Formulating a Software Traceability Model for Integrated Test Documentation: a Case Study

Azri Azmi and Suhaimi Ibrahim

***Abstract*— Software Documentation is a key of quality factor in software development but many among developers failed to produce document during their project. The purpose of documentation is to aid the developers in understanding the project. Unfortunately, the documentation is out dated, poor quality, difficult to access and therefore cannot be trusted. The key point of the problems is software traceability. This research is trying to develop a model of software traceability that can generate a software testing documentation to support test management. This will led to new software testing documentation generation process model based on software engineering standards. Information retrieval technique will be utilized to trace link between software artefacts.**

***Index Terms*—Software Documentation, Software Maintenance, Software Testing, Software Traceability.**

I. 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. Boehm [1] reported that cost and risk become higher in software development project with this kind of complexity. It is vital to ensure the reliability and correctness the software being developed. Such aims can be reached using documentation as tools. Documentation is used to embody information such 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 especially when developing large scale systems. As reported by Huang & Tiley [2], there are several shortcomings in current documentation such as out-of-date, inconsistency between source code and documentation, poor quality and others.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. Unfortunately, many organizations failed to implement effective traceability due to difficulties in using and maintaining traceability links [3]. The accurate traceability practices can help in maintaining software. Therefore it will significantly improve the quality of system. On the other hand, neglecting traceability can lead to reduce the quality of the software product. Uusitalo [4] in his survey reported that significant of linking between requirements and testing is to ensure the flow of information about requirements to the testing process. This can overcome the deficiencies of document links.

Manuscript received September 14, 2011; revised September 28, 2011. This research is funded by the UTM-RU grant under the vote no. 01J90. The authors would like to thank UTM-RMC and individuals who are directly and indirectly involved in this project.

Azri Azmi and Suhaimi Ibrahim are with Universiti Teknologi Malaysia, Kuala Lumpur.

II. Related Works

There are many benefits of software traceability. Most commonly, it is claimed to help in change management [5], system verification [3], help in performing impact analysis[6], reuse of software artefacts[7] and meets the need of the stakeholders [3]. Tracing requirement techniques has been used in software development since early 1970s with cross-referring be the first method followed by matrix table. Hence forth, numerous different approaches have been proposed to overcome traceability link issues [8]. These include standard approaches such matrices, databases or repository [9], hypertext links[10], graph-based approaches[11] and many more. There are many approaches available such *information retrieval*, *event based*, *goal centric*, *scenario based*, *rule based* and many more.

A. Information Retrieval (IR)

This approach has been proposed in several research papers [9], [12], [13]. Basically, the technique focuses on automating of retrieval traceability link of document from artefacts such databases. It is done by preparing the collection of artefacts for retrieval through an indexing process. In this process, user requirements are captured by phrases and then are indexed and will be used to rank the artefacts based on similarity of all possible pairs of artefacts. IR approach commonly used model in traceability generation such and Latent Semantic Indexing (LSI), Vector Space Model (VSM), and Probabilistic Model.

There are three general steps in IR approach include (i) pre-processing, (ii) analyzing (iii) analyzing arriving artefacts using some ranking algorithms. IR approach extensively reduces the effort required for creating traceability links between artefacts, but in contrast it still requires significant efforts by the engineers.

B. Event Based (EB)

This approach proposed by Cleland-Huang et al. [5] to provide accurate maintenance of traceability relationships. The traceability links are based on the Observer design pattern of which the links are established by an event but did not use direct links. The traceability links act like as a publisher-subscriber pattern. In this context, dependent

objects such as artefacts should subscribe to the requirement that it depends on. When there are changes to the requirement, an event message is published and all dependent objects will be notified. It is divided into three main components such event server, subscriber manager and requirements manager. Event server task is to manage subscription from dependent objects, while subscriber manager task is to take an action from the event server and manage notification triggered by event server. Meanwhile, requirement manager is responsible for managing all the requirements and publishing the event message to the event server when a change request takes in place.

C. Goal Centric (GC)

Goal Centric traceability established to maintain effectively non-functional requirements (NFR)[14]. This is due to the difficulties of handling NFR compared to functional requirement. NFR describe the requirement that species criteria that can be used to judge the operation of the system rather than specific behaviour while functional requirements describe what the system should do. GC traceability allows the stakeholders to evaluate the impact of changes made by changing the functional requirements to the NFR. GC traceability consists of four phases; (i) goal modelling, where in this phase, NFR are modelled as softgoals; (ii) impact detection, link between functional requirement and NFR are created; (iii) goal analysis, the impact of change is propagated though the related regions and (iv) decision making, where stakeholders analyze the impact introduced by the change.

