Tool Support to Model-based Quality Analysis of Software Architecture

Qian Zhang, Jian Wu
Department of Computer Science
National University of Defense Technology
Changsha, China
Email: zhangqian@nudt.edu.cn

Hong Zhu
Department of Computing
Oxford Brookes University
Oxford OX33 1HX, UK
Email: hzhu@brookes.ac.uk

ABSTRACT

This paper presents an automated software tool SQUARE (Software QUality and ARchitecture modelling Environment). It is designed and implemented to support the analysis of software quality from software architectural designs. The tool is based on a model-based method and follows a structured process to systematically derive quality models from software architectural designs by adapting and applying the principles of system hazard analysis. Through identification of potential quality hazards and their consequences, the quality related properties of the components and connectors and the causal relationships between them are derived and then translated into a quality model represented in a graphical notation. The tool enables automated analysis of the quality models in the graphical notation to recognize a number of types of software quality features including quality sensitive components, quality risks and quality trade-off points, etc. A case study with a real e-commerce system is also reported.

Keywords: Automated software tools, Software architecture design, Software quality models, Analysis of software architecture

1. Introduction

Software quality is an elusive concept [1]. A great amount of effort has been made over the past a few decades to define software quality models in order to understand the concept, to measure software systems' quality and to improve software quality. Existing software quality models fall into two types: hierarchical models and relational models. Hierarchical models, such as McCall model [2], Boehm model [3], ISO model [4], and the more recent Bansiya and Davis' model of OO software design [5], define a set of quality related properties and organise them into a hierarchical structure to express the positive relationships between them. However, they are incapable of expressing negative relations between quality related properties. A relational model usually defines a number of stereo types of relationships between quality attributes, such as positive, negative and neutral relations. Typical examples of such quality models include Perry Model [6] and Gillies Model [7, 8]. There are also a number of quality models of information systems [9].

These quality models can help software developers to improve software quality by providing guidelines to software development activities, such as in the elicitation of quality requirements. However, as pointed out by Dromey [10, 11], they fail to take software structures into account. Moreover, they are incapable to deal with complicated relationships between quality attributes. They provide little help to the design of software systems.

In the past a decade or so, a significant progress has been made in the analysis of software architectures. A number of methods have been advanced in the literature to analyse the quality of software architectural designs [12,13,14,15,16,17]. Among the most well-known are SAAM [15, 16] and ATAM [17], etc.; see [18] for a survey. Almost all of these methods are scenario-based. They examine software architectures in the context of a set of scenarios, although the ways that scenarios are elicited and used vary. They have a number of advantages, including the examination of software behaviour in realistic situations, reduction of complexity of analysis through focusing on typical scenarios, etc. However, there are difficulties to build an overall picture of the system's quality especially when there are intensive and complicated interactions between scenarios. The elicitation of a complete and representative set of scenarios is by no means a trial task, which is currently still a brainstorming process. The result of quality analysis may heavily depend on the selection of scenarios as reported in practices [19].

In our previous work [20,21], we proposed an alternative approach to the quality analysis of software architectures. It is a model-based method aiming at systematically analyzing the architectural designs through building a quality model for the system under scrutiny. A graphical notation of software quality models are devised so that detailed and complex relationships between quality attributes in the context of the architecture can be represented. In this paper, we further investigate how quality issues can be automatically identified from such a quality model. We present a software tool that supports quality model construction and analysis. A case study of the method and the tool is also reported.

The remainder of the paper is organised as follows. Section 2 outlines the quality modelling and analysis method HASARD, which stands for Hazard Analysis of Software ARchitectural Designs. It includes a number of improvements of the method proposed in our previous work. Section 3 describes the supporting tool SQUARE, which stands for Software QUality and ARchitecture modelling Environment. Section 4 reports a case study. Section 5 concludes the paper with a discussion of the directions for further work.

2. The HASARD Method

The HASARD method consists of three main elements: a graphical notation for representation of software quality models, a structured process of deriving software quality models from architectural designs, and a repository of algorithms that enables the automated identification of quality features concealed in a quality model.

2.1. Representation of software quality models

As shown in Figure 1, a quality model in the graphical notation is a directed graph, which consists of a set of *nodes* and a set of *links* between the nodes.

