International Journal of Digital Information and Wireless Communications (IJDIWC) 1(1): 109-125
The Society of Digital Information and Wireless Communications, 2011 (ISSN 2225-658X)

Implementing Test Management Traceability Model To Support Test Documents

Azri Azmi, Suhaimi Ibrahim
Advanced Informatics School (UTM AIS)
Universiti Teknologi Malaysia
54100 Kuala Lumpur, Malaysia
{azriazmi, suhaimiibrahim} @utm.my

ABSTRACT

Documentation is one of the key quality factors in software development. However, many developers are still putting less effort and less priority on documentation. To them writing documentation during project development is very tedious and time consuming. As a result, the documentation tends to be significantly out-dated, poor quality and difficult to access that will certainly lead to poor software maintenance. Current studies have proved that the key point to this problem is software traceability. Traceability relates to an ability to trace all related software components within a software system that includes requirements, test cases, test results and other artefacts. This research reveals some issues related to current software traceability and attempts to suggest a new software traceability model that focuses on software test documentation for test management. This effort leads to a new software test documentation generation process model based on software engineering standards.

KEYWORDS

Software Traceability, Software Documentation, Software Testing, Software Maintenance, Test Management.

1 INTRODUCTION

Nowadays software is becoming more complex. It consists of diverse components with distributed locations, complex algorithms, on varieties of platforms, many sub-contractors with different kind of development methodologies and rapid technology innovation. The cost and risk will become higher in software development project with this kind of complexity as reported by Boehm [1]. It is vital to ensure the reliability and correctness the software being developed. Such aims can be reached using documentation as tools. Documentation is a detailed of descriptions of particular items and used to represent information such model, architecture, record artefacts, maintain traceability of requirement and serial decisions, log problems and help in maintaining the systems.

Software developers rely on documentation to assist them in understanding the requirement, architecture design, coding, testing and details of intricate applications. Without such documentation, engineers have to depend only on source code. This will consume time and lead to make mistakes [2] especially when developing large scale systems. As reported by Huang and Tiley [3] and Sommerville [4], there are several shortcomings in current documentation such as out-of-date, inconsistency between source code and documentation, poor quality and others.

The key point solution to the above problems is software traceability. Traceability is defined as the ability to link between various artefacts in software development phases linking requirements, design, source code and testing artefacts. In the early seventies, requirements traceability was driven mainly obligatory policy such DoD2167A for US military systems [5]. Later, many institutions recommended traceability IEEE Standard, SPICE, CMM/CMMI have gathered more awareness. Today, software traceability has become one of the key attributes to software quality. Unfortunately, many organizations failed to implement effective traceability due to difficulties in creating, assessing, using and maintaining traceability links [6, 7]. The accurate traceability practices can help in maintaining software. Having an accurate documented traceability links between software artefacts is essential for a various software maintenance activities including impact analysis. Thus it will improve the quality of system as well as the software process. On the other hand, neglecting traceability can lead to reduce the quality of the software product. The quality of the software product cannot be achieved when it is not fully tested and traced with the requirements.

In this paper, we present an implementation of software traceability model that support test management in generating software testing documentation base on software engineering standards.

2 RELATED WORKS

As development becomes complex, the task of connecting between requirements and various artefacts becomes tedious and sophisticated. The IEEE Standard Computer Dictionary [8] defines traceability as *"The degree to which a relationship can be established between two or more product of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match*". Meanwhile [9] defines traceability in much broader definitions. Aizen defined traceability as any relationship that exists between artefacts that involved the software development life-cycle.

There are many benefits of software traceability. Most commonly, it is claimed to help in change management [10-12], system verification [7, 13], help in performing impact analysis [12], reuse of software artefacts[14] and meets the need of the stakeholders [7, 15, 16]. With support of sophisticated tool that capable to store and retrieve links, traceability still facing several issues such [17]: (i) the process of tracing is still done manually, (ii) missing of information to be traced, (iii) engineering issues that could arise later, so the trace information may be insufficient. Traceability is referenced in many software development and standards, however specific requirements or guidelines on how it should be implemented are rarely provided.

The IEEE Standard Computer Dictionary [8] defines model as "*An approximation, representation, or*

idealization of selected aspects of the structure, behaviour, operation, or other characteristics of a real-world process, concept, or system". Meanwhile, a broader definition from Microsoft Computer Dictionary [18] stated that model as "*A mathematical or graphical representation of a real-world situation or object -for example, a mathematical model of the distribution of matter in the universe, a spreadsheet (numeric) model of business operations, or a graphical model of a molecule. Models can generally be changed or manipulated so that their creators can see how the real version might be affected by modifications or varying conditions.*" In summary, model is a hypothetical description of complex entity or process. Software traceability model is an obligation for an organization before any development begins. It arises from the complexity of development such varieties of platforms, distributed locations, tools and technology used, organizational structure, different organizations, policies, standards, and development methodologies.

As discussed in previous section, traceability is crucial to verify and validate the completeness and consistency with the requirements. It can provide significant benefits, if it is properly implemented. Therefore, a strategy for implementation of software traceability must be carefully defined. Different organizations or projects will have different set of traceability model depending on business/organizational, domain, project, product or technology [19]. The key point of implementing efficacious software traceability includes linkage of data and artefacts, semantic content, and automation capability. Traceability will be able to be achieved when all these aspects are addressed. After performing analysis of the existing model, a number of findings and limitations of existing software traceability models were identified. (Detailed of comparative study is tabulated in Table 1). There are several software traceability model discussed by researchers and next subsections will discussed them.