D. Scenario Based (SB)

Introduced by Egyed [15] , this approach works on how to generate traceability link based on observing test scenarios. These observations are then used for establishing traces between model elements and their corresponding source code. In this approach, scenarios are used to model functionality of system and to generate test cases. SB required a pre-requisite in order to run the traceability. Firstly, it needs to have an existence of an observable and executable software system. Secondly, a responding software model and lastly the scenarios describing test cases. SB reduces the complexity of generating and validating trace information, since it is only required to input the list of observed traces by running test scenarios and a set of hypothesized traces between those scenarios and model elements.

E. Rule Based (RB)

RB approach proposed by Spanoudakis [16] to create a traceability link using rules that supports automatic generation. The generation is based on traceability rules of two types. *Requirement-to-object-model* rules are used in tracing functional requirements and use case specification artefacts to analysis object model. Meanwhile, *inter-requirement-traceability* rules are used in tracing functional requirement and use case specification artefacts to each other. RB supports the automatic generations of traceability relations between different parts of requirement artefacts including requirement statement documents and use case document plus analysis object models. RB approach consists of four steps: (i) grammatical tagging, (ii) conversion of tagged functional requirement into XML, (iii) generation of traceability link between documents, (iv) generation of traceability link between different parts of documents.

III. Evaluation of Software Traceability Approaches

Before a new framework can be proposed, an evaluation of existing approaches needs to be investigated and evaluated. Therefore, the way of assessment needs to be determined. The software evolution taxonomy [17] has been used in order to evaluate the traceability techniques studied together with perspectives discussed by Hazeline[18]. 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 criterion is based on accessibility (mapping between artefacts), granularity, scalability, capturability, formality and tool support. The rational of choosing the criterion is explained in details in next paragraph. (summary of evaluation of traceability approaches depicted in Table 1)

The accessibility criterion evaluates whether the approaches are capable in mapping between artefacts in software development life cycle. Such documents are requirement specification, software design, source code, and test suits. Comparison shows that all the approaches can be link between artefacts up to source code level. In the EB, SB and RB it can trace up to the level of test suits artefacts. Granularity refers to the scale of the artefacts to be changed and can be range from very coarse medium and very fine. For the features of granularity, all approaches are at the level of very fine accept GC. The scalability criterion analyzed whether the approach is capable to be applied to large-scale projects. The evaluation shows that IR and GC are capable to be applied to large system compared to SB and RB only for small to medium system project. Meanwhile, capturability criterion help in analyzed, managed, control, implement or measure software changes. The mechanisms for this are automated, semi-automated or manual. Results of the comparisons shows that the link for approaches such IR, EB and RB can be generated automatically, whereas others semi-automatic.

TABLE I: Traceability Approach Comparison

Features √	IR	EB	GC	SB	RB
Accessibility					
(i) Requirement	√	√	√	√	√
(ii) Design	√	√	√	√	√
(iii) Source Code	√	√	√	√	√
(iv) Test Suits		√		√	√
Granularity					
(i) Coarse					
(ii) Medium			√		
(iii) Very Fine	√	√		√	√
Scalability	√		√		
Capturability					
(i) Automated	√	√			√
(ii) Semi			√	√	
(iii) Manual					
Formality					
(i) Ad-hoc					
(ii) Formal	√	√	√	√	√
Tool Supports	√	√	√	√	√

Formality criterion refers to mechanism that can be implemented either in an ad-hoc way, or based on some underlying mathematical formalism. All approaches need formal methods in implementing the link rather than ad-hoc. To provide a complete and effective traceability solution, tool support is essential, since it allows system developers to use it in real software development contexts. Without tools, any approach is doomed to failure, because in manual traceability schemes it is not easy to maintain the trace information updated and it is very hard to reason and evaluate change impact analysis, coverage analysis, and more. Tool supports criterion evaluates whether the approaches provides tool support to accommodate links between requirements. Aim of tool support is to help in visualizing and managing link. In this case, all approaches can be supported by tools. The result of evaluation is tabulated in Table 1. IR was chosen to be implemented due to its simplicity facilitate dynamic link generation and provides practicable solution of semi-automatic recovering link between code and documentation. The correct traceability use can help a lot of activities within the development process and this will improved the quality of software. On the other hand, ignoring traceability can lead problems in maintaining software in the future.

