Title: Pitfalls and Countermeasures in Software Quality Measurements and Evaluations

Author: Hiroshi Washisaki

Affiliation: Waseda University / National Institute of Informatics / SYSTEM INFORMATION

Contact: washizaki@waseda.jp, Waseda University, 3-4-1 Ohkubo, Shijuku-ku, 169-8555 Tokyo, JAPAN

Abstract
This chapter discusses common pitfalls and their countermeasures in software quality measurements and evaluations based on research and practical achievements. The pitfalls include negative Hawthorne effects, organization misalignment, uncertain future, and self-certified quality. Corresponding countermeasures include goal-oriented multidimensional measurements, alignment visualization and exhaustive identification of rationales, prediction incorporating uncertainty and machine-learning based measurement improvement, and standard/pattern-based evaluation.

Keywords
Software measurement, software metrics, software quality, goal-orientation, GQM, ISO/IEC 25000, SEMAT, software patterns

1. Introduction

Measurements to evaluate quality are essential to specify, manage, and improve the quality of product in software developments. However, a development project may become worse, such as a misleading conclusion, if the measurement program is not properly adopted. This chapter discusses common pitfalls and their countermeasures in software quality measurements and evaluations based on research and practical achievements at the Global Software Engineering Laboratory (PI: Prof. Hironori Washizaki) of Waseda University in collaboration with many software companies [GSE]. Table 1 summarizes the specific pitfalls addressed and their corresponding countermeasures.

Table 1. Pitfalls and countermeasures in quality measurements and evaluations

Pitfall	Countermeasure
Negative Hawthorne effects	Goal-orientation
	Multidimensional measurements

Organization misalignment	Visualization of relationships among organizational goals, strategies, and measurements
	Exhaustive identification of rationales
Uncertain future	Prediction incorporating uncertainty
	Measurement program improvement by machine learning
Self-certified quality	Standard-based evaluation
	Pattern-based evaluation

2. Pitfall: Negative Hawthorne effects

Measurements are so powerful that they drive people's behavior. This phenomenon is known as the Hawthorne effect (or the observer effect). It was derived from famous extensive productivity research conducted at the Western Electric/AT&T Hawthorne plant between 1924 and 1932, which confirmed that whatever management paid attention to and measured improved [Linda06].

Thus, measurements should be carefully employed in software development and quality management to help stakeholders focus on what is truly important. Otherwise, quality may improve with regard to the measurements, while quality of aspects not measured may decline at the expense of the overall quality. This is a common symptom when a measurement program is build based on available data or what is of most interest to the metrics engineer [Linda06].

There are at least two countermeasures to prevent negative Hawthorne effects: goal-orientation and multidimensional measurements. The former contributes to clarifying the focus and corresponding measurements, while the latter incorporates various aspects to ensure total quality.

2.1. Countermeasure: Goal-orientation

Goal-orientation is a generic term for approaches involving goal setting and variable derivation in a top-down manner. Goal-Question-Metrics (GQM, hereafter) is a goal-oriented approach to define a measurement program from the top goal [Basili02]. GQM takes the following three steps to define a measurement program [Linda06]:

(1) Identify the Goal for the product/process/resource from the viewpoint of the actual "customer" of the measurement program.

(2) Determine the Questions that characterize how achievement of the goal is assessed.

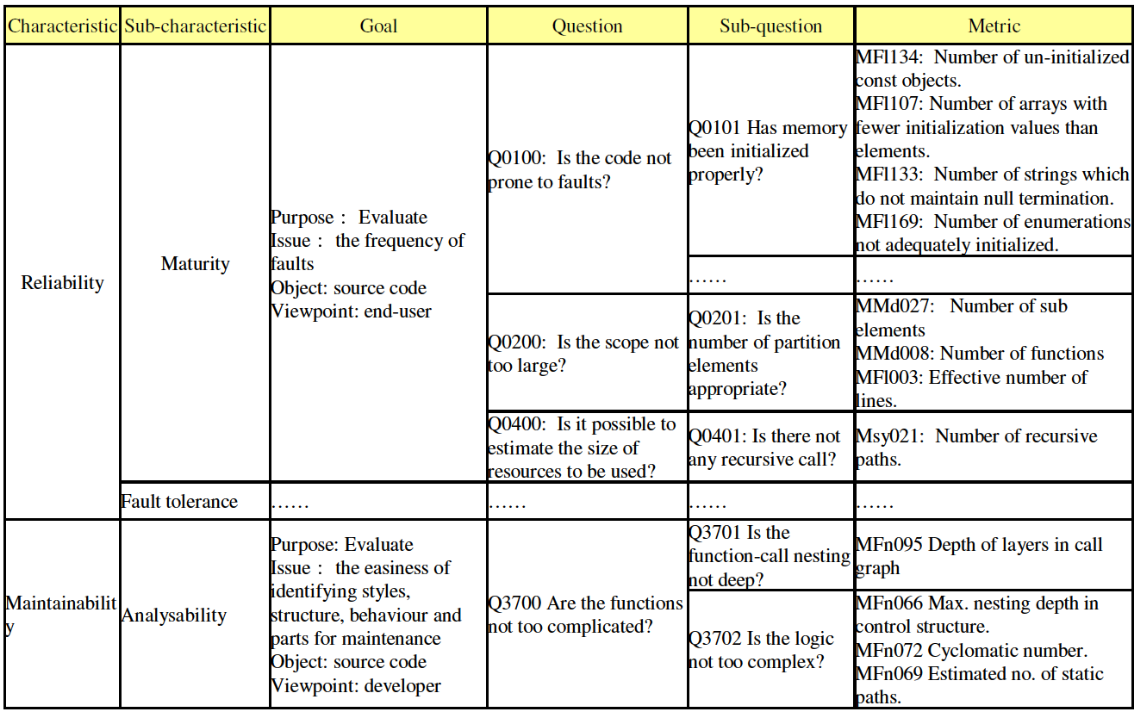
(3) Define the Metrics that quantitatively answer each question.

GQM is particularly useful to capture the nature of software quality since quality is an abstract and inherently invisible concept. Applying GQM makes it much easier to focus on what is truly important for the "customer" and build a measurement program based on the goal instead of available data. Consequently, GQM may mitigate the possibility of negative Hawthorne effects and turn them into positive ones.

There are many successful cases of GQM adoption in software quality measurements, including:

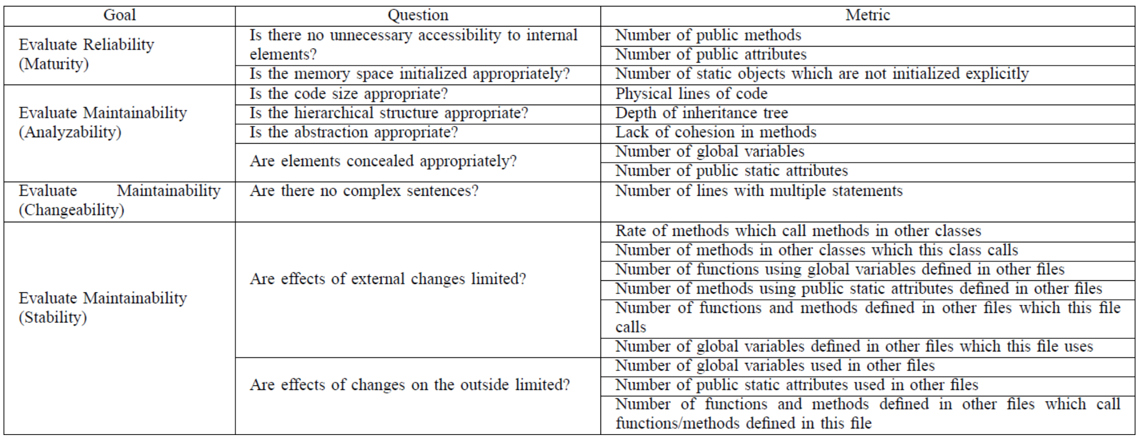
- OGIS-RI Co. and GSE jointly built a static analysis and measurement tool called Adqua to evaluate the quality of embedded program source codes written in C language. Measurements in Adqua have been identified using GQM and the ISO9126-1 quality model [Washizaki07]. The GQM model consists of ten goals to evaluate quality sub-characteristics, 47 questions, 101 sub-questions, and 236 metrics. Figure 1 shows an excerpt of the model. For example, several language-independent questions (e.g., Q3700) help to evaluate how easily the source code is analyzed. Because Q3700 is quite abstract and difficult to measure directly, it is divided into several sub-questions, including Q3701 and Q3702. Finally, metrics are assigned to each sub-question, allowing useful data to assess the goal to be obtained. The single metric, MFn095, is assigned to Q3701, and three metrics, MFn066, MFn072 and MFn069, are assigned to Q3702. Thus, the source code quality can be evaluated via quality-sub-characteristic units from the measurement. By conducting experiments targeting several embedded programs, it has been confirmed that Adqua can be used effectively to evaluate programs for reliability, maintainability, reusability, and portability. Adqua has been used to evaluate embedded programs in Japan successfully for over five years.
- GSE, OGIS-RI Co. and Yamaha Corporation extended the above-mentioned tool to evaluate the reusability of C language program source code more precisely by adopting GQM to identify a set of metrics [Washizaki12]. By applying the tool to ten actual projects involving the development of existing software modifications and adoptions, it has been confirmed that these metrics effectively reflect and estimate the magnitude of necessary effort to reuse a target.
- GSE and FUJITSU CONNECTED TECHNOLOGIES investigated the impact of software transfer from one development organization to another organization on software maintainability and reliability by introducing the concept of “origins” as files’ creation and modification histories [Sato13]. They adopted GQM to specify necessary measurements to determine maintainability and reliability under the context of software transfer. Figure 2 shows the GQM model constructed by setting goals to evaluate specific quality characteristics. Measurements are from the static analysis tool Adqua. By analyzing two open source projects, OpenOffice and VirtualBox, which were each developed by a total of three organizations, the results show that files modified by multiple organizations or developed by later organizations tend to be faultier due to the increase in complexity and modification frequency. The concept of origins as well as the measurements specified have been utilized to investigate the impact of individual developer’s experience on the software quality [Ando15][Tsunoda17] and to support the overall comprehension of large programs with long histories involving transfers [Ishizue16]. Figure 3

shows an example of “Origin City”, which represents the measurement values for files with different origins in the form of stacked 3D buildings [Ishizue16].



Characteristic	Sub-characteristic	Goal	Question	Sub-question	Metric
Reliability	Maturity	Purpose : Evaluate Issue : the frequency of faults Object: source code Viewpoint: end-user	Q0100: Is the code not prone to faults?	Q0101 Has memory been initialized properly?	MFl134: Number of un-initialized const objects. MFl107: Number of arrays with fewer initialization values than elements. MFl133: Number of strings which do not maintain null termination. MFl169: Number of enumerations not adequately initialized.
				……	……
			Q0200: Is the scope not too large?	Q0201: Is the number of partition elements appropriate?	MMd027: Number of sub elements MMd008: Number of functions MFl003: Effective number of lines.
			Q0400: Is it possible to estimate the size of resources to be used?	Q0401: Is there not any recursive call?	Msy021: Number of recursive paths.
	Fault tolerance	……	……	……	……
Maintainability	Analysability	Purpose: Evaluate Issue : the easiness of identifying styles, structure, behaviour and parts for maintenance Object: source code Viewpoint: developer	Q3700 Are the functions not too complicated?	Q3701 Is the function-call nesting not deep?	MFn095 Depth of layers in call graph
				Q3702 Is the logic not too complex?	MFn066 Max. nesting depth in control structure. MFn072 Cyclomatic number. MFn069 Estimated no. of static paths.

Figure 1: GQM model to evaluate the quality of C programs (excerpt) [Washizaki07]



Goal	Question	Metric
Evaluate Reliability (Maturity)	Is there no unnecessary accessibility to internal elements?	Number of public methods
		Number of public attributes
	Is the memory space initialized appropriately?	Number of static objects which are not initialized explicitly
Evaluate Maintainability (Analyzability)	Is the code size appropriate?	Physical lines of code
	Is the hierarchical structure appropriate?	Depth of inheritance tree
	Is the abstraction appropriate?	Lack of cohesion in methods
	Are elements concealed appropriately?	Number of global variables
		Number of public static attributes
Evaluate Maintainability (Changeability)	Are there no complex sentences?	Number of lines with multiple statements
Evaluate Maintainability (Stability)	Are effects of external changes limited?	Rate of methods which call methods in other classes
		Number of methods in other classes which this class calls
		Number of functions using global variables defined in other files
		Number of methods using public static attributes defined in other files
		Number of functions and methods defined in other files which this file calls
		Number of global variables defined in other files which this file uses
	Are effects of changes on the outside limited?	Number of global variables used in other files
		Number of public static attributes used in other files
		Number of functions and methods defined in other files which call functions/methods defined in this file

Figure 2: GQM model to evaluate the reliability and maintainability of programs [Sato13]

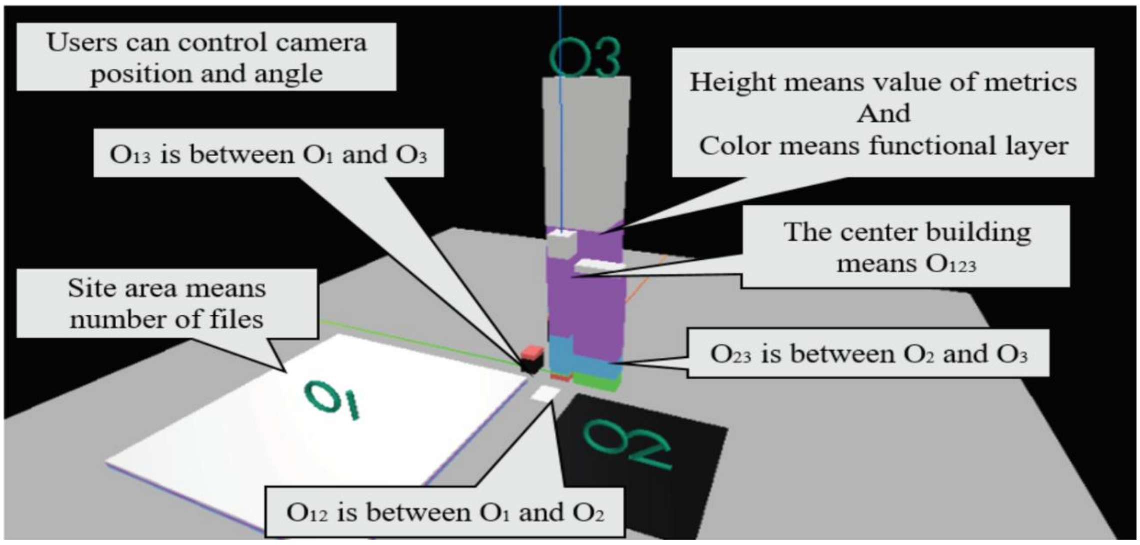

Users can control camera position and angle
O_{13} is between O_1 and O_3
O3
Height means value of metrics And Color means functional layer
The center building means O_{123}
Site area means number of files
O1
O_{23} is between O_2 and O_3
O2
O_{12} is between O_1 and O_2

Figure 3: Example of Origin City [Ishizue16]

2.2. Countermeasure: Multidimensional measurements

Beside goal-oriented measurements, it is also important to measure and evaluate targets multidimensionally to cover various aspects and ensure total quality since any feature may have side effects or unintended quality characteristics. A typical example is the trade-off between maintainability and performance (i.e., time behavior); a program tuned for computing performance may be less comprehensible for human developers.

Multidimensional measurements and evaluations are particularly crucial to grasp the total quality of software. For example in [Washizaki07], the GQM model and specified measurements successfully cover most major quality characteristics to measure and evaluate embedded C programs multidimensionally.

A multidimensional evaluation may reveal trends and tendencies of software quality in detail. For example, GSE and Yahoo Japan jointly built a dashboard (Fig. 4) to visualize multiple measurement results based on the underlying GQM model to support decision-making [Nakai14]. Visualizing the multidimensional measurement results allows users to easily grasp possible side effects and the overall total quality.