Narmanli [20] presents a traceability model to support requirements-to-requirements traceability known as Inter Requirements Traceability Model (IRT). The model consists of three types of software requirements such as use cases (GUI, test data, qualification), business rules (business-rule engine, test data, qualification test procedures) and data definitions (entities, database, test data). Figure 1 depicted the model. The traceability model proposed bidirectional traces of these types. The model tries to minimize the effect and the workload of change requests on implementation and tests to satisfy both customers and development teams. The contribution for this model is that it support for change request impact analysis. Advantages of using this model are: i) minimize the effect and the workload of change requests on implementation and tests when constructing the software design, ii) ease in *make-buy-reuse* analysis, iii) effective tests, where every business rule is ready to become a test procedure. Meanwhile, the drawback of this model is that it only supports traceability between requirements only.

.

Table 1. Comparative Study of Traceability Model

Model /Item	IRT	TT	CPRT	ETET	TW	TMT
Tracing Item	Business Rule, Use Case and Data Definition	Method, Class, Package	Source Code, Requirements	Documents, artefacts repository	Artefacts at different level of granularity	Requirement, Design, Code, Test Documents
Tracing Type	R-R	R-D-C-TC	R-C	MR-UC-R-TC	R-D-C-T	R-D-C-P-TA
Tool Support	n/a	Yes, CATIA	n/a	Yes, no name given	Yes, combination of TMS, APXTMS, HWTMS	Yes, Implementation Phase
Case Study	Change request of SRS (no name given)	OnBoard Automobile (OBA)	Not conducted yet	Real project at Wonderware (small s/w development company)	Real project at Alcatel-Lucent	OnBoard Automobile (OBA)
Support	Change Impact Analysis	- Software Maintenance from system documentation - Change Impact Analysis	Improve Software Quality by validating and verifying requirements	Prescriptive workflow	Notification mechanisms and configurable control	-Test document generation based on SE standard -Test Management
Strength	-	Support traceability through SDLC	Efficient way of tracking and tracing requirements	Post-requirements traceability supporting SDLC	-Effortless change impact analysis -simpler maintenance for large volume data	-pre and post requirements traceability -support whole SDLC
Limitation	Tracing only on requirement to requirement	No tracing for pre-requirements	Tracing only on requirement to source code	-no tracing in maintenance phase and pre-requirement -tracing text-base artefacts only	Applicable to small team (agile methodologies)	-

C=Code D=Design MR=Marketing requirements R=Requirements T=Testing TC=Test Cases UC= Use Cases P=Plan

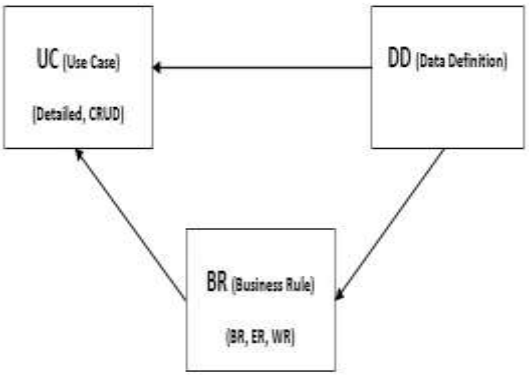

Figure 1. Inter-requirements Traceability Model

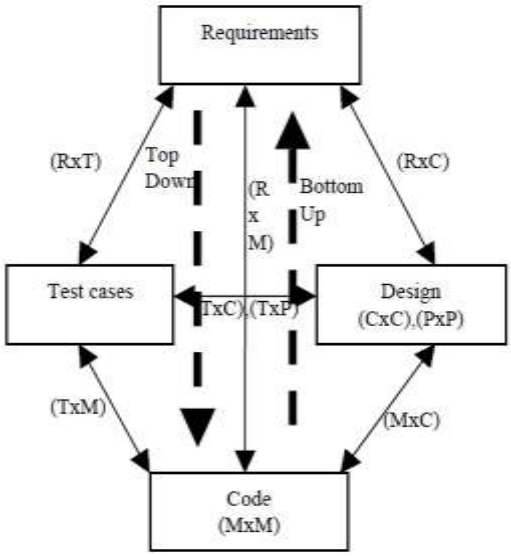

Figure 2. Total Traceability Model

Ibrahim et at.[21] introduced a model that derived requirement traceability from system documentation. This model is called a Total Traceability Model (TT). It provides links of different artefacts that include requirements, design, test cases and source code. It uses horizontal traceability and vertical traceability to make up a total traceability. The model capture relationships across different levels of artefacts before an impact analysis can be implemented. The process of tracing and capturing these artefacts is called hypothesizing traces. Figure 2 describes the traceability model. The horizontal relationships can occur at the cross boundaries as shown by the thin solid arrows, while the vertical relationships can occur at the code level and design level respectively. The thick doted lines represent the total traceability that need to implement in either top down or bottom up tracing. Tools such *McCabe* and *Code Surfer* were used to help in capturing the dependencies. The tracing type is between requirement-test cases, test case-code, method-method and class-class. A prototype *CATIA* has been developed to demonstrate the model. A significant contribution of this model is that it ability to support top down and bottom up tracing from a component perspective.

Meanwhile, Salem [22] in his research established a traceability model that provides an intuitive and dynamic way of requirements traceability during software development process. This model is named as Coding Phase Requirements Traceability Model (CPRT). The model composed of a *Traceability Viewer Component* (TVC), a *Traceability Engine Component* (TEC) and *Quality Assurance Interface* (QAI) as illustrated in Figure 3. The TEC is used to help developers to link source code element with the software requirements. Meanwhile, TVC acts as viewing medium to view links between requirements and source code. It provides software engineer with a distinctive way to scrutiny all the information that TEC has gathered. Lastly, QAI is the component that validates and verifies of requirements.