IV. Result

This research is intended to find 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 [19]. 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. Out of all, there is only one prototype was closely similar to this proposed study. PROMETUE [20] 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. (Details of comparison is tabulated in Table 2)

TABLE 2: Research Approach Comparison

Research	Traceability Representation	Link Between Artefacts						
		R	D	SC	TS	TTF	LS	SH
Zou et al.[21]	Probabilistic Model	√	√					
Lormans et al.[22]	Latent semantic Index (LSI)	√	√	√	√			
Duan and Cleland Huang [23]	Probabilistic Model	√	√	√				
Antoniol et al.[24]	Vector Space Model	√	√	√	√	√		
Da Cruz et al.[20]	Horizontal Traceability (IEEE 829-1998 based)		√		√			
Proposed Research	Yes (IEEE 829-2008 +MIL498 + ISO)	√	√	√	√	√	√	√

R=Requirement D=Design SC=Source Code TS=Test Suit TTF=Testing Tools Files LS=Legacy System SH=Stake Holder

V. Discussion

Fig. 1 shows the proposed framework of software testing documentation process based on Software Engineering Standards. There are five main components namely *Extractors/Parsers*, *Traceability Engine, Analyzer, Traceability Repository* and *Document Generator*. The proposed framework illustrates that all the data are gathered and stored in a repository. Firstly, the tool will analyse 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, in different notations, managed by different software tools such as: (i) source code (Java and C++), (ii) software documents such Software Development Plan (SDP), Interface Requirements Specification (IRS), Software Requirements Specification (SRS), Sofware Design Specification (SDD), Software Test Result (STR) and Sofware Test Desriptions (STD), (iii) legacy systems, (iv) stakeholders/users, (v) output files from testing tools (including bug-tracking systems), (vi) requirements and (vii) experts. All the information includes function and non-functional requirement.

Extractor/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. XML is an open format and self-describing way of encoding both text and data, and can be changed across variety of platforms, languages and applications. XML can be used with wide range of development tools and utilities. 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. Next subchapters discussed details of the proposed model components.

Extractor job is to analyze continues flow of input and breaks into constituent parts. Several parsers will be used to

convert multiples format of input data into XML. It will support the capture, summarization and linking of software artefacts information. This support for extracting information from wide variety source artefacts, viewing it in summarization form, manages changes to the artefacts in different representations.

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.

The input data for traceability engine will be a traceability repositories. These repositories vary in their usage, content and storage format. 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.

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.

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.