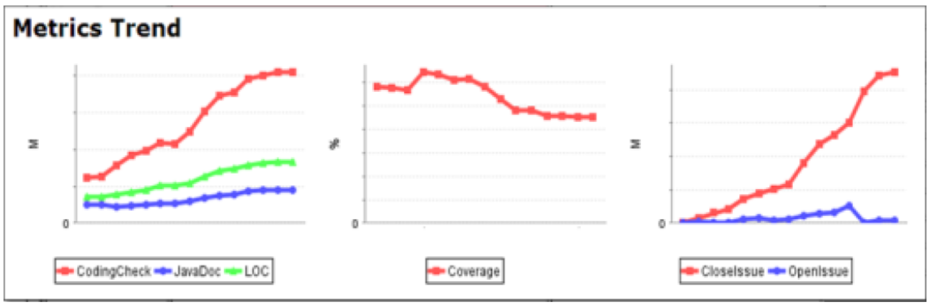


Figure 4: Dashboard visualizing multiple measurement results (excerpt) [Nakai14]

A multidimensional evaluation is also useful to capture the software development process and project status. For example, the SEMAT (Software Engineering Methods and Theory) initiative proposed a framework, the SEMAT Kernel, to reason about the progress of stakeholders and the health of their endeavors in terms of six different but mutually dependent concerns: opportunity, stakeholders, requirements, software system, work, team, and way of working [Jacobson12]. These concerns are called "alphas". Alphas are essential elements of a software engineering endeavor, and their progress and health must be assessed. Figure 5 shows the relationships among these alphas. Through this framework, stakeholders can capture a project's status multidimensionally rather than through work products (such as documents).

In the SEMAT Japan Chapter, a working group of ITA (Information Technology Alliance, which is an association of Japanese information technology companies) analyzed existing project failure cases using the SEMAT Kernel. They then identified root causes and countermeasures of these cases efficiently from wider viewpoints. Figure 6 shows an example of root cause analysis results by analyzing a failure case through the relationships among alphas such as opportunity and stakeholders.

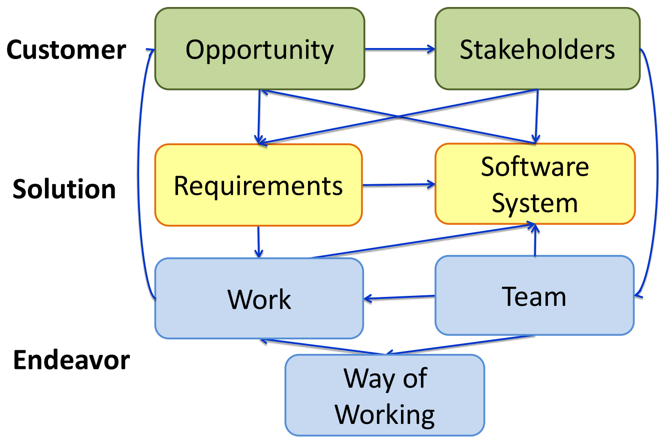


Figure 5: Relationships among SEMAT alphas

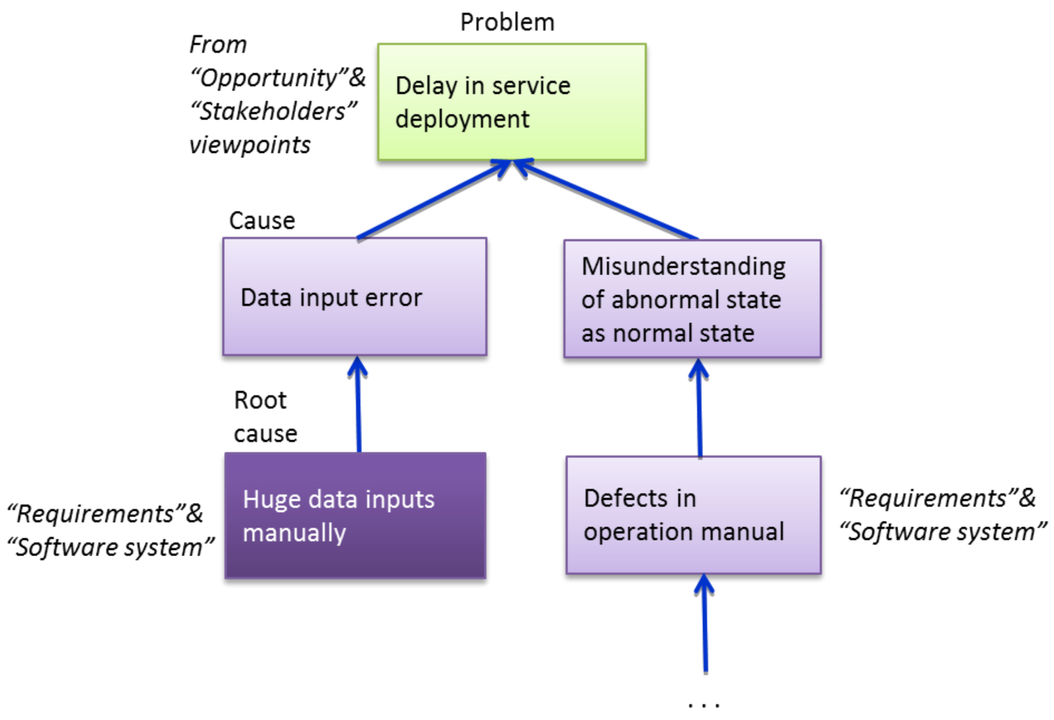

Problem
From "Opportunity"& "Stakeholders" viewpoints
Delay in service deployment
Cause
Data input error
Misunderstanding of abnormal state as normal state
Root cause
"Requirements"& "Software system"
Huge data inputs manually
Defects in operation manual
"Requirements"& "Software system"
. . .

Figure 6: Root cause analysis based on SEMAT alphas

3. Pitfall: Organization misalignment

A measurement program must be fully aligned with organizational goals and strategies; otherwise, even if GQM is adopted to clarify measurement goals and corresponding metrics, these goals and metrics may be useless from the organizational management's point of view since their contributions to the organization may be unclear without coherent rationales. To prevent such misalignments, at least two countermeasures are possible: visualization of relationships among organizational goals, strategies, and measurements and exhaustive identification of fact-based rationales.

3.1. Countermeasure: Visualization of relationships among organizational goals, strategies, and measurements

By visualizing the relationships among organizational units, goals, strategies, and measurements, whether (or not) the measurement program is consistent and fully aligned with the organization becomes clear. GQM+Strategies, which was developed by Basili, et al., is an extension of GQM that aligns and assesses the organizational and business goals at each organizational level to the overall strategies and goals of the organization [Basili10][Basili14]. Figure 7 shows the structure of the

GQM+Strategies model (called “grid”) [Aoki16]. GQM+Strategies has been used to establish management strategies and plans, determine the value of a contribution, ensure the integrity of a goal between a purchaser and a contractor, and evaluate management based on quantitative data.

There are many successful cases applying GQM+Strategies with extensions for measurement-based IT business alignment, including:

- GSE introduced GQM+Strategies to Recruit Sumai Company Co., which provides services and products related to housing [Aoki16]. In this case, GQM+Strategies maintains consistency within a vertical refinement tree composed by goals, strategies, and measurements. In addition, since horizontal relations such as conflicting ones at different branches (Fig. 8) may be missed in the original GQM+Strategies approach, we proposed the Horizontal Relation Identification Method (HoRIM) to identify horizontal relations by employing Interpretive Structural Modeling (ISM) [Aoki16][Aoki17]. Applying GQM+Strategies along with HoRIM identifies about 1.5 times more horizontal relations than an ad hoc review.
- GSE together with Yahoo Japan proposed a method, GO-MUC method (Goal-oriented Measurement for Usability and Conflict) (Fig. 9), which is a goal-oriented strategy design approach considering the requirements of both the user and the business by combining GQM+Strategies and Persona approaches [Uchida16]. Applying GO-MUC to an actual software service development and operation demonstrated that GO-MUC can identify the interest between the business side and users side, realizing more effective and user-friendly strategies to resolve conflicting interests.