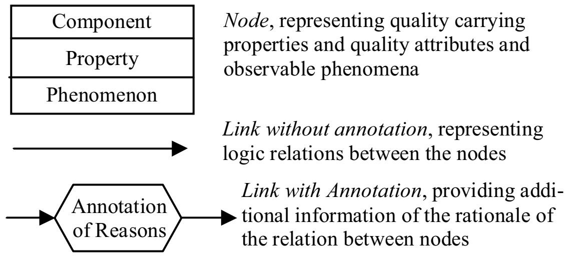

Component
Property
Phenomenon
Node, representing quality carrying properties and quality attributes and observable phenomena
Link without annotation, representing logic relations between the nodes
Annotation of Reasons
Link with Annotation, providing additional information of the rationale of the relation between nodes

Figure 1. Graphical notation of quality models

Each node represents a quality related property of the software system. It contains three parts specified in three compartments. The *property compartment* gives a property of the element specified in the component compartment. Such a property can be a *quality attribute*, such as correctness, or a *quality carrying property*, which is a property that may not be a quality attribute but affect the quality of the system somehow, such as the size of the component. The *phenomenon compartment* further describes a particular observable phenomenon of the property of the related element. The element in the *component compartment* can be a component or a connector of the software system, or a subsystem even the system itself, or an external entity, etc.

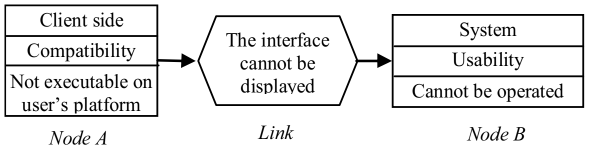


Figure 1. Example of nodes and link

For example, in Figure 2, the node *A* states that the client-side subsystem of a web-based application in the client-server architecture has a compatibility issue which is demonstrated by the phenomenon that the code is not executable on the user's platform.

The *links* are directed arcs between the nodes. A link from node *A* to node *B* means that the observation of the phenomenon on node *A* implies the occurrence of the phenomenon on node *B*. Each link may contain an optional annotation for the reasons why the two nodes are related. For example, in Figure 2, the link between nodes *A* and *B* states that if the client-side code cannot be executed on the user's platform, the system cannot be operated, because the interface cannot be displayed on the user's screen at all. In most cases, the reasons are self-evident and obvious. However, the annotations on the links between two nodes provide a means of validation of quality models.

2.2. Derivation of quality models

The HASARD method is inspired in the hazard analysis techniques that have been widely used in the development and deployment of safety critical systems [22, 23, 24, 25]. They were developed to systematically identify, assess and control hazards before a new work process, a piece of equipment, or other activity is initiated. Some of the hazard analysis methods have been adapted for software safety. In order to analyse a wider range of software quality attributes not just safety, the concept of hazard is extended and analysis methods are adopted. In our context, the word *hazard* means a situation that may cause undesirable effect on software quality.

In the HASARD method, the construction of quality models takes software architectural models as the input. It consists of the following four steps.

(1) *Hazard identification*. A hazard identification method is applied to identify all quality sensitive observable phenomena of the components, connectors, the system, etc.
(2) *Cause-consequence analysis*. The causal relationships between the identified hazards are recognized.
(3) *Model assembling*. The information obtained in the previous steps are assembled together and represented in the graphical notation.
(4) *Quality concern recognition*. The quality carrying properties/quality attributes that a phenomenon demonstrates are recognized according to the nature of the phenomenon.

The following describes the process step by step.

2.2.1. Hazard identification

The process of hazard analysis starts with the identification of the hazards. One of the most effective methods of hazard identification is the so called Hazard and Operability Studies, or shortly HAZOP [26].

The method relies on determining answers to questions of *what-if* nature. A set of guide words has been developed to systematically develop a collection of what-if questions. They are applied to the attributes of various components and connectors of the system being studied. If a deviation from the normal working of the component is credible, the behaviour of the component is considered as a possible hazard.

Table 1 lists the guide words that we adapted for analyzing software architectural designs. Each guide word can be applied to one or more attributes. Its meaning depends on the type of attribute and the context in the system. For example, the guide word 'NO' can be applied to the data produced by a component. It means no data is produced by the component. In hazard identification, the analyst will be asked the what-if question that 'what would happen if the data is not produced by the component?'. The same guide word can also be applied to an architectural component. In such a context, it means that the component is not contained in the system.

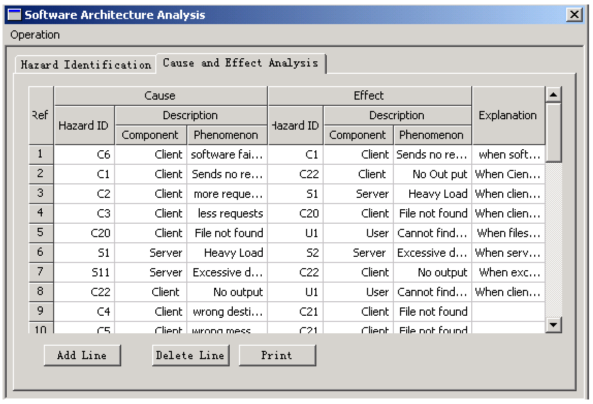
Table 1. Guide words for software hazard identification