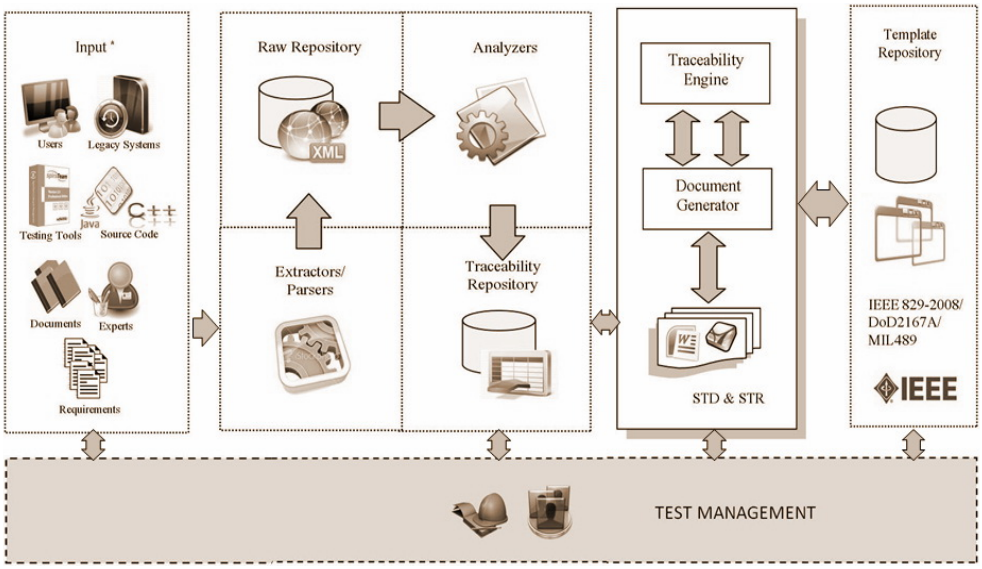

Fig. 1. Proposed model

VI. Case Study: OBA

OnBoard Automobile System (OBA) will be used as a case study for this research. OBA is final project for masters student at Advanced Informatics School (UTM AIS), Univeristi Teknologi Malaysia, Kuala Lumpur. OBA mission is to improve the safety of vehichle driving, in particular for long trip journey. It is developed using combination of C++ and Java, and estimated about 3K line of code (LOC) and based on DOD2167A and MIL498 standards with full set of artefacts such SDP, IRS, SRS, SDD, STD and STR. OBA consist of several component such cruise control, car maintenance, calculate average speed, calibration and calculate fuel fill between two stops. Fig 2 illustrate the autocruise state mode where autocruise can be activated by pressing the activation button. In the autocruise state, the system can be either in mode of accelerating, regulating and suspended. Autocruise cannot be activated while the system is in the calibration mode.

The documents provided us with some useful information on traceability between artefacts. OBA project begin with requirements document and some letters to kick off the project. Fig 3 shows an example of letter from stakeholders where traceability start taking place in the project. Meanwhile Fig.4 illustrate part of requirements document.

Requirements document is an artefact that consists of functional requirements from a stakeholder. Letters, requirements document, emails, an expert opinion are gathered and analyzed to form an IRS and SRS document. The resourses and time to produce IRS and SRS are stricly based on the SDP. There will be a review after each miliestone completed. After the IRS and SRS completed and reviewed, SDD take place and so fourth until bugs reporting is performed. Each documents has it own traceability table/matrix and its refer only to the document which is one level above it. The issue is, it cannot refer to more than one level above or below the document. Fig 5 decpicted the situation. Source code artefact cannot refer to the SRS directly, but should refer to traceability matrix in SDD first. This will be difficult to make a referral where necessary without first refer the other documents. This will led to error prone and time consuming. This research will try to automate

the traceability link between artefacts for the whole of software development life cycle.

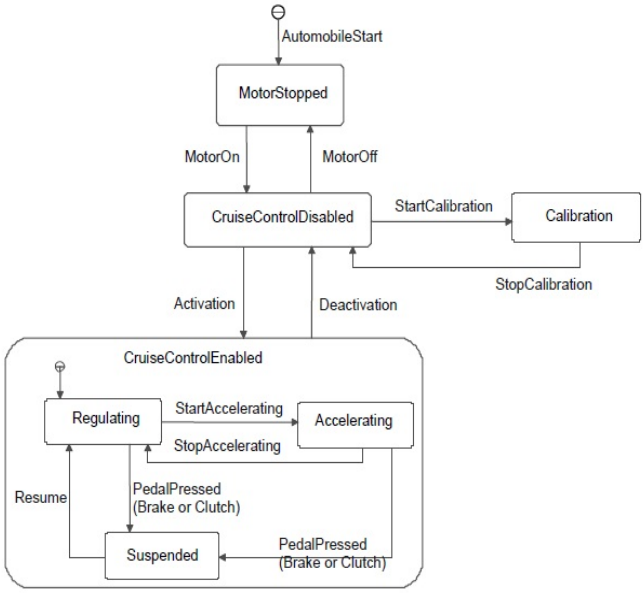

Fig. 2. Cruise control state

Kuala Lumpur January 24th, 2008

Dear Sirs,

Further to the "Driving Assistance System" under study by the road safety department, note that the project can be accepted only with the following modifications (those concerning the "On-Board Automobile Computer" software):

1. Just as for pressing on the break pedal, pressing on the clutch pedal should cause the deactivation of the on-board computer. The driver must press on "Resume" and the transmission must be on the highest speed, and the brake and clutch pedals released, so that the system returns the vehicle to the previously selected speed.
2. The driver must be able to ensure, by simply consulting visually a specific indicator on the instrument panel, whether the on-board computer is operating in automatic driving mode.
3. Significant over speeding in relation to the cruising speed under automatic

Fig. 3. Letter from client

2.1 Control of cruising speed
The function consists in taking over for the driver in order to maintain a constant speed when requested by the driver. This function can be activated, deactivated, suspended or resumed. The driver shall be able to define and modify the cruising speed.