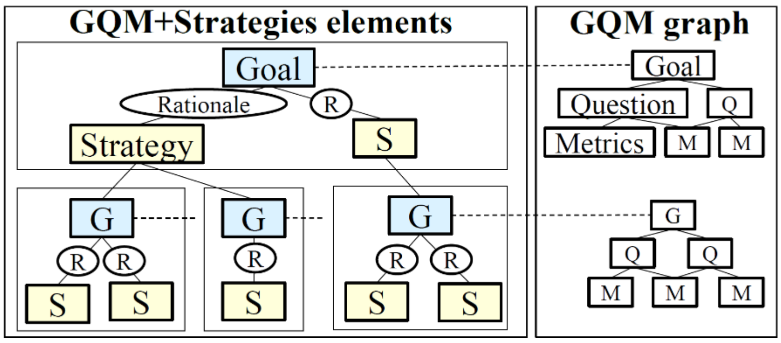


Figure 7: Structure of a GQM+Strategies grid

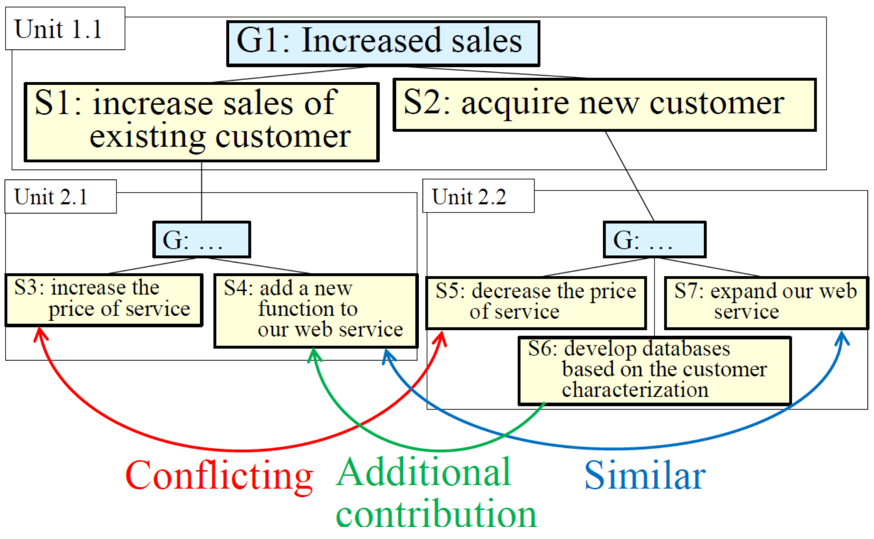
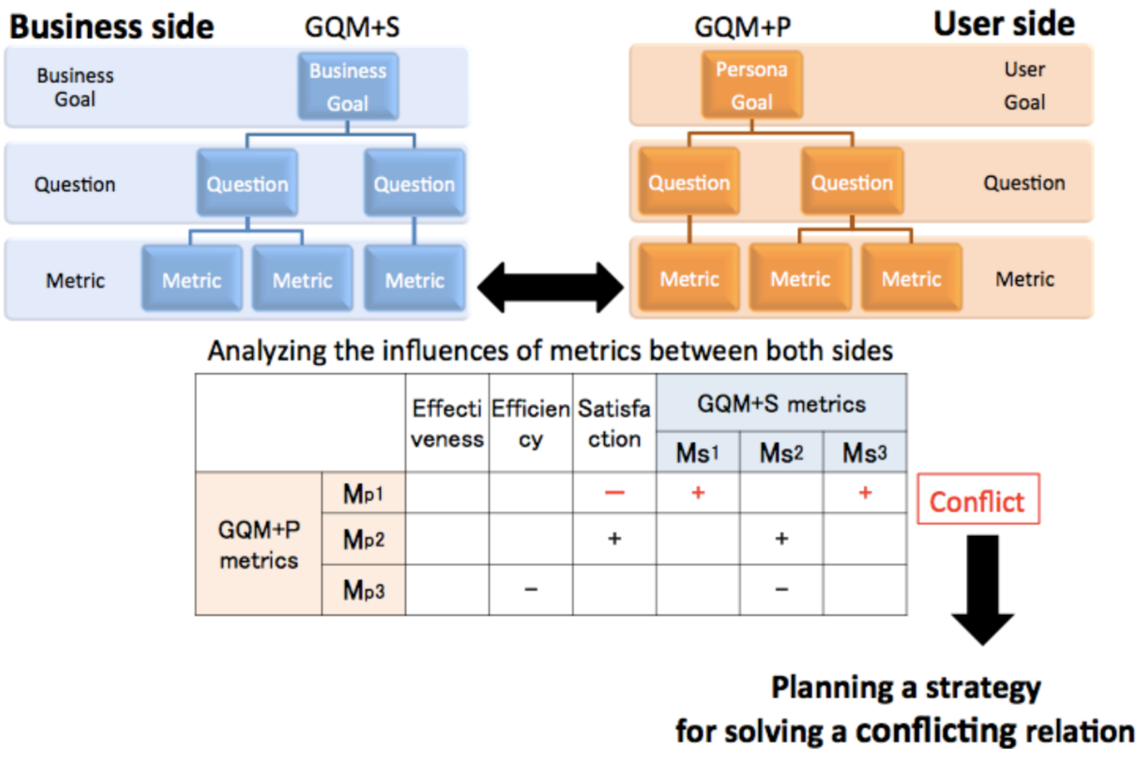

Unit 1.1
G1: Increased sales
S1: increase sales of existing customer
S2: acquire new customer
Unit 2.1
G: …
S3: increase the price of service
S4: add a new function to our web service
Unit 2.2
G: …
S5: decrease the price of service
S7: expand our web service
S6: develop databases based on the customer characterization
Conflicting
Additional contribution
Similar

Figure 8: Horizontal relations in GQM+Strategies grid



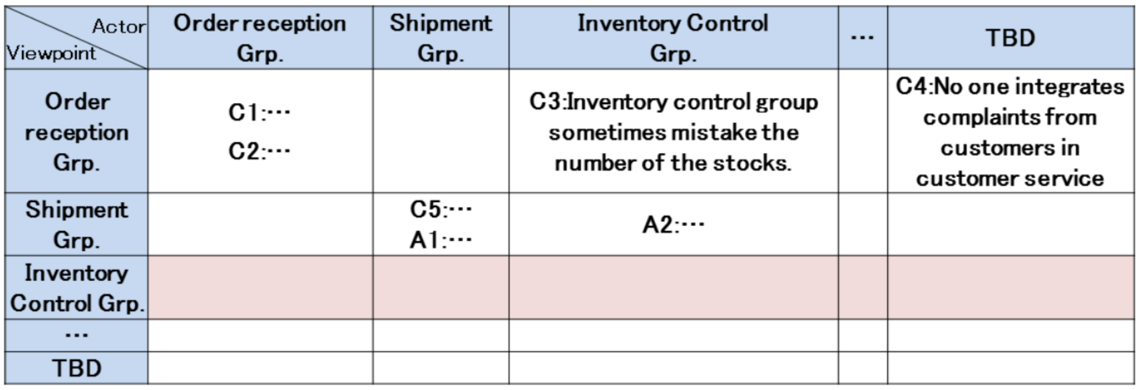
		Effectiveness	Efficiency	Satisfaction	GQM+S metrics		
					Ms1	Ms2	Ms3
GQM+P metrics	Mp1			−	+		+
	Mp2			+		+	
	Mp3		−			−	

Figure 9: Overview of GO-MUC [Uchida16]