Guide word	Applicable attribute	Interpretations
No	Date/control signals	No data / control signal exchanged; No data / control signals produced/received.
	Component property/ function	The component / connector does not have the designed property / function.
	Component / connector	The system does not contain the component / connector.
More	Quantitative parameters	The value of the parameter is too large.
Less	Quantitative parameters	The value of the parameter is too small.
As well as	Event or activity	The intended event / activity occurs, but in addition, redundant data are sent. Data is sent to the designated receiver as well as an unintended receiver.
	Component / connector	In addition to the intended components / connectors, other components / connectors are added.
Part of	Structured data	Only a part of the data produced, stored or received.
	Structure events	Only a part of the events happened.
Reverse	Direction of flow	The information flow in the opposite direction.
	Event	The opposite event happened.
Other than	Data / control signals, quantitative / qualitative parameters	Incorrect data / control signals produced; The parameter has a value different from the design.
	Component / system's function/ property	The component has a functionality / property different from the designed.
	Component / connector	Other kind of component / connector is contained.
Early	Periodical events	The event happened earlier than expected.
Late	Periodical events	The event happened later than expected.
Before	Temporal orders	The event happened in the order earlier than designed.
After	Temporal orders	The event happened in the order later than designed.

2.2.2. Cause-consequence analysis

Cause-consequence analysis aims at understanding the causal relationships between hazards. The identified hazards in the previous step are investigated of their causes and consequences. It can be performed backward or forward, or a combination of both.

Forward analysis searches for potential effects, i.e. consequences, of a hazard until the consequence is terminal. A hazard or failure mode is terminal if it does not affect any other component of the system or does not cause any other hazards/failures. In many cases, a hazard or failure mode is regarded as terminal simply because we are not interested in its further consequences. *Backward* analysis starts with a hazard to search for its causes until the hazard is primitive. A hazard or failure mode is primitive if its causes cannot be further identified without additional knowledge about the system. A hazard/failure mode can also be considered as primitive if we are not interested in its causes. The results of cause-consequence analysis can be recorded in a form for the use in the assembling of a graph quality model at the next steps. Figure 3 shows the structure of the form.



Ref	Cause			Effect			Explanation
	Hazard ID	Description		Hazard ID	Description		
		Component	Phenomenon		Component	Phenomenon	
1	C6	Client	software fai...	C1	Client	Sends no re...	when soft...
2	C1	Client	Sends no re...	C22	Client	No Out put	When Cien...
3	C2	Client	more reque...	S1	Server	Heavy Load	When clien...
4	C3	Client	less requests	C20	Client	File not found	When clien...
5	C20	Client	File not found	U1	User	Cannot find...	When files...
6	S1	Server	Heavy Load	S2	Server	Excessive d...	When serv...
7	S11	Server	Excessive d...	C22	Client	No output	When exc...
8	C22	Client	No output	U1	User	Cannot find...	When clien...
9	C4	Client	wrong desti...	C21	Client	File not found	
10	C5	Client	wrong mess	C21	Client	File not found	

Figure 2. Structure of cause-consequence analysis records

2.2.3. Constructing graphic model

The construction of a quality model takes the information charted in the cause-consequence analysis records and translates them into graphical representation. Each hazard or failure mode in the record becomes a node with the component and phenomenon as specified in the record. Each row in the record becomes a link from the node that represents the cause to the node that represents the consequence. The explanation column of the row forms the reason of the link.

2.2.4. Identification of quality concerns

For each node in the diagram generated so far, the observable phenomenon is compared with the definitions of a set of quality attributes and quality-carrying properties of the components. The quality attribute or quality carrying property that the phenomenon demonstrates is then identified, or a new attribute or property is recognised. This property is filled into the slot of each node. For example, 'a hyperlink is broken' demonstrates the quality attribute correctness of the HTML file. 'Server is down' is related to the availability of the server.