The suspension of this function (temporary stopping of vehicle speed management) shall be instantaneous in the event of danger requiring the resumption of the speed control by the driver. In this case, it shall not require any action by control panel, but be associated with the control devices (pedals, etc.).

2.2 Maintenance scheduling assistance
The function consists in indicating to the driver when maintenance is necessary (oil change, oil filter, air filter, general check) according to a given "schedule", expressed in terms of distance.

2.3 Average speed calculation
The function consists in providing the driver with an indication of the average speed upon a trip.

Fig. 4. Requirements document

To overcome this predicament, all the artefacts should be kept in one repository. As discussed in previous section (Fig 1), all the aftefact will be gathered as an input to the systems. An extractor will be used to extract all the information to keep in repository. In general the data can be interpreted in the following point of view:

$$db = a_1 \cup a_2 \cup a_3 \cup \ldots \cup a_n$$
$$t = (a, r)$$

where, *db=database/repository a=artefact r=relation t=traceability*

We believe that there exist some relationships among software artefacts. Therefore, we need to capture and trace their link not only with same level of artefacts heirarchy but with different level of phases. Multi-bidirectional traceability link will be impemented in this OBA project. The process of tracing and capturing these artefacts is called hypothesizing traces [25] as dicpited in Fig 6. Fig 7 represent an example of mapping one requirement (Cruise Control) in OBA project using multi-bidirectional traceability approach. The thin arrows indicate the normal bidirectional traceability while thick dashed shows relationship using multi-bidirectional approach. Previously, if developer or tester need to refer traceability table in STD, he manage to see links up to one level only. By automated the traceability link, he now can refer to any direction of traceability and to any documents.

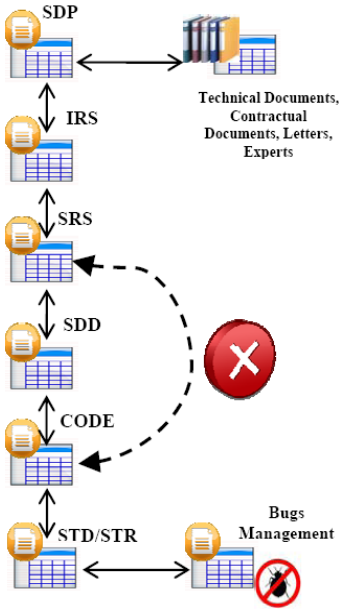

Fig. 5. Traceability hierarchy (bidirectional relationship)

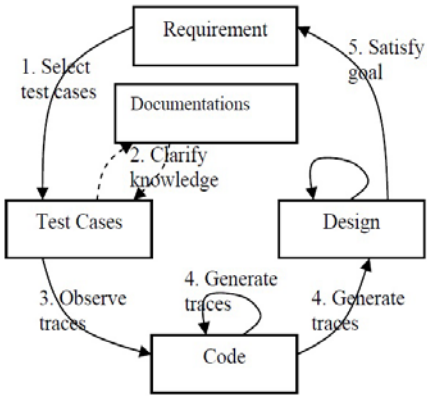

Fig. 6. Hypothesized and observed trace

As example for *cruise control* requirement, the requirement comes from multiple sources such requirement documents itself, email, letters from stakeholders and an expert opinion. After the analysis is made, then IRS and SRS forms. Cruise control was given a referral number. In this case, it was given as SRS_REQ-03-00. The reference number will be stored in traceability matrix table and will be used to trace its origin sources. Same with the other of document, which will each be given a unique reference number. In SDD, the referral number for cruise control is SDD_REQ-04-00 while in STD, with STD_REQ-02. Now each of the document can be linked to another without the level of document as a contraints.