A flagging procedure is designed using *Requirement Flags* to provide traceability between requirements and source code. The preliminary model provides a simple interface that allows developers to seamlessly locate the correct requirements and link them to the correct source code elements. Limitation of the model is that only trace links between requirements and source code.

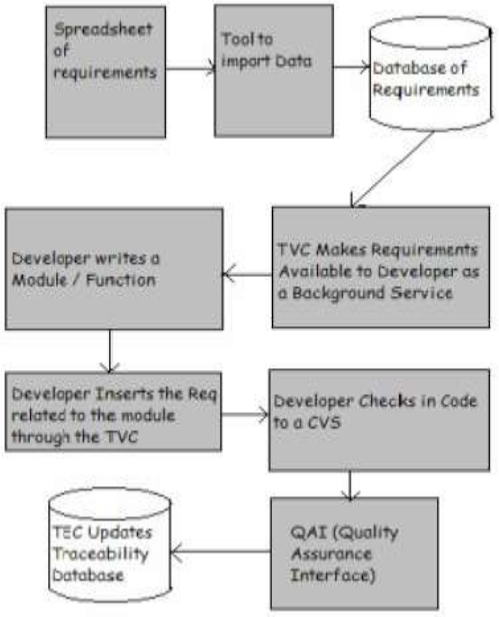

Figure 3. Coding Phase Requirements Traceability Model

Asuncion et al. [23] proposed end-to-end traceability model (ETET). This process-oriented model achieves comprehensive traceability and supports the entire software development life cycle (SDLC), from the requirements phase to the test phase by focusing on both requirements traceability and process traceability. It emphasized on *process traceability* as an important facet of effective requirements traceability. Three main goals were set to be achieved by using this traceability model. First, minimize overhead in trace definition and maintenance, followed by preserve document integrity and lastly to support SDLC activities. A successful prototype tool has been developed to demonstrate the model. It used bidirectional updates between documents and the artefact repository to guarantee the document integrity. Figure 4 describes the end-to-end traceability model. A tool has been developed at Wonderware, a mid-sized software development company. Limitation of the model is that only support for post-requirements traceability. The boxes at the top represent the global trace artefacts and solid lines represent their requirements trace links. Meanwhile, the users of the system shown at the bottom of diagram and all of them are consumers of trace information.

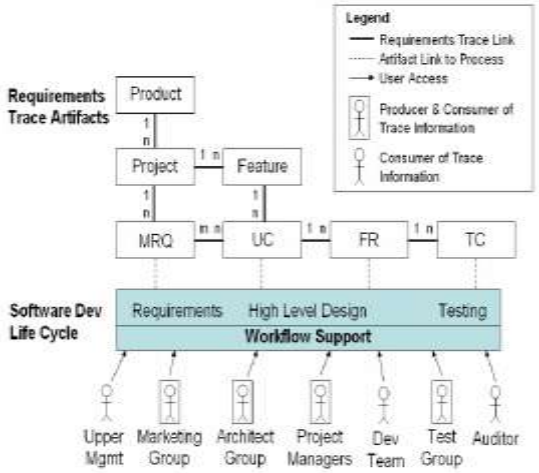

Figure 4. End-to-End Traceability Model

Finally, TraceabilityWeb model (TW) was introduced by Kirova et at. [19]. They have examined the traceability problem-solution in the context of multiple aspects such business

/organization, project, product, development model and technology. A tool called TraceabilityWeb has been developed to demonstrate the feasibility and the benefits of integrated tools environments, which automate the creation and maintenance of traceability information. It also provides enough content to start test in early phase compared to traditional approach. These include test planning, test creation and design specification efforts. The model also established benefits such as effortless change impact analysis, simpler maintenance of large volumes of data, flexible levels of granularity when creating links, metrics and reporting. Figure 5 illustrates the traceabilityWeb model. It integrates multiple artifact repositories, including requirements management systems, test management system and databases. It also auto generates a significant portion of artifact mappings and supports most functional areas such system engineering, architecture, development, test, product management and test management. The drawback of this model is that it only capable to small teams such agile methodologies.

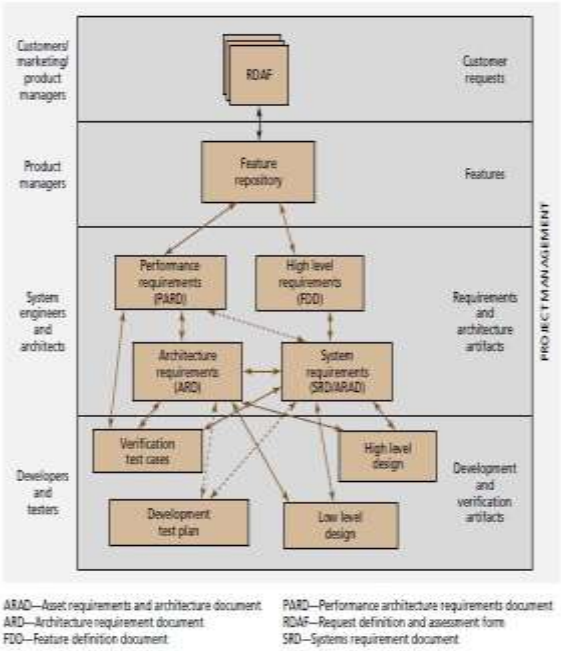

Figure 5. TraceabilityWeb Model

Documentation is a written material that serves as a record of information and evidence. Software engineering documentation encompasses not only source code, but also all intermediate work products associated to the code and its validation and operation, such as contracts, design architectures and diagrams, reports, configurations, test cases, maintenance logs, design comments, and user manuals. According to Wang [24] documentation is a software engineering principle that is used to embody system design and architectures, record work products, maintain traceability of serial decisions, log problems and maintenance solutions, and enable post-mortem analysis. While Forward [25] in his thesis defined software documentation as an artefact whose purpose is to communicate information about the software system to which it belongs. From the Agile perspective, a document is any artefact external to source code whose purpose is

to convey information in a persistent manner [26].