3.2. Countermeasure: Exhaustive identification of fact-based rationales

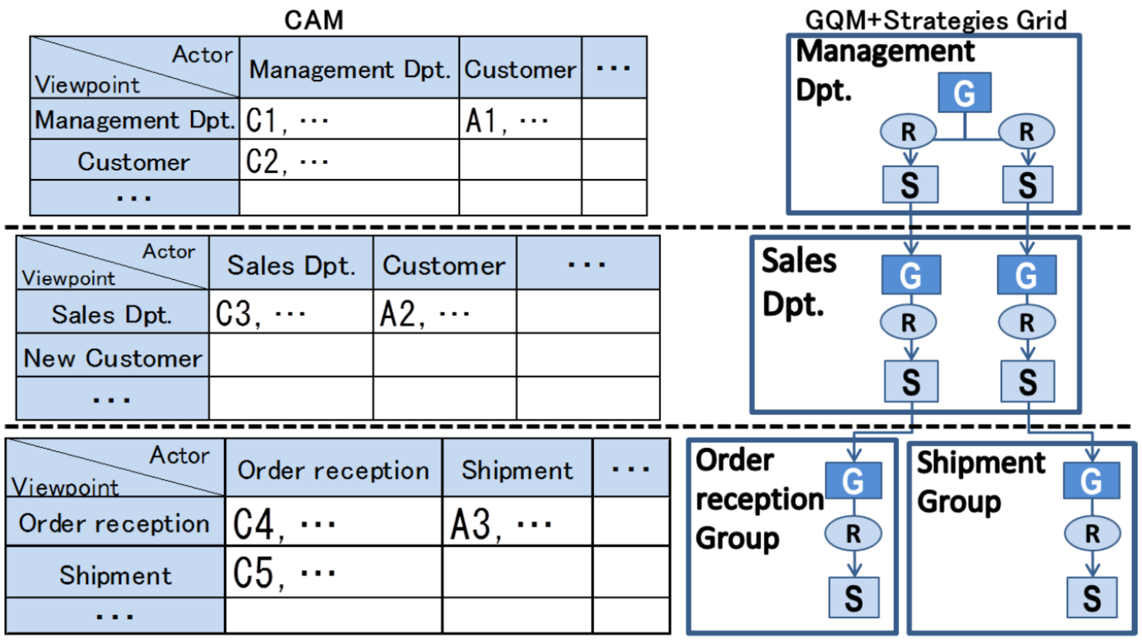
GQM+Strategies extracts strategies from goals based on rationales such as fact-based contexts and assumptions. A lack of rationales tends to be misleading and may result in deriving incorrect strategies. Consequently, rationales must be identified exhaustively.

GSE proposed a method named CAM (Context-Assumption-Matrix) to extract contexts and assumptions efficiently and exhaustively by analyzing the relationships between stakeholders. Figure 10 shows an example of CAM [Kobori]. CAM organizes common contexts and assumptions between stakeholders into a two-dimensional table. CAM can be employed as part of the GQM+Strategies grid construction process to refine business goals and strategies iteratively from top to bottom (Fig. 11).



Actor / Viewpoint	Order reception Grp.	Shipment Grp.	Inventory Control Grp.	···	TBD
Order reception Grp.	C1:··· C2:···		C3:Inventory control group sometimes mistake the number of the stocks.		C4:No one integrates complaints from customers in customer service
Shipment Grp.		C5:··· A1:···	A2:···		
Inventory Control Grp.					
···					
TBD					

Figure 10: Context-Assumption-Matrix



Actor / Viewpoint	Management Dpt.	Customer	···
Management Dpt.	C1, ···	A1, ···	
Customer	C2, ···		
···			

Actor / Viewpoint	Sales Dpt.	Customer	···
Sales Dpt.	C3, ···	A2, ···	
New Customer			
···			

Actor / Viewpoint	Order reception	Shipment	···
Order reception	C4, ···	A3, ···	
Shipment	C5, ···		
···			

Figure 11: Iterative process of GQM+Strategies grid refinement with CAM

4. Pitfall: Uncertain future

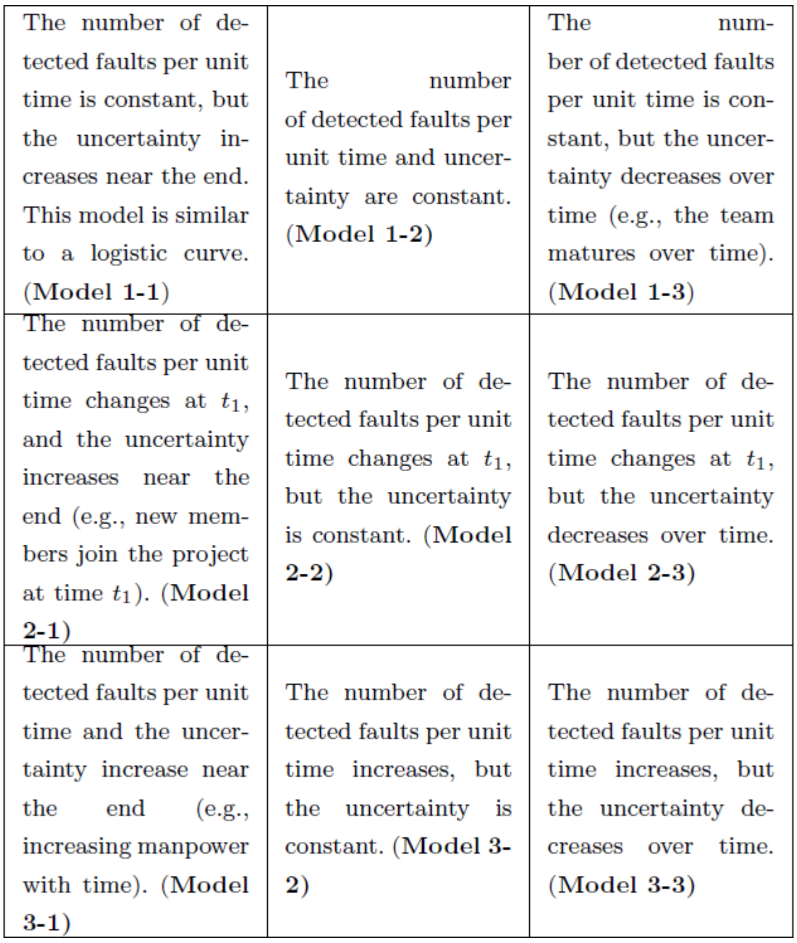
Quality measurements and evaluations are often conducted based on the strong assumption that the "future is an extension of the present." Especially in an era of uncertainty in computing and environments, there is no guarantee that a prediction or estimation of quality or related elements based on past data at a certain time point will be correct in the future. At least two countermeasures will prevent such incorrect predictions and estimations: quality prediction in consideration with uncertainty and continuous improvement of quality measurement and continuous measurement program improvement by machine learning.

4.1. Countermeasure: Prediction incorporating uncertainty

Among the various quality characteristics, software reliability is a critical component of computer system availability. Software reliability growth models (SRGMs), such as the Times Between Failures Model and Failure Count Model, can indicate whether a sufficient number of faults have been removed to release the software [Honda17]. Although logistic and Gompertz curves are both well-known software reliability growth curves, neither can account for the dynamics of software development because developments are affected by various elements in the development environment (e.g., skills of the development team and changing requirements).

To adapt to changes, GSE proposed a generalized software reliability model (GSRM) based on a stochastic process to simulate developments, which include uncertainties and dynamics such as unpredictable changes in the requirements and the number of team members [Honda17]. Figure 12 shows combinations of three different types of dynamics and three different types of uncertainty, resulting in nine models. Figure 13 plots the ratio of the cumulative number of detected faults at time *t* versus the total number of detected faults for the entire project using these nine models, where the x-axis represents time in arbitrary units.

GSE assessed two actual datasets using our formulated equations, which are related to three types of development uncertainties by employing simple approximations in GSRM. The results confirm that developments can be evaluated quantitatively. Additionally, a comparison of GSRM with existing software reliability models demonstrates that the approximation by GSRM is more precise than those by existing models [Honda17].