2.3. Analysis of quality features

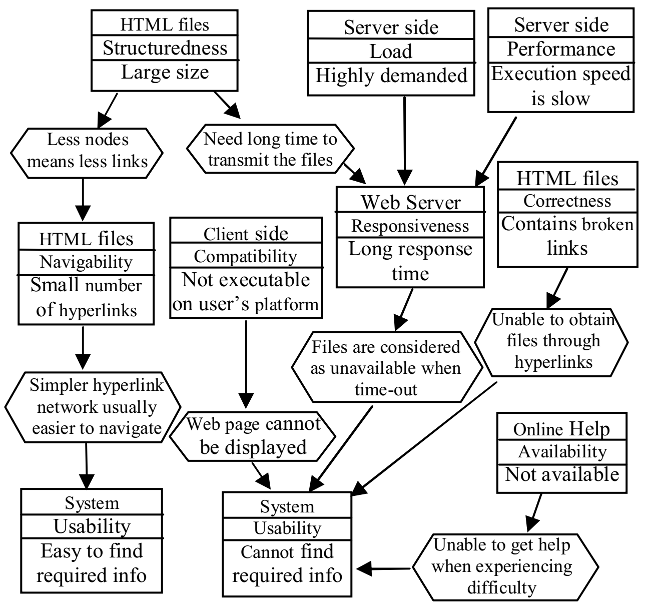
Given a graphic quality model, a number of different types of quality features of an architectural design can be derived. The following discusses a few such quality features that can be automatically recognized by using the algorithms that have already been implemented in our quality analysis tools. Details of these algorithms can be found in [27].

2.3.1. Contribution factors of a quality attributes

In the analysis of software architectural designs, we often want to know how a quality issue is addressed. We want to know which components, connectors or the configuration are related to the quality issue and how they collectively provide the solution to meet quality requirements. The contribution factors of a quality attribute is a set of properties of the components and/or connectors and the configuration of the architecture that affect the quality issue according to the design. For example, consider the quality model given in Figure 4. We can derive the sub-graph shown in Figure 5 for the contribution factors of server's responsiveness.

2.3.2. Impacts of design decisions

Another frequently asked question in the analysis of a software architectural design is "what are the consequences of a design decision on the properties and functionality of a component or connector?' In such cases, we need to find out what are the quality attributes that are affected by the design decision. Such information can also be derived from a well constructed quality model. For example, consider the quality model depicted in Figure 4. We can obtain the sub-graph shown in Figure 6 that represents the impacts of the quality carrying property of HTML files' size on other quality attributes. It shows that the size of HTML files affects the navigability and responsiveness of the system, which further affects the usability of the system.



HTML files
Structuredness
Large size
Server side
Load
Highly demanded
Server side
Performance
Execution speed is slow
Less nodes means less links
Need long time to transmit the files
Web Server
Responsiveness
Long response time
HTML files
Correctness
Contains broken links
HTML files
Navigability
Small number of hyperlinks
Client side
Compatibility
Not executable on user's platform
Files are considered as unavailable when time-out
Unable to obtain files through hyperlinks
Simpler hyperlink network usually easier to navigate
Web page cannot be displayed
Online Help
Availability
Not available
System
Usability
Easy to find required info
System
Usability
Cannot find required info
Unable to get help when experiencing difficulty

Figure 3. An example of quality model

2.3.3. Quality risks

A design decision may have positive as well as negative effects on a quality attribute. The negative effects may impose quality risks to the system. Therefore, it is often desirable to know where the quality risks are within an architectural design. This can also be derived from a quality model.

A negative effect of a design decision can be recognised by searching for the links in the quality model that have a negative effect. For example, in the quality model depicted in Figure 4, there is a link between the node of *HTML* with the property of *large file size* and the node of *web server* with a property of *responsiveness*. The link is negative since the larger the size of the file, the poorer the responsiveness of the web server. Therefore, a design decision of large file size is a risk to the quality attribute of responsiveness. The

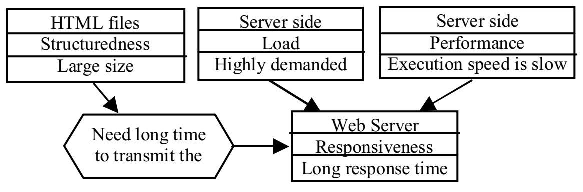
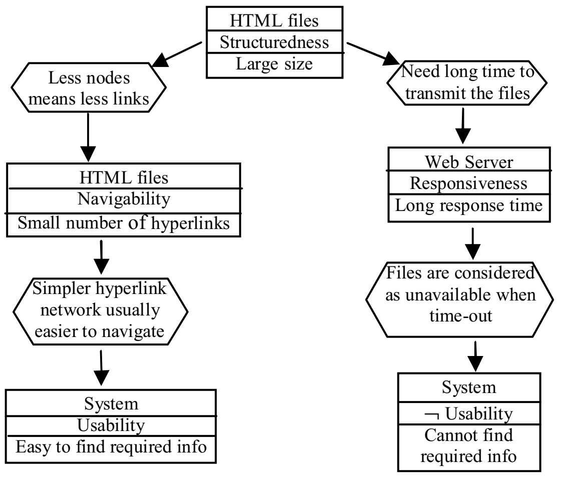


Figure 4. Factors of server's responsiveness



HTML files
Structuredness
Large size
Less nodes means less links
Need long time to transmit the files
HTML files
Navigability
Small number of hyperlinks
Web Server
Responsiveness
Long response time
Simpler hyperlink network usually easier to navigate
Files are considered as unavailable when time-out
System
Usability
Easy to find required info
System
¬ Usability
Cannot find required info

Figure 5. Relation of usability to the size of HTML file

further consequences of a quality risk can be identified and analyzed. In certain cases, a negative effect, i.e. a quality risk, is not the consequence of a single design decision. Instead, it can be the consequence of a number of other design decisions. In that case, all the causes must be identified so that a better design can be made. This can also be derived from the quality model.