Software engineers need to rely on documentation to aid in understanding the software systems. Regrettably, the artefacts available for them usually are out of date and therefore cannot be trusted [2]. Developers need to depend solely on source code because of unavailability of such documentation. Thus, the process becomes an error prone and time consuming, especially when dealing with manual tracing the traceability link for large scale systems. Sulaiman et al. [27] in her survey stated three main reasons why software documentation were not produce during software development; time constraint, commercial pressures and not requested by supervisors. Other reasons are not requested by customers, tedious task, too costly to keep updated, boring task, and more.

3 EVALUATION OF SOFTWARE TRACEBILITY MODEL

Before a new model can be proposed, an evaluation on existing models needs to be investigated and evaluated. Therefore, the way of assessment needs to be determined. The software evolution taxonomy [28] has been used in order to evaluate the traceability model together with perspectives introduced by Hazeline[29]. The taxonomy of software evolution is based on multiple dimensions characterizing the process of change and factors that influence these mechanisms while the perspectives discuss on economic, technical and social perspectives. The evaluation criteria are based on accessibility (mapping between artefacts), capturability (degree of automation), tool supportability, temporality (when to trace) and scalability. The results of evaluation are tabulated in Table 2. The rational of choosing the criterion is explained in details in next paragraphs.

The accessibility criterion is evaluated to determine whether a model can be mapped between various artefacts in software development life cycle. Such documents are requirements specification, software design, source code, and test suits. Of the observations made, it appears that the entire models are capable to link requirements to design, source code and test suits except for IRT and CPRT. For CPRT, tracing is between requirements and source code only. While the IRT, the link is between the requirements and requirements. Meanwhile, the criterion capturability is to help in analyzed, managed, control, implement or measure changes in software. The mechanisms for this case can be automated, semi-automated or manual. Result of the comparison made shows for all models, links are formed in a semi-automatic and there is no link made manually. For TT, ETET, TW and TMT, links can be generated automatically.

Table 2. Evaluation of Software Traceability Model

Features		IRT	TT	CPRT	ETET	TW	TMT
Accessibility							
(i)	Requirement	√	√	√	√	√	√
(ii)	Design		√		√	√	√

(iii)	Source Code		√	√	√	√	√
(iv)	Test Suits		√		√	√	√
Capturability							
(i)	Automatic		√		√	√	√
(ii)	Semi	√	√	√	√	√	√
(iii)	Manual						
Tool Supportability			√		√	√	√
Temporality							
(i)	Development	√		√	√	√	√
(ii)	Maintenance		√			√	√
Scalability			√		√		

As for the next criterion is tool supportability, which is evaluating whether the model provides tool support to accommodate links between artefacts. The purpose of the tool support is to help in visualizing and managing traceability. On this criterion, the entire model can be supported using tool except IRT and CPRT. The next criterion is temporality. Temporality refers to time and when a link is created or updated. TW and TMT allow links in both development phase and maintenance phase. Meanwhile, other models such IRT, CPRT and ETET only in the development phase. In addition, the TT model dedicated in maintenance phase. Finally is the scalability criterion. This criterion is to analyze whether the model can be applied to large-scale projects. The result showed that TT and ETET models can be applied to large-scale systems compared to others only to small and medium-sized projects.

4 RESULTS

This research is intended to create a traceability model that will be used in order to generate a software testing documentation based on Software Engineering Standards. As such, a preliminary study was conducted in finding an approach that can suite the traceability within software testing artefacts that lead to establish a repository. Software traceability has been used by many researchers and practitioners and it is a key factor for improving software quality. There are numerous benefits of using traceability such as to keep documentation updated and consistent within artefacts, enabling requirements-based testing, early revision of requirements testing, and improve in management of change impact analysis etc. Despite these advantages, traceability is hard to implement in software development. There are weaknesses in current approaches and models [23]. There is a lack of research on traceability in finding relationships between software testing artefacts.

Several tools and research prototypes have been analyzed and compared in order to find the similarities and differences in previous paper [30, 31]. Out of all, there is only one prototype was closely similar to this proposed study. PROMETUE [32] is a prototype tool that was developed to introduce or improve the software testing process of a small software company. The artefacts and information elements to trace are based on IEEE829-1998. However, PROMETUE was developed to support

traceability within artefacts such of documents and requirements only. Our proposed research is to establish a traceability model that governs various artefacts such source code, documents, testing tools files, requirements, legacy systems and stakeholders.

5 DISCUSSION

Figure 6 shows the proposed model (Test Management Traceability - TMT) that will generate software testing artefacts based on Software Engineering Standards. There are four main components namely *Traceability Engine*, *Analyzer*, *Extractor/Parser* and *Document Generator*. The proposed model illustrates that all the data are gathered and stored in a repository. Firstly, the tool will analyze the information to be stored and will create a repository of traceability links. The stored data in the repository may come from a variety of sources and format such as: (i) source code (Java and C++), (ii) software documents such as Software Development Plan (SDP), Interface Requirements Specification (IRS), Software Requirements Specification (SRS), Software Design Descriptions (SDD), Software Test Result (STR), Software Test Descriptions (STD), (iii) legacy systems, (iv) stakeholders/users, (v) output files from testing tools, (vi) requirements and (vii) experts.