The number of detected faults per unit time is constant, but the uncertainty increases near the end. This model is similar to a logistic curve. (**Model 1-1**)	The number of detected faults per unit time and uncertainty are constant. (**Model 1-2**)	The number of detected faults per unit time is constant, but the uncertainty decreases over time (e.g., the team matures over time). (**Model 1-3**)
The number of detected faults per unit time changes at t_1, and the uncertainty increases near the end (e.g., new members join the project at time t_1). (**Model 2-1**)	The number of detected faults per unit time changes at t_1, but the uncertainty is constant. (**Model 2-2**)	The number of detected faults per unit time changes at t_1, but the uncertainty decreases over time. (**Model 2-3**)
The number of detected faults per unit time and the uncertainty increase near the end (e.g., increasing manpower with time). (**Model 3-1**)	The number of detected faults per unit time increases, but the uncertainty is constant. (**Model 3-2**)	The number of detected faults per unit time increases, but the uncertainty decreases over time. (**Model 3-3**)

Figure 12: Combinations of dynamics and uncertainty in the generalized software reliability model [Honda17]

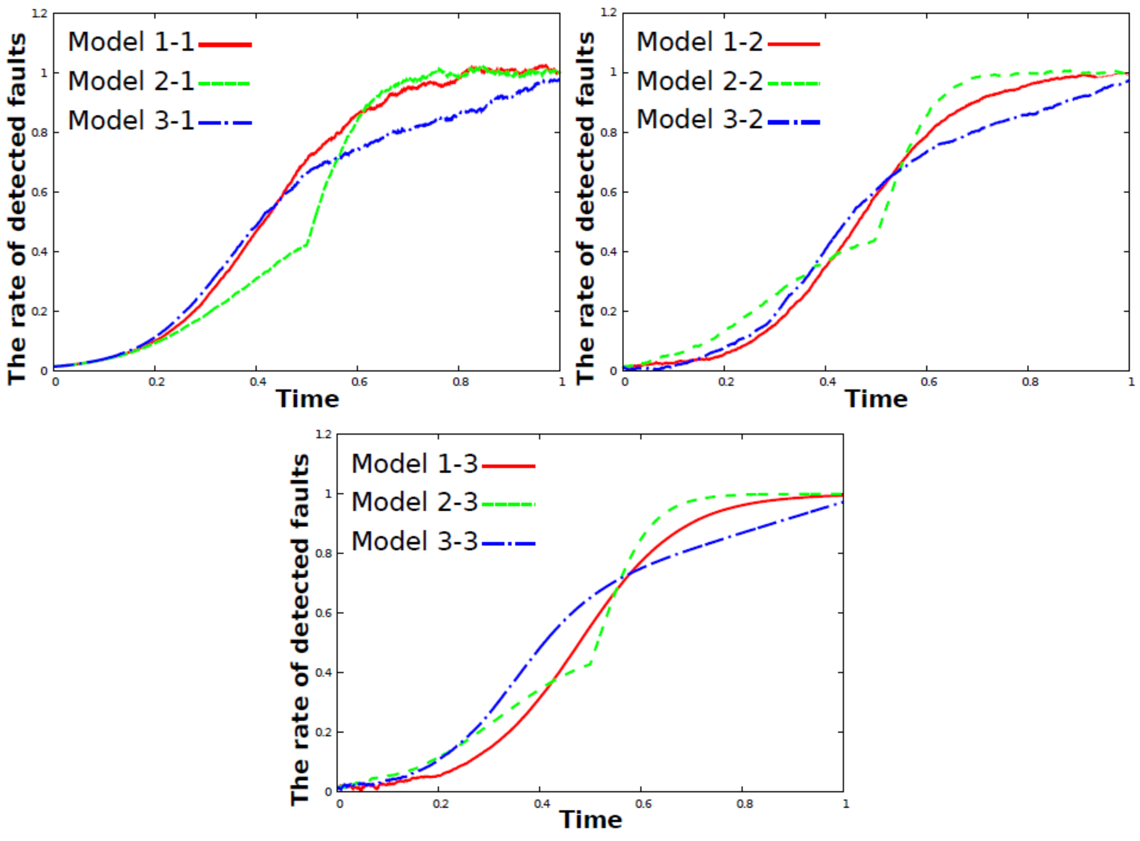


Figure 13: Example plots of the generalized software reliability model

4.2. Countermeasure: Continuous measurement program improvement by machine learning

Employing automated measurement tools and thresholds allows quality evaluations to be conducted automatically. For example, program static analysis tools can be used to measure attributes of programs. The measurement results can indicate high-risk program modules against given thresholds. However, the problem is how to establish appropriate measurements and thresholds from the viewpoint of quality evaluation because these may vary over time.

To solve this problem, GSE proposed a GQM-based process to search for optimal thresholds by supervised machine learning using measurement values taken from modules sampled as training data and improving measurement methods by experts' analysis of the evaluation results based on the thresholds [Tsuda16]. Figure 14 shows the entire process. Implementing the process iteratively can continuously improve and adapt a measurement program to the given context. A case study at a construction machinery company with the goal of detecting high-risk C++ files in embedded systems confirms that the proposed process is useful to achieve a goal in an iterative manner [Tsuda16].

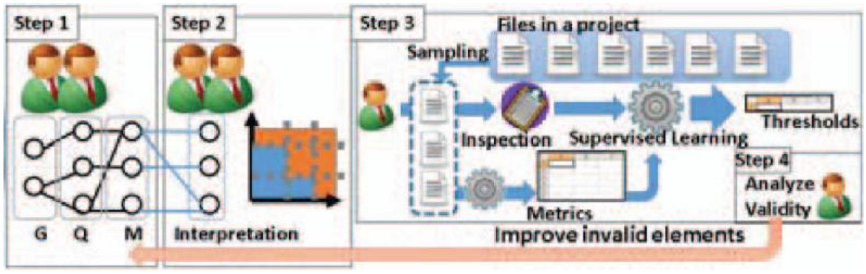


Figure 14: Iterative process to improve a measurement program by machine learning

5. Pitfall: Self-certified quality

Although quality requirements and evaluation methods can be defined for each development and operational context, these definitions should be "reasonable" from the viewpoint of targeted domains and industries. Otherwise, self-declared or self-certified software quality management without consideration of outside standards or industrial de-facts may cause the quality to decline. Such management may result in having relatively poor quality requirements or incomprehensive evaluations. At least two possible countermeasures exist: standard-based quality evaluation and pattern-based quality evaluation.

5.1. Countermeasure: Standard-based quality evaluation

Several works have strived to identify software quality, but the quality of software products is often not comprehensively, specifically, or effectively defined because previous approaches focused on specific quality aspects. Moreover, the evaluation results of quality metrics often depend on software stakeholders, making it difficult to compare quality evaluation results across software products. ISO/IEC has tried to define evaluation methods for the quality of software products and provide common standards, called the SQuaRE (Systems and software Quality Requirements and Evaluation) series [ISO25000][ISO25010], including ISO/IEC 25022:2016 [ISO25022] and ISO/IEC 25023:2016 [ISO25023]. Because the SQuaRE series includes ambiguous metrics, applying the series to products and comparing results can be challenging. Thus, GSE proposed a SQuaRE-based software quality evaluation framework, which successfully concretized many product metrics and quality in use metrics originally defined in the SQuaRE series [Nakai16].

Figure 15 overviews the framework. The framework is composed of two parts: "Product Quality" and "Quality in Use". The former contains internal and external product quality characteristics and metrics based on ISO/IEC 25023:2016, whereas the latter has quality characteristics and quality-in-use metrics based on ISO/IEC 25022:2016. Since the product quality is expected to influence the quality in use, the framework measures both qualities to clarify the relationship. In relation to PSQ Certification System [PSQ], GSE selected and concretized 47 product metrics and 18 quality-in-use metrics. These metrics can be concretely applied to almost any software package/service products. These metrics cover more than 50% of the metrics originally defined in the SQuaRE series.

This framework requires manual specifications, test specifications, and bug information to measure product quality (Fig. 15). Moreover, the framework requires information collected via a questionnaire and a user test to measure quality-in-use metrics. Finally, the overall software quality is assessed based on the results to clarify what quality characteristics are sufficient (or insufficient).

Through a case study targeting a commercial software product, GSE confirmed that the framework is concretely applicable to the software package/service product. The framework together with the measurement results of 21 Japanese software products is available as the Waseda Software Quality Benchmark (WSQB) at [WSQB17].