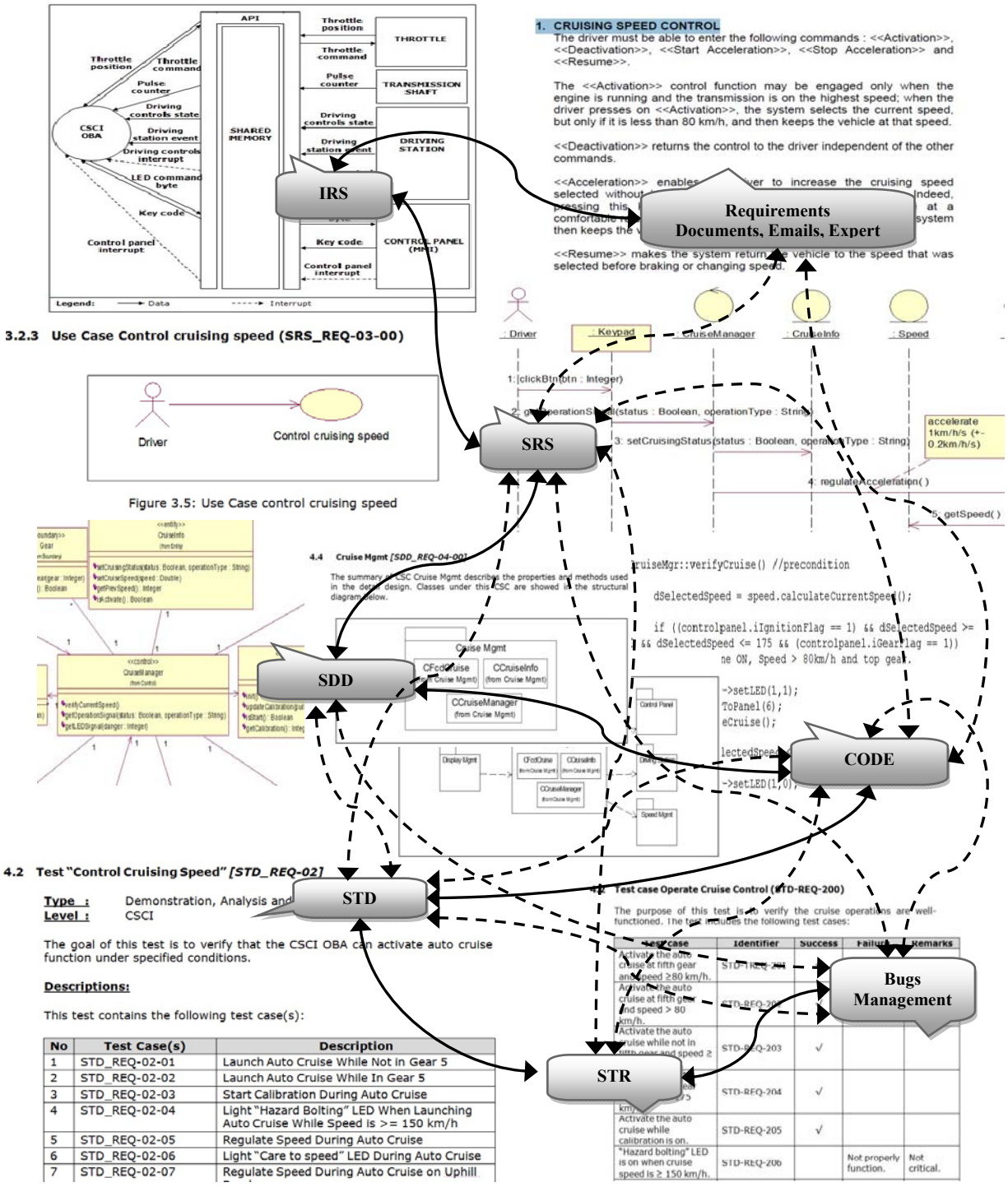

Fig. 7. Multi-bidirectional traceability for cruise control requirements

VII. Conclusion

This research is at the beginning of implementation phase. Our approach of traceability is using multi-bidirectional relationship between software artefact. At the end, the system will generate software testing documents based on traceability produced. A key point here is to establish a workable traceability process and approach to meet the demand of software documentation. 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 new approach and process model is expected to support software testing documentation that will certainly be useful in software maintenance activities.

References

[1] B. Boehm, "Value-based software engineering: reinventing," *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 2, p. 3, 2003.
[2] S. Huang and S. Tilley, "Towards a documentation maturity model," in *Proceedings of the 21st annual international conference on Documentation*, San Francisco, CA, USA, 2003, pp. 93-99.
[3] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 58–93, 2001.
[4] E. J. Uusitalo, M. Komssi, M. Kauppinen, and A. M. Davis, "Linking requirements and testing in practice," *16th IEEE International Requirements Engineering, 2008. RE'08*, pp. 265–270, 2008.