Extractors/Parsers as agents will be used in order to extract the desired information to be converted into eXtensible Markup Language (XML) as a raw format. The XML files will be used by analyzers to create the traceability among artefacts and the output will be saved into a repository called a traceability repository. This repository then will be used as an input to the process of generating software testing document. The document generator will be developed in an integrated environment with the template repository to produce a software testing documentation. The next paragraphs will explain in more detail of the proposed model components.

Parser task is to analyze continuous flow of input and breaks into constituent parts. Several parsers will be used to convert multiples format of input data into XML format. It will support the capture, summarization and linking of software artefacts information. This support for extracting information from wide variety of source artefacts, viewing it in summarization form, manages changes to the artefacts in different representations. Software information sources may include format of Microsoft Word™, Excel™, modeling application such Rational Rose™, testing application such SpiralTeam™, TestSuite™ and Robot™. It also may include of an email files, data from legacy systems or a text files (*.txt*). The goal is to gather all artefacts that available into one central repository to streamline test management approach. All the data stored in the raw repository will be in XML format. XML is designed to transport and store data and is acknowledged as the most effective format for data encoding and exchange over domains ranging from internet to desktop applications. The hierarchical nature of XML documents provides useful structuring mechanisms for program artefacts. It is chosen because of easily processed by commonly

available third-party tools such editors, browsers or parsers.

In order to generate documentation from an artefact, an analyzer is needed to analyze the data. There are some existing approaches being made available to analyze the data, such *lexical analysis, syntactic analysis, island grammars,* and *parse tree analysis*. The most appropriate approach will be determined during the implementation phase. Meanwhile, the input data for traceability engine will be a traceability repository or corpus. In this repository, relationships or link across artefacts will be kept. A unique key will be given to each requirement related to it. Several items or artefacts such test cases, design item (module, class, packages), and codes may refer or link to one requirement. In order to define this repository, a structure must first be defined. Traceability link meta-model will be used as shown in Figure. 7 that inspired by Valderas [33].

According to the meta-model, traceability link presents as *Identifier* and will be develop using relational database. MS SQL will be used as database to store all artefacts. Before the data is stored in the database, we need to trace and capture their relationships among artefacts. The process of tracing and capturing these artefacts is called hypothesizing traces that was introduce by Ibrahim [21]. Figure 8 depicted one way of hypothesizing traces. It can be described in the following sequence. For a selected requirement, choose a test case. One requirement might have several test cases. Then clarify with other documentations such plan, design and codes followed by observe traces with code. From this, link can be generated.

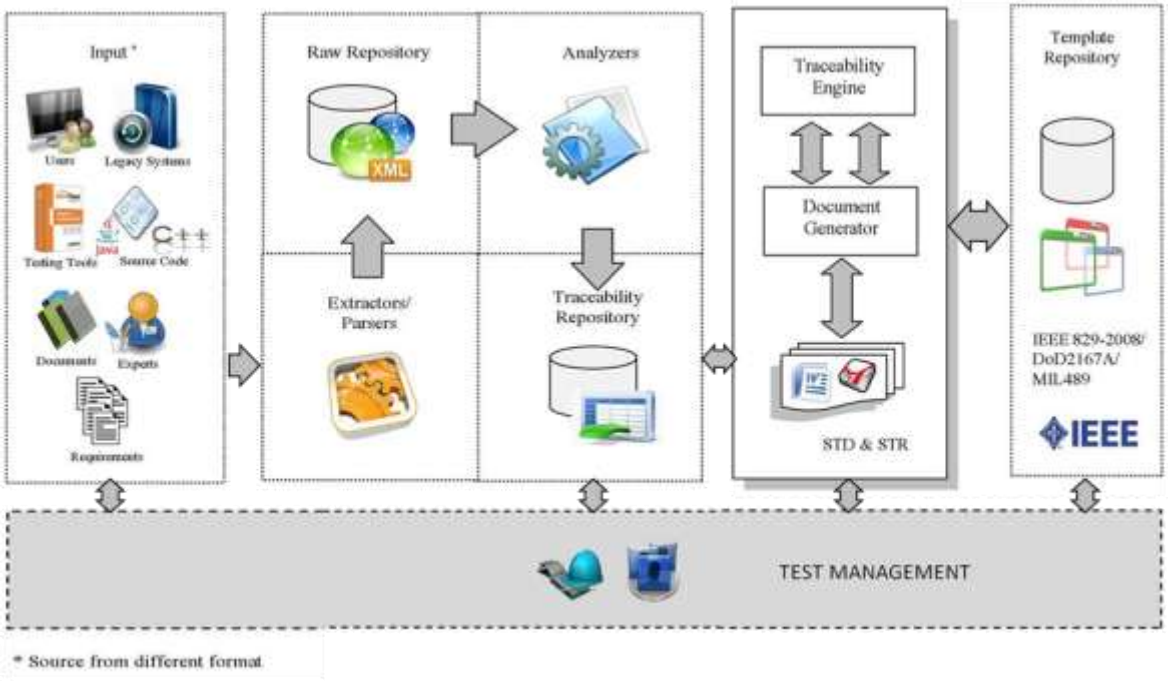

Figure 6. Test Management Traceability Model

An information retrieval (IR) method will be utilized to drive the traceability link. A distinct advantage of using this traceability method is that it does not rely on a predefined vocabulary or grammar for the documentation. As a consequence, this method can be applied without large amount of pre-processing of the input. A method of IR so called Latent Semantic Indexing (LSI) will be used. A traceability engine involves with task of setting traceability elements specifically designed to link with other artefacts to constitute some traceability links in a repository. In order to create this repository, a specific structured must be defined first. This structure must be used to store information relating to traceability links.