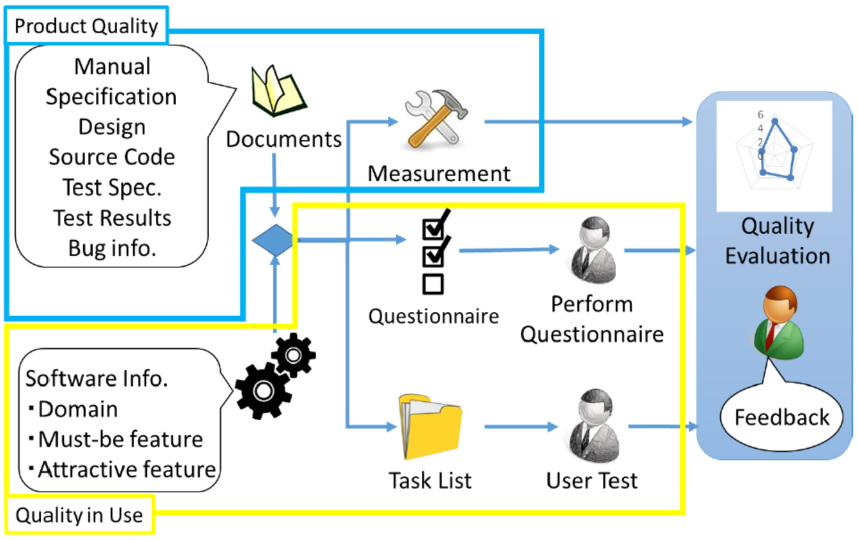


Figure 15: Overview of SQuaRE-based comprehensive evaluation framework

5.2. Countermeasure: Pattern-based quality evaluation

Some specific quality characteristics, especially security, are difficult to accommodate because not all software engineers are specialists in these characteristics. Patterns are reusable packages that encapsulate expert knowledge. Specifically, a pattern represents a frequently recurring structure, behavior, activity, process, or "thing" during the software development process. Many security patterns have been proposed [Yoshioka08] [Fernandez08].

For example, the Role-based Access Control (RBAC) pattern (Fig. 16), which is a representative pattern for access control, describes how to assign precise access rights to roles in an environment where access to computing resources must be controlled to preserve confidentiality and the availability requirements.

Security design patterns are difficult to implement because they are currently abstract descriptions. Additionally, validating security design patterns in the implementation phase is challenging because an adequate test case is required. Hence, a security design pattern can be inappropriately applied, which leads to serious vulnerability issues [Yoshizawa16].

To evaluate program implementations in terms of security, GSE proposed a method to support the implementation of security design patterns using a test template (Fig. 17). The method creates a test template from a security design pattern, which consists of an “aspect test template” to observe internal processing and a “test case template”. By providing design information in the test template, a test is created to evaluate the system in the early implementation stage using a tool and fixing the code. The test can be executed repeatedly. Thus, it can validate whether a security design pattern is appropriately applied in the implementation phase. The method has been used in a previous design-level validation tool called TESEM [Kobashi14][Kobashi15].

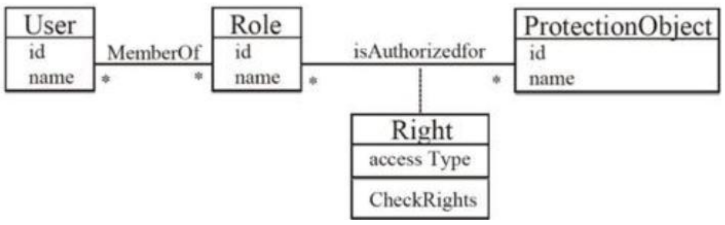


Figure 16: Essential structure of the RABC pattern

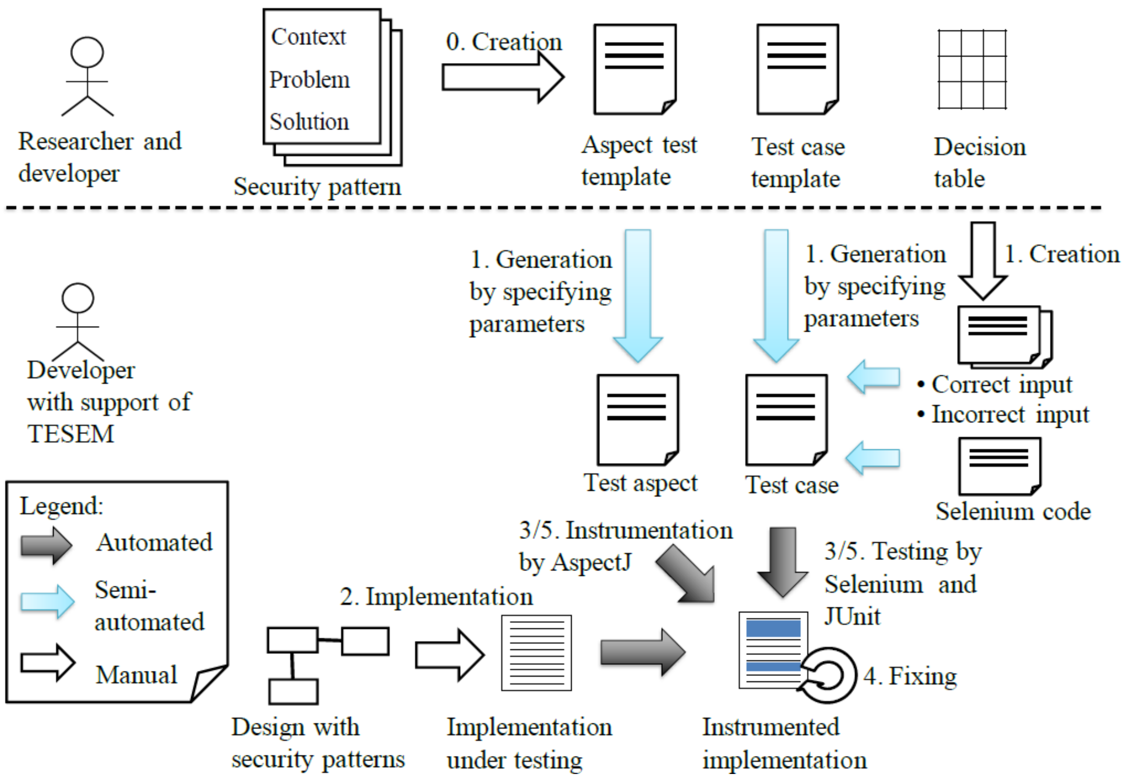


Figure 17: Overview of the security pattern implementation validation

6. Conclusion

Four pitfalls and eight corresponding countermeasures in software quality measurements and evaluations are explained using actual case studies and adaptation results mostly taken at GSE with industrial collaborations. These pitfalls and countermeasure could be utilized for efficient and effective software quality measurements and evaluations.

Acknowledgments

Achievements of GSE introduced in this chapter were supported by JSPS KAKENHI Grant Numbers 25330091 and 16H02804, “Research Initiative on Advanced Software Engineering in 2015” supported by Software Reliability Enhancement Center (SEC), and IISF SSR Forum 2015 and 2016.

References

[GSE] Global Software Engineering Laboratory, Waseda University, https://www.waseda.jp/inst/gcs/en/labo/globalsoftware/

[Linda06] Linda M. Laird and M. Carol Brennan, “Software Measurement and Estimation: A Practical Approach,” Wiley-IEEE Computer Society, 2006.

[Basili02] Victor R. Basili, Gianluigi Caldiera, Dieter Rombach, and Rini van Solingen, “Goal

Question Metric (GQM) approach," Encyclopedia of Software Engineering, John Wiley & Sons, Hoboken, New Jersey, 2002.

[Washizaki07] Hironori Washizaki, Rieko Namiki, Tomoyuki Fukuoka, Yoko Harada and Hiroyuki Watanabe, "A Framework for Measuring and Evaluating ProgramSource Code Quality," Proceedings of the 8th International Conference on Product Focused Software Development and Process Improvement (PROFES 2007), Springer LNCS, pp.284-299, 2007.

