symptom is mitigated before the service stops.

The completeness of fault-avoidance analysis is important but not pursued in the experiment, in part because we want to evaluate the iterative updates of assurance cases during the operations.

Figure 8 is a fragment of assurance cases arguing about the server's availability. Note that some embedded parameters are used for operation scripts (*Kuramitsu, 2013*). One could consider the given evidence questionable, but the user and the owner trusted the operator's claim without any doubt due to their limited knowledge. They did not make any rebuttals against the operator, except for some help-desk support in case of service failures.

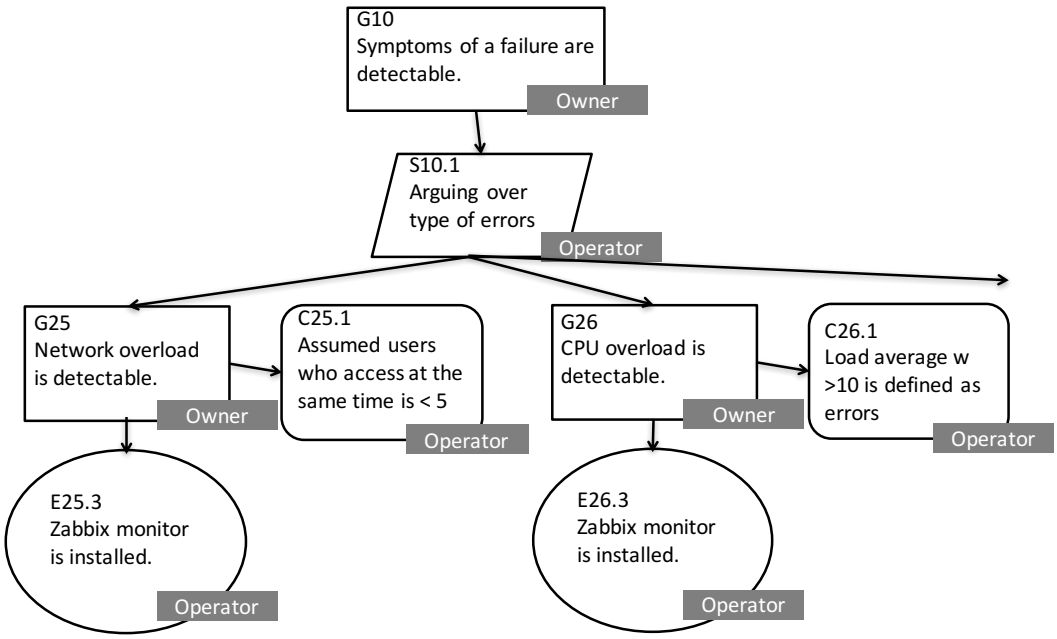

Figure 8 **Example of arguments on failure avoidance in operation.**

Failure recovery

We encountered several service failures while running the Aspen system. Since unexpected failures imply faulty arguments, the operator or the developer needs to revise the assurance cases.

Here, we highlight a service failure that occurred in the middle of the classroom. This failure appeared only when students used Aspen in the classroom. The system monitor indicated that the server's operating system was unexpectedly down. At first, the operator suspected that there were unknown bugs in the Aspen system. Unfortunately, the developer found some bugs that seemed related to the service failure. If there were no assurance cases, they used an incorrect method to recover the service failure.

In reviewing assurance cases, the claim "the servers can scale out" was overconfident. The operator never tested the server in any network traffic settings. However, no one pointed out that this claim seemed faulty. After the server problem occurred, the operator recognized that the scale-out setting was incapable of handling simultaneous access by 40 students in the classroom. The instructor strongly requested that the operator should increase the servers as the operator had claimed in the assurance cases.

Another serious dependability problem was found later in the context of the top goal, which described common assumptions about the Aspen system. Originally, the Aspen system was assumed to allow 100 students to submit their assignments from home computers. Based on this assumption, the maximum simultaneous access was estimated to be at most five connections. The number of students in the classroom was fewer than 100 students, but the density of access traffic differed from the estimated patterns. In other

words, the top-goal assumption was a sort of bug, which resulting in rechecking all given evidence.

System evolution

The Aspen project ran for two years. In the second year, the Aspen system evolved with the following major updates:

- The adoption of Moodle, an open-source solution to reduce in-house development
- A system movement to Amazon Web Services (a standard cloud computing platform).

These system updates were not random but derived from several dependability goals and potential risks that were argued in the revised assurance cases. More importantly, all stakeholders, including those who were newly involved in the Aspen projects, could share the justification of these system updates in terms of dependability improvements. Compared to unstructured e-mail communications, the GSN-formed arguments made it easier to trace dependability problems that actually happened or were expected in the first year.

LESSONS LEARNED AND DISCUSSIONS

This section describes lessons we learned throughout the Aspen case in the form of answering research questions.

(Question 1). Do non-experts join dependability arguments with GSNs?