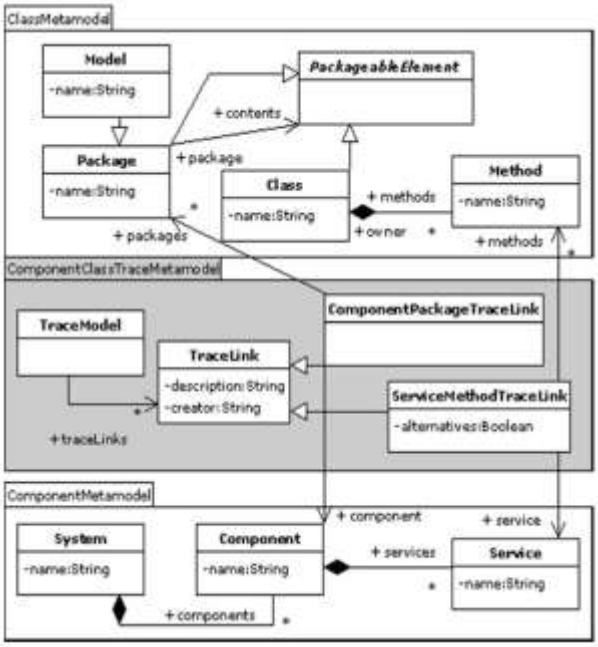

Figure 7. Traceability link meta-model

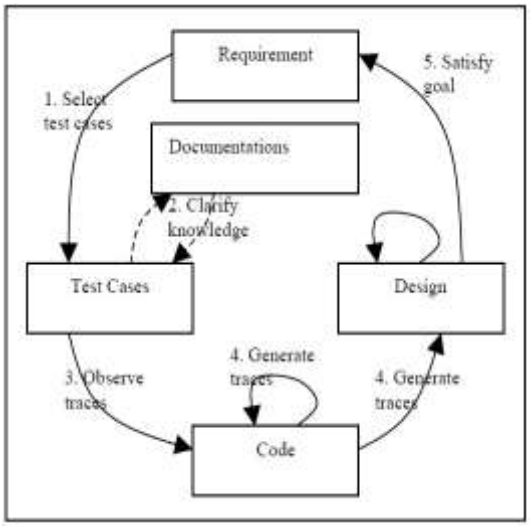

Figure 8. Hypothesized traces

Document generator is a process of generating documentation based on Software Engineering Standards. It will generate document such STD and STR based on the template repository. Existing artefacts and documents are also used as input to the data gathering process. Generate documentation is a process of collecting data and information, analyzes it, combining this information with other resources, extrapolate new facts, and generating updated documentation. By generating documentation when it is needed, software engineers can simply acquire documentation when they need it without worry about cataloguing, storing or sharing a repository of documentation. The generated documents need to preserve its integrity. It is carried out by using bidirectional updates between documents and the traceability repository.

5.1 Case Study : OBA

The On Board Automobile (OBA) system will be used in order to validate the proposed model. OBA is a final year project for masters student at Advanced Informatics School (UTM AIS), Universiti Teknologi Malaysia International Campus, Kuala Lumpur and widely used worldwide such Thales Universitè (France), Institute Teknologi Bandung - ITB (Indonesia), Thailand, Brunei, UEA and others. OBA is a simulation system which interacts with actor to activate the auto cruise functionality such as operate cruise control, calculate fuel consumptions, calculate average speed and perform car maintenance. It is written in C++ or Java and used UML notation throughout the entire SDLC. The project was built with complete set of documentations (artefacts) such SDP, IRS, SRS, SDD, STD and STR based on DoD2167A standards and MIL-STD-498. The documents provided us with some useful information on artefacts traceability between the documents and requirements. Figure 9 reflects the notion of our traceability architecture that establishes relationships between various software artefacts. The dashed thick arrows indicate the relationship between artefacts while the thin arrows shows the relationship between one requirement in every phase from planning phase until the testing phase.