[Washizaki12] Hironori Washizaki, Toshikazu Koike, Rieko Namiki, Hiroyuki Tanabe, "Reusability Metrics for Program Source Code Written in C Language and Their Evaluation," Proceedings of the 13th International Conference on Product-Focused Software Development and Process Improvement (Profes 2012), pp.89-103, June 13-15, 2012.

[Sato13] Seiji Sato, Hironori Washizaki, Yoshiaki Fukazawa, Sakae Inoue, Hiroyuki Ono, Yoshiiku Hanai and Mikihiko Yamamoto, "Effects of Organizational Changes on Product Metrics and Defects," Proceedings of the 20th Asia-Pacific Software Engineering Conference (APSEC 2013). pp.132-139, Bangkok, Thailan, 2-5 December 2013.

[Ando15] Reou Ando, Seiji Sato, Chihiro Uchida, Hironori Washizaki, Yoshiaki Fukazawa, Sakae Inoue, Hiroyuki Ono, Yoshiiku Hanai, Masanobu Kanazawa, Kazutaka Sone, Katsushi Namba, Mikihiko Yamamoto, "How Does Defect Removal Activity of Developer Vary with Development Experience?,"Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE 2015), Wyndham Pittsburgh University Center, Pittsburgh, USA, July 6-8 2015.

[Tsunoda17] Taketo Tsunoda, Hironori Washizaki, Fukazawa Yosiaki, Inoue Sakae, Yosiiku Hanai and Masanobu Kanazawa, "Evaluating the Work of Experienced and Inexperienced Developers Considering Work Difficulty in Software Development," 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, June 26-28, 2017, Kanazawa, Japan.

[Ishizue16] Ryosuke Ishizue, Hironori Washizaki, Yoshiaki Fukazawa, Sakae Inoue, Yoshiiku Hanai, Masanobu Kanazawa and Katsushi Namba, "Metrics visualization technique based on the origins and function layers for OSS-based development," 4th IEEE Working Conference on Software Visualization (VISSOFT 2016), NIER Track, Raleigh, North Carolina, USA, October 3-4, 2016.

[Nakai14] Hidenori Nakai, Kiyoshi Honda, Hironori Washizaki, Yoshiaki Fukazawa, Ken Asoh, Kaz Takahashi, Kentarou Ogawa, Maki Mori, Takashi Hino, Yosuke Hayakawa, Yasuyuki Tanaka, Shinichi Yamada, Daisuke Miyazaki, "Initial Industrial Experience of GQM-based Product-Focused Project Monitoring with Trend Patterns," 21st Asia-Pacific Software Engineering Conference (APSEC 2014), Jeju, Korea, December 1-4, 2014.

[Jacobson12] Ivar Jacobson, Pan-Wei Ng, Paul E. McMahon, Ian Spence, Svante Lidman, "The Essence of Software Engineering: The SEMAT Kernel," Communications of the ACM, vol.55, no.12, pp.42-49, December 2012.

[ITA] Information Technology Alliance, http://ita.gr.jp/

[Washizaki17] Hironori Washizaki, “Analyzing and refining project failure cases from wider viewpoints by using SEMAT Essence,” Essence Conference in Seoul, 2017.

[Basili10] Victor Basili, Mikael Lindvall, Myrna Regardie, Carolyn Seaman, Jens Heidrich, Jurgen Munch, Dieter Rombach, Adam Trendowicz, "Linking Software Development and Business Strategy Through Measurement", Computer, Vol. 43, No. 4, pp.57–65., 2010.

[Basili14] Victor Basili, Adam Trendowicz, Martin Kowalczyk, Jens Heidrich, Carolyn Seaman, Jürgen Münch, Dieter Rombach, “Aligning Organizations through Measurement - The GQM+Strategies Approach,” Springer-Verlag, 2014.

[Aoki16] Yohei Aoki, Takanobu Kobori, Hironori Washizaki, Yoshiaki Fukazawa, “Identifying Misalignment of Goal and Strategies across Organizational Units by Interpretive Structural Modeling,” Proceedings of the 49th Hawaii International Conference on System Sciences (HICSS-49), January 5-8, 2016 Grand Hyatt, Kauai

[Aoki17] Yohei Aoki, Hironori Washizaki, Chimaki Shimura, Yuichiro Senzaki, Yoshiaoki Fukazawa, “Experimental Evaluation of HoRIM to Improve Business Strategy Models,” 16th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2017), May 24-26, 2017, Wuhan, China.

[Uchida16] Chihiro Uchida, Kiyoshi Honda, Hironori Washizaki, Yoshiaki Fukazawa, Kentaro Ogawa, Tomoaki Yagi, Mikako Ishigaki, Masashi Nakagawa, “GO-MUC: A Strategy Design Method Considering Requirements of User and Business by Goal-Oriented Measurement,” 9th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2016), collocated with ICSE 2016, Autin, Texas, USA, May 16, 2016.

[Honda17] Kiyoshi Honda, Hironori Washizaki and Yoshiaki Fukazawa, “Generalized Software Reliability Model Considering Uncertainty and Dynamics: Model and Applications,” International Journal of Software Engineering and Knowledge Engineering, pp.1-29, 2017.

[Tsuda16] Naohiko Tsuda, Masaki Takada, Hironori Washizaki, Yoshiaki Fukazawa, Shunsuke Sugimura, Yuichiro Yasuda, Masanao Futakami, “Iterative Process to Improve GQM Models with Metrics Thresholds to Detect High-risk Files,” IEEE TENCON 2016, Marina Bay Sands, Singapore, 22-25 November 2016.

[ISO25000] ISO/IEC. ISO/IEC 25000:2014 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE. 2014.

[ISO25010] ISO/IEC. ISO/IEC 25010:2011 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. 2011.

[ISO25022] ISO/IEC. ISO/IEC 25022:2016 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of quality in use. 2015.

[ISO25023] ISO/IEC. ISO/IEC 25023:2016 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of system and software product quality. 2015.

[Nakai16] Hidenori Nakai, Naohiko Tsuda, Kiyoshi Honda, Hironori Washizaki, Yoshiaki Fukazawa, "A SQuaRE-based Software Quality Evaluation Framework and its Case Study," IEEE TENCON 2016, Marina Bay Sands, Singapore, 22-25 November 2016.

[PSQ] CSAJ, "PSQ Certification System," http://www.psq-japan.com/english/

[WSQB17] Global Software Engineering Laboratory, "Waseda Software Quality Benchmark," http://www.washi.cs.waseda.ac.jp/?page_id=3910

[Yoshioka08] Nobukazu Yoshioka, Hironori Washizaki, Katsuhisa Maruyama, "A Survey on Security Patterns," Progress in Informatics, Vol.5, pp.35-47, 2008.

[Fernandez08] Eduardo B. Fernandez, Hironori Washizaki, Nobukazu Yoshioka, "Abstract security patterns," Proceedings of the 15th Conference on Pattern Languages of Programs, 2008.

[Yoshizawa16] Masatoshi Yoshizawa, Hironori Washizaki, Yoshiaki Fukazawa, Takao Okubo, Haruhiko Kaiya and Nobukazu Yoshioka, "Implementation Support of Security Design Patterns Using Test Templates," Information, Vol.7, No.2(34), pp.1-19, 2016.

[Kobashi14] Takanori Kobashi, Nobukazu Yoshioka, Haruhiko Kaiya, Hironori Washizaki, Takao Okubo, Yoshiaki Fukazawa, "Validating Security Design Pattern Applications by Testing Design Models," International Journal of Secure Software Engineering, Vol. 5, Issue 4, pp.1-30, 2014.

[Kobashi15] Takanori Kobashi, Masatoshi Yoshizawa, Hironori Washizaki, Yoshiaki Fukazawa, Nobukazu Yoshioka, Haruhiko Kaiya, Takano Okubo, "TESEM: A Tool for Verifying Security Design Pattern Applications by Model Testing," Proceedings of the 8th IEEE International Conference on Software Testing, Verification, and Validation (ICST 2015), Tool Track, 13 – 17 April 2015, Graz, Austria.