Yes. For all of the participants, the concept of assurance cases was totally new, and they were suspicious of the assurance cases' benefits. We gave them about a 1-hour technical lecture on the notation of GSN (not the concept of assurance cases itself.) This short lecture and a Web-based GSN viewer were all we prepared for participants to read GSNs for dependability arguments. Note that organizing dependability arguments is a known difficulty according to *Bloomfield & Bishop (2010)*. We prepared several simple argument patterns, which are described in 'Experimental Setup: Aspen Project'. Due to the argument patterns, arguments grew in a well-structured way.

(Question 2). What was the hardest part in the development of continuously revised assurance cases?

Collecting acceptable evidence for their dependability claims was difficult and costly. In part because the assurance cases were new for both the developers and the operators, they did not prepare any forms of evidence in their process. For example, the developers performed many software tests as usual, but these test results were far from those requested in the non-functional arguments. Neither the developers nor the operators wanted to spend extra resources for evidence, while competing stakeholders always requested much more evidence. Perhaps dependability guidelines are necessary to reach agreement on proper forms and levels of evidence.

(Question 3). Did stakeholders really validate others' claims?

Yes, they were willing to review and tried to validate them. Rebuttal serves an enforced communication vehicle between stakeholders. One reason for their many responses was that

we introduced some penalty (e.g., sharing responsibilities in failure cases) for no-rebuttal claims. This penalty seemed unrealistic but forced all stakeholders to find faulty claims.

The activity of cross-validation can be measured by the number of revisions and the growth of GSN elements:

- 10 revisions for the development claims
- two revisions for the operation claims.

The development claims were revised more often than the operation claims. The difference comes mainly from the stakeholders' expertise. The users were likely to trust the operator's claims and evidence.

(Lesson 5). Is the stakeholder cross-validation strong enough?

The answer is that the strength depends on stakeholders' expertise. The dependability arguments between developers and operators seemingly worked. System failures that occurred in the Aspen system were caused by bad operations, not a flaw in the developed software. On the other hand, the users lacked the knowledge of service operations, and they could not point out any weaknesses in the operators' claims documented in the assurance cases. However, the faulty claims were costly in cases of system failures because the users strongly requested the fulfillment of the operator's claims as documented. We expected that the GSN documentation became a strong incentive for operators to avoid faulty claims. In reality, the costs of collecting evidence overwhelmed such incentives.

(Question 6). Is the development of assurance cases useful in software?

The answer is positively yes. Our finding in the case analysis of failure recovery suggests that the developed assurance cases make it easier to find dependability problems throughout structured arguments in GSNs. Even if we are not able to avoid the service failure the first time, the dependability problems can be transferred clearly in the redevelopment phase. Since transferring dependability problems across the organization is a missing part in dependability engineering, the contentiously revised assurance cases can bridge the missing part.

CONCLUSION

Recently, assurance cases have received much attention in the field of software-based computer systems and IT services. However, software often changes, and there are no strong regulations for software. These facts make the use of software assurance cases more difficult. We proposed a development method for assurance cases with stakeholder cross-validation at every stage of the system life cycle, including in operation and service recovery in a case of system failure.

This paper reported our practitioner experience based on the Aspen project. As we expected, stakeholders competing with each other were well motivated to find the faulty claims of the others in the GSN documents. This improves the quality of dependability arguments. Unfortunately, the stakeholder cross-validation was not able to avoid simple system failures in advance. The case analysis of failure recovery with assurance cases suggests that dependability problems can be easily transferred in a structured way across

stakeholders and organizations. Since transferring dependability problems is an important missing part of long-term dependability, continuously revised assurance cases may serve as a potential new approach to the long-term improvement of service dependability. In future work, we will investigate the dependability effect of assurance cases from a longer-term perspective.

ACKNOWLEDGEMENTS

The authors thank to all the DEOS research members, especially Mario Tokoro, Shuichiro Yamamoto, Yutaka Matsuno, Midori Sugaya, Yoshiki Kinoshita, Makoto Takeyama, and Makoto Yashiro.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

This work was supported by the JST/CREST grant "Dependable Embedded Operating Systems for Practical Uses." The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Grant Disclosures

The following grant information was disclosed by the author:
JST/CREST.

Competing Interests

The author declares there are no competing interests.

Author Contributions

- Kimio Kuramitsu conceived and designed the experiments, performed the experiments, analyzed the data, contributed reagents/materials/analysis tools, wrote the paper, prepared figures and/or tables, performed the computation work, reviewed drafts of the paper, funding.

Data Availability

The following information was supplied regarding data availability:

The raw data has been supplied as a Supplemental File.

Supplemental Information

Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj-cs.101#supplemental-information.

REFERENCES

Avizienis A, Laprie J-C, Randell B, Landwehr CE. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* **1(1)**:11–33 DOI 10.1109/TDSC.2004.2.

Bloomfield R. 2013. Open assurance. *Security Privacy, IEEE* **11(5)**:3–4 DOI 10.1109/MSP.2013.119.