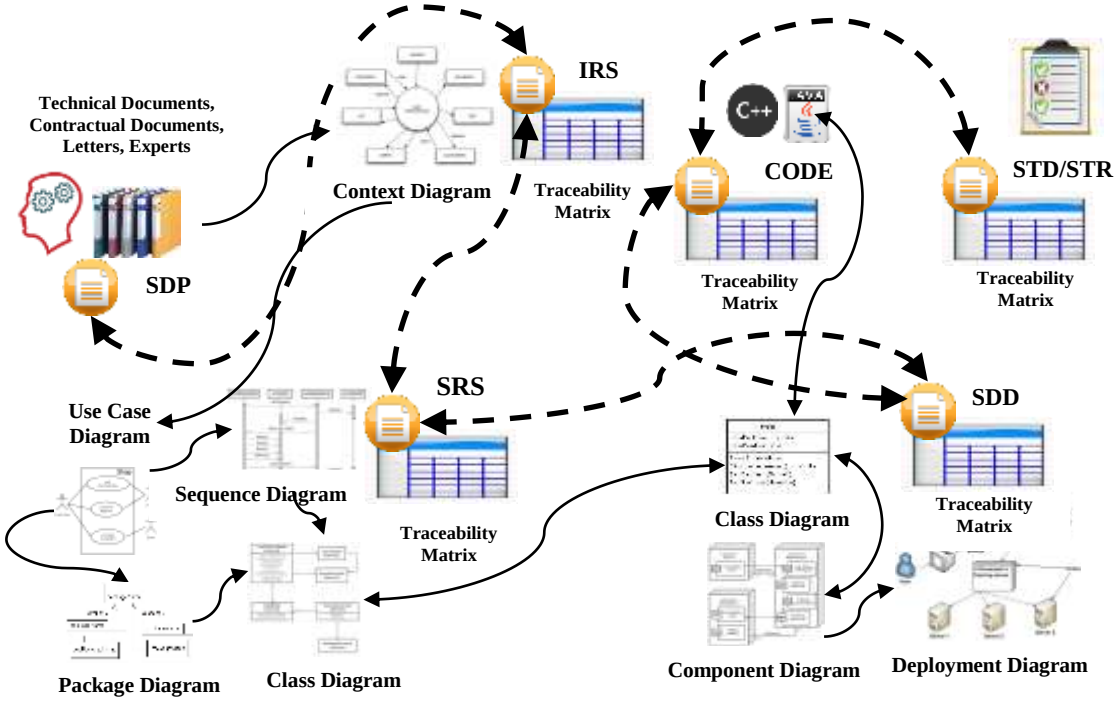

Figure 9. Traceability Architecture

We believe that there exists some relationships among software artefacts in OBA system. Therefore, we need to map out and capture their links not only within the same level but also across different level of phases and artefacts. The process of tracing and capturing these artefacts is called hypothesizing traces. Figure10 represent an example of mapping one requirement in OBA system so called *Cruise Control* in different phases and different artefacts. Firstly, *use cases* are indentified based on technical documents, contractual documents and letters from client or stake holders. In this case, the *Cruise Control* is given a reference number as SRS_REQ-03-00 or so called requirements number. The reference number will be stored in traceability matrix table and will be used to trace its origin sources. From *use case* to *test cases* of *Cruise Control*, we can trace the relationship forward and backward and in each phases and artefacts, it will be given a unique reference number for the purpose of tracing. For instance, in the SDD the reference number will be SDD_REQ-04-00 and STD_REQ-02 in the STD, etc.

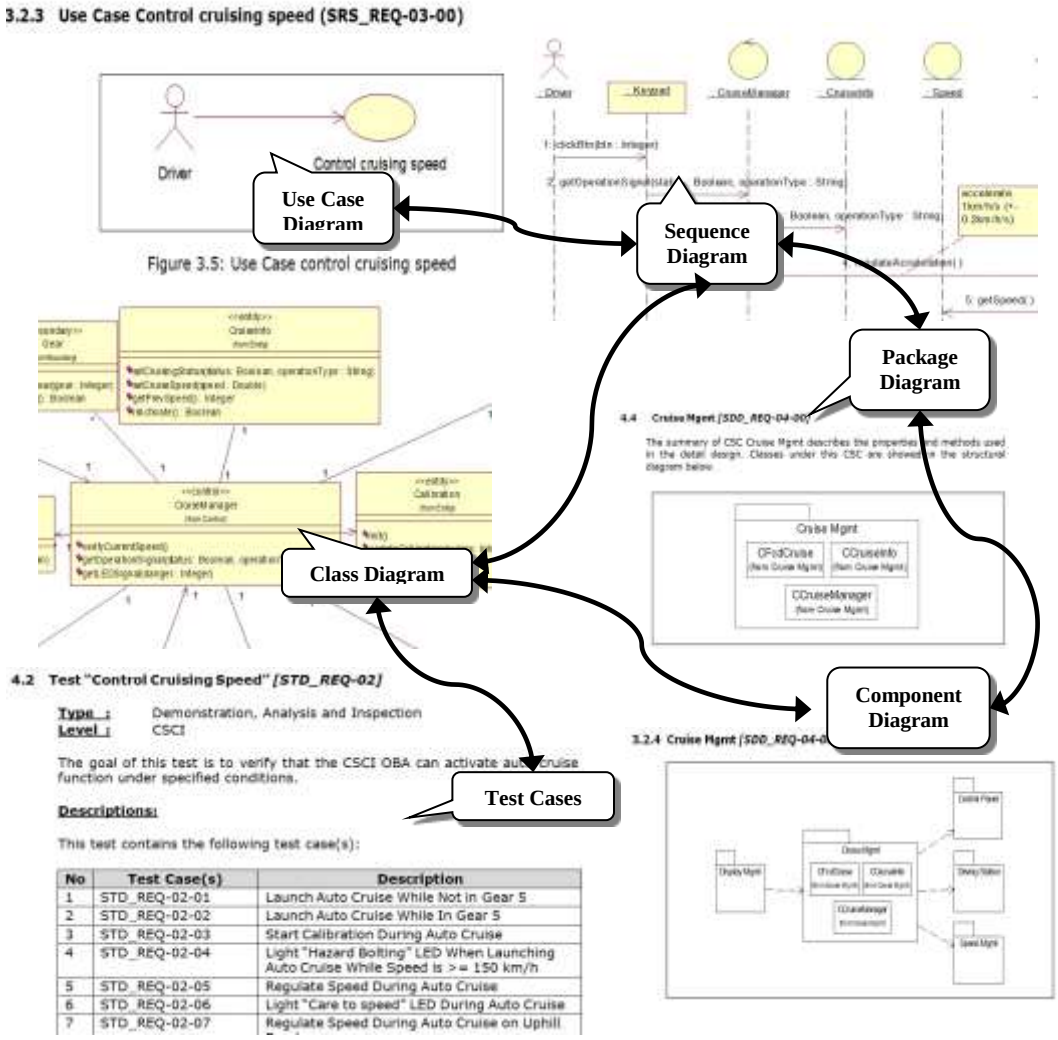

Figure 10. Control Cruise Traceability from Use Case to Test Cases

5.1 Expected Findings

This research is expected to establish a software testing documentation process model using a traceability approach. The findings include: (i) Defining documentation features based on software engineering standards. (ii) an evaluation result on existing approaches and models, (iii) established a new comprehensible process of integrated system as a proposed solution, (iv) a tool to be developed to support test management for software testing activities.