2.3.4. Relationships between two quality issues

An important question to be answered in quality analysis is the interrelationship between two quality issues. For example, how server's performance is related to the system's usability? Answers to such questions can be found from the quality model by searching for all paths from a node that represents one quality issue to the node that represents the other quality issue.

2.3.5. Trade-off points

In many situations, a quality risk cannot be resolved without compromising on other quality issue(s) because these quality issues are conflicting with each other. In such cases, trade-offs between the quality attributes must be made and a balance between them must be achieved through appropriate design decisions.

For example, consider the quality model in Figure 4. The size of HTML files positively affects the navigability of the hypertext network, but negatively affects responsiveness of the web server. Therefore, navigability is in conflict with responsiveness. A trade-off between them must be made so that responsiveness is within a tolerable range while navigability is also acceptable. Such a trade-off occurs in the form of deciding on a suitable size of HTML file. In other words, HTML file size is a trade-off point. Trade-off points can also be derived from quality models automatically. Once a trade-off point is recognised, we can derive all quality attributes that the trade-off point affects, and to find all the factors that affect the trade-off point as discussed above.

3. The SQUARE Tool

To support the construction and analysis of software architectural designs using HASARD method, we developed a software tool called SQUARE, which stand for Software QUality and ARchitecture modeling Environment. It provides three main functions: modeling of software architecture in a graphical visual notation, analysing software architecture models using HAZARD method to derive software quality models in the graphical notation presented in section 2, and analysing quality models. As shown in Figure 7, the SQUARE tool consists of the following components.

(1) The *Architecture Model Editor* supports software architecture modeling through an interactive graphical user interface and represents software architectural models in the Software Architecture Visual Notation proposed by Bass, Clements, and Kazman in [16].

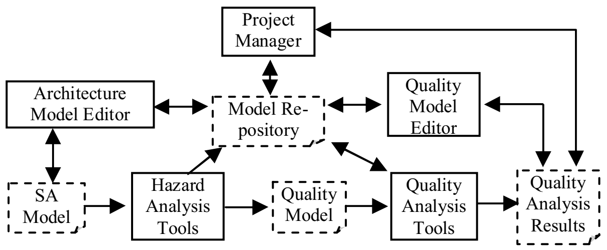


Figure 6. The structure of SQUARE tool

(2) The *Hazard Analysis Tools* help the developers to analyze software architectures using HASARD method. It records the analysis results and automatically transforms them into the graphic representation of quality models. It consists of three tools. The hazard identification tool helps the users to apply guide words to various attributes of components/connectors in software architecture models so that hazards are systematically identified. The cause-consequence analysis tool helps the user to identify the causal relationships between the hazards. The quality model generation tool automatically transforms the results of hazard analysis into a quality model in graphical notation. Figure 3 shows the interfaces of the hazard analysis tools.

(3) The *Quality Model Editor* provides an interactive graphical user interface to the users for the display and modification of software quality models.

(4) The *Quality Model Analysis Tools* automatically recognize and identify the quality features of the software designs from a quality models when invoked by the user as discussed in section 2. The results of the analysis are also displayed as a diagram in the graphical notation of software quality models. An example of such a generated sub-diagram is shown in Figure 9.

(5) The *Model Repository* stores the architecture and quality models, which can be reused across different development projects.

4. Case Study

A case study of the HASARD method and the SQUARE tool has been conducted with a real e-commerce software system to evaluate the usability of the approach. This section reports the case study.

4.1. The subject system

The subject of the case study is an e-commerce system of online trading of medicine. The system is operated by the local medicine trading regulation authority to supply medicines to all state-owned hospitals in the province. Its main functions include customer relationship management, product catalogue management, online trade management, online auction of medicine supplies, online information advertisement, a search engine for medicine information, and so on. The system was implemented in J2EE. The structure of its user management subsystem is shown in Figure 8.

4.2. Process of case study

The case study was conducted after the system was released and in operation for more than one year. It consists of the following activities.