Bloomfield RE, Bishop PG. 2010. Safety and assurance cases: past, present and possible future—an adelard perspective. In: *Making systems safer—proceedings of the eighteenth safety-critical systems symposium, Bristol, UK, February 9–11, 2010*. 51–67 DOI 10.1007/978-1-84996-086-1-4.

Denney E, Habli I, Pai G. 2012. Perspectives on software safety case development for unmanned aircraft. In: *Proceedings of the 42nd annual IEEE/IFIP international conference on dependable systems and networks (DSN 2012)*. Piscataway: IEEE.

Despotou G, White S, Kelly T, Ryan M. 2012. Introducing safety cases for health IT. In: *Proceedings of the 4th international workshop on software engineering in health care*. Piscataway: IEEE, 44–50.

Graydon P, Habli I, Hawkins R, Kelly T, Knight J. 2012. Arguing conformance. *IEEE Software* **29(3)**:50–57 DOI 10.1109/MS.2012.26.

Graydon PJ, Knight JC, Strunk E. 2007. Assurance based development of critical systems. In: *Dependable systems and networks, 2007. DSN'07. 37th annual IEEE/IFIP international conference on*. Piscataway: IEEE, 347–357.

Haeberlen A, Kouznetsov P, Druschel P. 2007. PeerReview: practical accountability for distributed systems. In: *Proceedings of twenty-first ACM SIGOPS symposium on operating systems principles, SOSP '07*. New York: ACM, 175–188 DOI 10.1145/1294261.1294279978-1-59593-591-5.

Hawkins R, Clegg K, Alexander R, Kelly T. 2011. Using a software safety argument pattern catalogue: two case studies. In: *Proceedings of the 30th international conference on computer safety, reliability, and security, SAFECOMP'11*. Berlin: Springer-Verlag, 185–198.

Hawkins R, Habli I, Kelly T, McDermid J. 2013. Assurance cases and prescriptive software safety certification: a comparative study. *Safety Science* **59**:55–71 DOI 10.1016/j.ssci.2013.04.007.

Hawkins R, Miyazawa A, Cavalcanti A, Kelly T, Rowlands J. 2014. Assurance cases for block-configurable software. In: *Proceedings of the 33rd international conference on computer safety, reliability, and security—volume 8666, SAFECOMP 2014*. New York: Springer-Verlag New York, Inc., 155–169 DOI 10.1007/978-3-319-10506-2-11978-3-319-10505-5.

Jee E, Lee I, Sokolsky O. 2010. Assurance cases in model-driven development of the pacemaker software. In: *Leveraging applications of formal methods, verification, and validation—4th international symposium on leveraging applications, ISoLA 2010, Heraklion, Crete, Greece, October 18–21, 2010, proceedings, part II. Lecture notes in computer science*, vol. 6416. Berlin Heidelberg: Springer, 343–356.

Kelly T. 2007. Reviewing assurance arguments-a step-by-step approach. In: *Workshop on assurance cases for security-the metrics challenge, dependable systems and networks (DSN). Available at https://www-users.cs.york.ac.uk/tpk/dsnworkshop07.pdf* .

Kelly T, Weaver R. 2004. The goal structuring notation—a safety argument notation. In: *Proceedings of the workshop on assurance cases, international conference on dependable systems and networks. Available at https://www-users.cs.york.ac.uk/tpk/dsn2004.pdf* .

Kim B, Ayoub A, Sokolsky O, Lee I, Jones P, Zhang Y, Jetley R. 2011. Safety-assured development of the GPCA infusion pump software. In: *Proceedings of the ninth ACM international conference on embedded software, EMSOFT '11*. New York: ACM, 155–164 DOI 10.1145/2038642.2038667978-1-4503-0714-7.

Kuramitsu K. 2013. D-script: dependable scripting with DEOS process. In: *IEEE symposium on software reliability engineering workshops (ISSREW)*. Piscataway: IEEE, 326–330.

Ruiz A, Melzi A, Kelly T. 2015. Systematic application of ISO 26262 on a SEooC: support by applying a systematic reuse approach. In: *Proceedings of the 2015 design, automation & test in europe conference & exhibition, DATE '15*. San Jose: EDA Consortium, 393–396.

Shida S, Uchida A, Ishii M, Ide M, Kuramitsu K. 2013. Assure-It: a runtime synchronization tool of assurance cases. In: *Safecomp 2013 fastabstract. Available at https://hal.archives-ouvertes.fr/hal-00926410/* .

Tokoro M (ed.) 2015. *Open systems dependability: dependability engineering for ever-changing systems*. Second edition. Boca Raton: CRC Press.

Weaver R, McDermid J, Kelly T. 2002. Software safety arguments: towards a systematic categorisation of evidence. *International System Safety Conference, Denver, CO*.

Yuan T, Kelly T. 2012. Argument-based approach to computer system safety engineering. *International Journal of Critical Computer-Based Systems* **3(3)**:151–167 DOI 10.1504/IJCCBS.2012.050295.