6 CONCLUSION

Software documentation is vital for software engineers to help them in producing a good quality of system software. Without such aid, they are solely relying on source code that leads to error prone and time consuming. A

key point here is to establish a workable traceability model and approach to meet the demand of software documentation. The correct traceability use can help a lot of activities within the development process. It can improve the quality of the product as well, as the software process. Traceability practices in general are far from mature and a lot of researches need to be done. A new traceability model is expected to support software testing documentation that will certainly be useful in software maintenance activities.

7 REFERENCES

1. Boehm, B.: Value-based software engineering: reinventing. ACM SIGSOFT Software Engineering Notes. 28, 3 (2003).
2. Thomas, B., Tilley, S.: Documentation for software engineers: what is needed to aid system understanding? Proceedings of the 19th annual international conference on Computer documentation. p. 236 (2001).
3. Huang, S., Tilley, S.: Towards a documentation maturity model. Proceedings of the 21st annual international conference on Documentation. pp. 93-99. ACM, San Francisco, CA, USA (2003).
4. Sommerville, I.: Software engineering, vol 2: the supporting process. Addison-Wesley. ISBN 0-321-31379-8 (2002).
5. Albinet, A., Boulanger, J.L., Dubois, H., Peraldi-Frati, M.A., Sorel, Y., Van, Q.D.: Model-based methodology for requirements traceability in embedded systems. 3rd ECMDA workshop on traceability, June (2007).
6. Knethen, A., Paech, B.: A survey on tracing approaches in practice and research. IESE-Report No. 095.01/E. 95, (2002).
7. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Transactions on Software Engineering. 27, 58–93 (2001).
8. Geraci, A.: IEEE standard computer dictionary: compilation of IEEE standard computer glossaries. Institute of Electrical and Electronics Engineers Inc. (1991).
9. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Systems Journal. 45, 515–526 (2006).
10. Cleland-Huang, J., Chang, C.K., Sethi, G., Javvaji, K., Hu, H., Xia, J.: Automating speculative queries through event-based requirements traceability. Proceedings of the IEEE Joint International Requirements Engineering Conference (RE'02). pp. 9–13 (2002).
11. Chang, C.K., Christensen, M.: Event-based traceability for managing evolutionary change. IEEE Transactions on Software Engineering. 29, 796-810 (2003).
12. Cleland-Huang, J.: Requirements traceability : When and how does it deliver more than it costs? 14th IEEE International Conference Requirements Engineering. pp. 330–330 (2006).
13. Ramesh, B.: Factors influencing requirements traceability practice. (1998).
14. Von Knethen, A., Paech, B., Kiedaisch, F., Houdek, F.: Systematic requirements recycling through abstraction and traceability. Proc. of the Int. Conf. on Requirements Engineering. pp. 273–281 (2002).
15. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering traceability links between code and documentation. IEEE Transactions on Software Engineering. 970–983 (2002).
16. Marcus, A., Maletic, J.I.: Recovering documentation-to-source-code traceability links using latent semantic indexing. Proceedings of the 25th International Conference on Software Engineering. pp. 125–135 (2003).
17. Egyed, A., Grünbacher, P.: Automating requirements traceability: Beyond the record & replay paradigm. Proceedings of the 17th IEEE international conference on Automated software engineering. p. 163--171 (2002).
18. Aiken, P., Arenson, B., Colburn, J.: Microsoft computer dictionary. Microsoft Press (2002).
19. Kirova, V., Kirby, N., Kothari, D., Childress, G.: Effective requirements traceability: Models, tools, and practices. Bell Labs Technical Journal. 12, 143-158 (2008).
20. Narmanli, M.: A business rule approach to requirements traceability, (2010).

21. Ibrahim, S., Idris, N.B., UK, M.M., Deraman, A.: Implementing a document-based requirements traceability: A case study. IASTED International Conference on Software Engineering. pp. 124-131 (2005).
22. Salem, A.M.: Improving software Quality through requirements traceability models. Computer Systems and Applications, 2006. IEEE International Conference on. pp. 1159-1162 (2006).
23. Asuncion, H.U., François, F., Taylor, R.N.: An end-to-end industrial software traceability tool. Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. pp. 115-124. ACM, Dubrovnik, Croatia (2007).
24. Wang, Y.: Software engineering foundations: A software science perspective. AUERBACH (2008).
25. Forward, A.: Software Documentation–Building and Maintaining Artefacts of Communication, (2002).
26. Ambler, S.W., Jeffries, R.: Agile modeling: effective practices for extreme programming and the unified process. Wiley New York (2002).
27. Sulaiman, S., Idris, N.B., Sahibuddin, S.: Production and maintenance of system documentation: what, why, when and how tools should support the practice. (2002).
28. Buckley, J., Mens, T., Zenger, M., Rashid, A., Kniesel, G.: Towards a taxonomy of software change. Journal of Software Maintenance and Evolution. 17, 309–332 (2005).
29. Asuncion, H., Taylor, R.N.: Establishing the Connection Between Software Traceability and Data Provenance. (2007).
30. Azmi, A., Ibrahim, S., Mahrin, M.N.: A software traceability model to support software testing documentation. Proc. 10th IASTED International Conference on Software Engineering. pp. 152-159. IASTED, Innsbruck, Austria (2011).
31. Azmi, A., Ibrahim, S.: Test Management Traceability Model to Support Software Testing Documentation. Communications in Computer and Information Science. pp. 21-32. Springer-Verlag, Dijon, France (2011).
32. da Cruz, J.L., Jino, M., Crespo, A.: PROMETEU-a tool to support documents generation and traceability in the test process. (2003).
33. Valderas, P., Pelechano, V.: Introducing requirements traceability support in model-driven development of web applications. Information and Software Technology. 51, 749-768 (2009).