(1) *Reverse engineering of the system to construct the architectural model of the system.* The design documents were reviewed as well as parts of the source code. The system's design document consists of four main parts: (a) a simple and schematic J2EE architecture showing that the system uses J2EE as implementation technology; (b) interface design, which consists of a lot of HTML files; (c) database design, which consists of about 70 database tables; (d) a simple UML class diagram that contains a dozen of classes and shows the logical view of the system. As some design information was not well documented, parts of the source code was reviewed for constructing an architecture model of the system, which was reviewed and corrected by some of the chief developers of the system, and then approved of its accuracy. Figure 8 shows a part of the architectural model for the user management sub-system.

(2) *Application of HASARD method and construction of quality model.* The architectural model of the system was then analysed using the HASARD method. The hazards of the components and connectors of the system were identified. The cause-consequence relationships between the hazards were recognised. The information was then transformed into a quality model in the graphical notation. After several iterations with the developers' reviews and revision, a quality model was constructed, which contains 70 nodes and 64 links between the nodes. For the sake of space, the details of the quality model are omitted in this paper.

(3) *Analysis of the quality model.* The quality model developed in the previous step was analysed by applying the SQUARE analysis tools to identify quality risks, quality trade-off points, and to derive the impacts of design decision on certain quality attributes and the contribution factors to certain quality attributes. More details are given in the next subsection.

(4) *Validation of analysis results*. The results obtained from quality analysis of the system were feed back to the developers of the system. A workshop was run to validate whether the outcomes of the quality analysis matches the reality in the development and operation of the system. It was found that all our findings were consistent with what has been observed in the operation of the system. Some of the phenomena observed in the operation of the system were first time satisfactorily explained through the architecture and quality model of the system. Based on the analysis results, a number of specific suggestions on the improvement of the system's architecture were made. Some of them were taken by the development team in the development of the new release of the system. Some would result in major changes of system's architecture and regrettably cannot be implemented within the budget of the new releases.

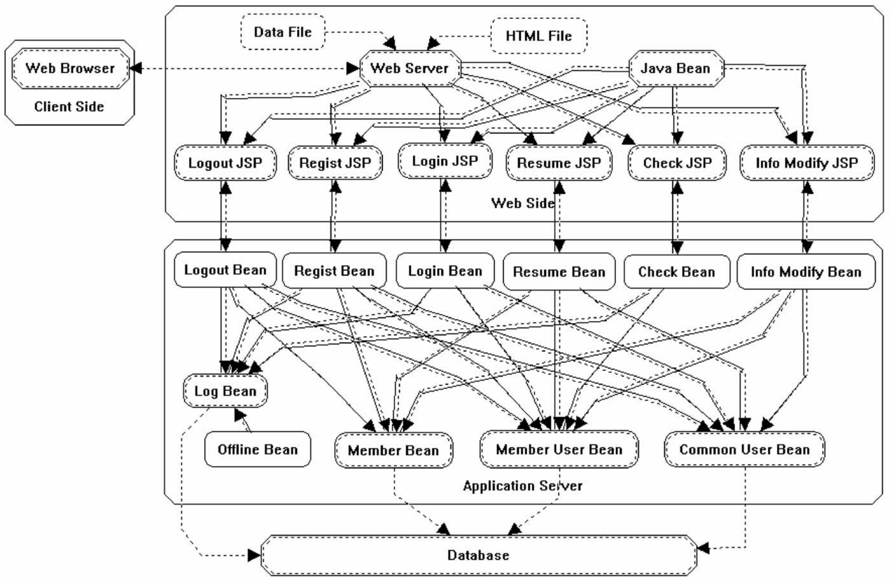


Figure 7. Architecture of user management subsystem

4.3. Main findings of quality analysis

In the analysis of the quality model using the tools provided by SQUARE, we discovered a number of quality issues. The following are some examples of the discovered quality issues.

(1) *Sensitive quality issues.* When concerned with the

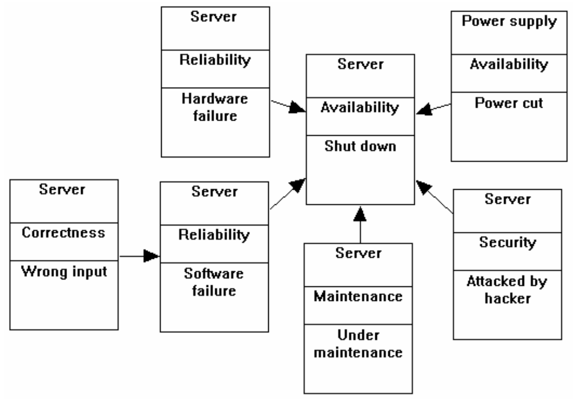


Figure 8. Quality factors that affect server's availability

problem that 'users cannot find desired information', we analyzed factors that may cause the problem by analysing the factors that affect the component "User" with usability as the component's property and "Cannot find info" as its phenomenon. The tool generated a sub-diagram that contains 35 nodes out of the 70 nodes in the quality model. This means that most components affect usability of the system. Consequently, we concluded that usability is a very sensitive quality issue in the design of the system. The generated sub-diagram provided detailed information about how properties of various components affect the usability of the whole system. Hence, it provided useful direction for how to enhance the usability.

(2) *Contribution factors that affect a quality attribute*. Intuitively, the server's availability is of particular importance to a number of other quality attributes. To find out what are the factors that affect server's availability, we applied the tool and generated the sub-diagram shown in Figure 9. The diagram shows that the factors that affect this quality attribute include hardware reliability, software reliability, power supply, system security, and maintenance. Therefore, we can conclude that necessary measures must be adopted to prevent hackers from attacking the server, to ensure a reliable power supply and the stability of server's hardware and software system to avoid the server crashes, and to provide maintenance tools to enable online maintenance facilities to reduce the time that the system has to be shut down for maintenance tasks.

(3) *Relationships between two quality attributes.* The quality model helped us to understand the relationships between quality attributes. For example, the quality model demonstrated that usability of the client side is affected by performance of the web server. So we must consider carefully on the system's hardware configuration and the deployment of software components onto the hardware cluster to balance the communication and computation load according to operation profiles.

(4) *Quality trade-off points*. In the analysis of the relationships between quality attributes, we found that the size of HTML files is a trade-off point. Because when the size is large, it has two different impacts on other quality attributes. One side, the HTML files of large sizes will make users find necessary information through fewer clicks. On the other side, the HTML files of large sizes also make the response time longer. Both of these are related to the usability of the system, but one has positive impact while the other is negative. Therefore, it is a trade-off point. Another trade-off point identified in the quality analysis is the granularity of the session beans. A small-sized session bean can only implement relatively simpler functions in comparison to larger sized session beans. Therefore, to complete a task, smaller session beans need to invoke more methods of other beans. This results in more execution time to complete a task. Consequently, the performance of the whole system also declines due to the time spent on creating other instances. On the other hand, if session beans are of a larger size, to serve the same number of clients, more memory will be consumed. Therefore, we can draw the conclusion that the granularity of session beans is a trade-off point between the response time of the system and the consumption of the memory space.

(5) *The impacts of a quality attribute*. In the case study we derived a large amount of information about the impacts of a quality attribute. For example, if the component of "Internet" has "heavy traffic", the usability and performance of the whole system will be affected.

(6) *Key quality issues*. In the analysis of the impact of a quality attribute, we found that the impacts of database's reliability are extensive as shown in the sub-diagram in Figure 10, which is created by SQUARE tool. It has effects on a wide range of issues ranging from business layer to presentation layer. So it's necessary to take some measures to avoid vicious attack and to ensure the stability of database server.

The above findings in the quality analysis of the system were all confirmed and agreed by the development team who are responsible for the development and maintenance of the e-commerce system.

5. CONCLUSION

In this paper, we represented the SQUARE automated software quality modeling and analysis tool based on the HASARD method. It enables software engineers to derive quality properties from software architectural designs. The method has the following features.

First, the quality models in the graph notation are much more expressive than existing hierarchical and relational representations of software quality models to represent complicated relationships between quality related properties. It can express system specific quality related information and relate quality issues to various components and connectors of the system.

Second, the derivation of the quality model from architectural design is a systematic and structured process although it is not formal. It adapted and ex-

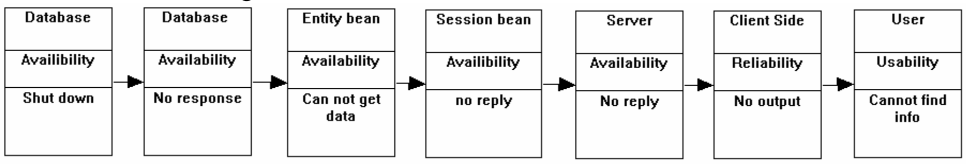


Figure 9. Effects of database's availability

tended the mature engineering methods of hazard analysis of safety critical systems to software systems. The quality models in the graphical notation can also be validated against design information. The annotation of the links provides a means of model validation.

Third, from the quality model, various quality issues can be automatically analysed, which include the identification of key quality issues, the contribution factors to a quality attribute, the impact of a quality attributes on the others, the quality trade-off points, the relationships between two quality attributes, etc. Such quality related features can provide detailed information for the assessment and improvement of software architectural designs.

The software tool SQUARE supports the quality model construction and analysis in the HASARD method. The tool implemented a set of algorithms to support automatic analysis of quality models. These algorithms are simple graph algorithms and of high performance. Details of the algorithms will be reported separately due to the lack of space.

A case study of the HASARD method and the SQUARE tool has been conducted. The case study investigated the quality of a real e-commerce system independent of the subject system's development team. Software developers confirmed that the findings of the case study were consistency to their independent observations on the system during operation and maintenance. It clearly demonstrated the applicability of the method and the tool to real software systems.

We are further investigating the automation of more activities in quality model construction and analysis to improve the intelligence in the identification of key quality issues, trade-off points, etc. The SQUARE tool currently uses architectural models in Software Architecture Visual Notation [16] that represents the conceptual view. We are also further investigating how architectural models representing other views can be analysed.

ACKNOWLEDGEMENT

The work reported in this paper is partly supported by China High-Technology R&D Programme (863 Programme) under the grant 2002AA11607.

REFERENCES

[1] B. Kitchenham, and S. L. Pfleeger. Software Quality: The Elusive Target. *IEEE Software*, 13(1):12-21, January 1996.

[2] J. McCall, P. Richards, and G. Walters. Factors in Software Quality. *Technical Report* CDRL A003, US Rome Air Development Centre, Vol.1, 1977.

[3] B.W. Boehm, J. Brown, H. Kaspar, M. Lipow, G. MacLeod, and M. Merrit. Characteristics of Software Quality. *TRW Series of Software Technology*, Vol. 1, North-Holland, New York, 1978.

[4] *International Organisation for Standardization.* ISO 9126: Information Technology--Software Product Evaluation - Quality Characteristics and Guidelines for Their Use, 1992.

[5] J. Bansiya, and C. G. Davis. A Hierarchical Model for Object-Oriented Design Quality Assessment. *IEEE TSE*, 28(1):4-17, January 2002.

[6] W.E. Perry. *Quality Assurance for Information Systems: Methods, Tools and Techniques*. John Wiley & Sons, 1991.

[7] A. Gillies. Modelling Software Quality in The Commercial Environment. *Software Quality Journal*, 1:175-191,1992.

[8] A. Gillies. *Software Quality: Theory and Management*, 2nd Edition. International Thomson Computer Press, 1997.

[9] Y. Zhang. Quality modelling and metrics of Web Information Systems. *PhD Thesis*, Dept of Computing, Oxford Brookes University, Oxford, UK, April 2005.

[10] R. G. Dromey. A Model for Software Product Quality. *IEEE TSE*, 21(2): 146~162, February 1995.

[11] R.G. Dromey. Cornering the Chimera. *IEEE Software*, 13(1): 33~43, January 1996.

[12] N. Lassing, D. Rijsenbrij, and H. van Vlient. On Software Architecture Analysis of Flexibility, Complexity of Changes: Size Isn't Everything. *Proc. of Second Nordic Software Architecture Workshop* (*NOSA'99*), 1103–1581, 1999.

[13] S. Bot, C. -H. Lung, and M. Farrell. A Stakeholder-Centric Software Architecture Analysis Approach. *Proc. Int'l Software Architecture Workshop* (*ISAW 2*), pp152-154, 1996.

[14] J. Bosch. *Design & Use of Software Architectures: Adopting and Evolving a Product-Line Approach.* Addison Wesley, 2000.

[15] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. *IEEE Software*, November, 1996.

[16] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, 1998.

[17] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures-Methods and Case Studies.* Addison Wesley, 2002.

[18] L. Dobrica, and E. Niemela. A survey on software architecture analysis methods. *IEEE TSE*, 28(7): 638–653, 2002.

[19] T. Kostelijk. Misleading Architecting Tradeoffs. *IEEE Computer*, 38(5): 20-26, May 2005.

[20] H. Zhu, Y. Zhang, Q. Huo & S. Greenwood. Application of Hazard Analysis to Software Quality Modelling. *Proc. of COMPSAC'02*, pp139~144, August 2002 .

[21] H. Zhu. *Software Design Methodology: From Principles to Architectural Styles*. Elsevier, 2005.

[22] T. Kletz. *Computer control and Human Error*. Gulf Publishing Company, Houston,, 1995.

[23] N. G. Leveson. *Safeware: System Safety and Computers*. Addison Wesley, Reading, MA, 1995.

[24] P. G. Neumann. *Computer-Related Risks*. ACM Press, New York, 1995.

[25] N. Storey. *Safety-Critical Computer Systems*. Addison, Reading, MA, 1996.

[26] Ministry of Defence. *HAZOP Studies on Systems Containing Programmable Electronics.* Defence Standard 00-58, Issue 2, 19 May 2000.

[27] Q. Zhang and H. Zhu, Automated Analysis of Software Designs with Graphic Quality Models, *Proc. of The 10th WSEAS International Conference on Computers*, Vouliagmeni, Athens, Greece, July 13-15, 2006. (*In press*)