[5] C. K. Chang and M. Christensen, "Event-based traceability for managing evolutionary change," *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 796-810, 2003.
[6] S. Ibrahim, N. B. Idris, M. M. UK, and A. Deraman, "Implementing a document-based requirements traceability: A case study," in *IASTED International Conference on Software Engineering*, 2005, pp. 124-131.
[7] A. Von Knethen, B. Paech, F. Kiedaisch, and F. Houdek, "Systematic requirements recycling through abstraction and traceability," in *Proc. of the Int. Conf. on Requirements Engineering*, 2002, pp. 273–281.
[8] G. Spanoudakis and A. Zisman, "Software Traceability: A Roadmap," *Handbook of Software Engineering and Knowledge Engineering*, 2005.
[9] P. S. Kritzinger and H. Krüger, "Software Traceability using Latent Semantic Analysis and Relevance Feedback," *Technical Report CS08-01-00, Department of Computer Science, University of Cape Town.*, pp. 391-402, 2008.
[10] I. Alexander, "Towards automatic traceability in industrial practice," in *Proc. of the 1st Int. Workshop on Traceability*, 2002, pp. 26–31.
[11] H. Schwarz, J. Ebert, and A. Winter, "Graph-based Traceability: A Comprehensive Approach," 2009, pp. 1–20.
[12] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia, "Information retrieval models for recovering traceability links between code and documentation," in *Software Maintenance, 2000. Proceedings. International Conference on*, 2000, pp. 40-49.
[13] C. McMillan, D. Poshyvanyk, and M. Revelle, "Combining textual and structural analysis of software artifacts for traceability link recovery," in *Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, 2009, pp. 41–48.
[14] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhanskaya, and S. Christina, "Goal-centric traceability for managing non-functional requirements," in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 362-371.
[15] A. Egyed, "A Scenario-Driven Approach to Traceability," *Proceedings of the 23rd international conference on Software engineering*, pp. 123–132, 2001.
[16] G. Spanoudakis, A. Zisman, E. Pérez-Miñana, and P. Krause, "Rule-based generation of requirements traceability relations," *Journal of Systems and Software*, vol. 72, no. 2, pp. 105-127, Jul. 2004.
[17] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel, "Towards a taxonomy of software change," *Journal of Software Maintenance and Evolution*, vol. 17, no. 5, pp. 309–332, 2005.
[18] H. Asuncion and R. N. Taylor, "Establishing the Connection Between Software Traceability and Data Provenance," 2007.
[19] H. U. Asuncion, F. François, and R. N. Taylor, "An end-to-end industrial software traceability tool," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, Dubrovnik, Croatia, 2007, pp. 115-124.
[20] J. L. da Cruz, M. Jino, and A. Crespo, "PROMETEU-a tool to support documents generation and traceability in the test process.," 2003.
[21] X. Zou, R. Settimi, and J. Cleland-Huang, "Term-based enhancement factors for improving automated requirement trace retrieval," in *Proceedings of International Symposium on Grand Challenges in Traceability*, pp. 40–45.
[22] M. Lormans and A. van Deursen, "Can LSI help reconstructing requirements traceability in design and test?," in *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, 2006, p. 10 pp.-56.
[23] C. Duan and J. Cleland-Huang, "Clustering support for automated tracing," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, 2007, pp. 244–253.
[24] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, pp. 970–983, 2002.
[25] S. Ibrahim, N. B. Idris, M. Munro, and A. Deraman, "A requirements traceability to support change impact analysis," *Asian Journal of Information Tech*, vol. 4, no. 4, pp. 345–355, 2005.

Azri Azmi is currently a PhD candidate at Universiti Teknologi Malaysia, Kuala Lumpur. He received Diploma in Computer Science (1994), Bachelor Degree in Computer Science (1996) and Master of Science (Computer Science – Realtime Software Engineering) (1998). He is an ISTQB certified tester. He is member of IEEE and ACM (Association for Computing Machinery)

Suhaimi Ibrahim received the Bachelor in Computer Science (1986), Master in Computer Science (1990), and PhD in Computer Science (2006). He is an Associate Professor attached to Dept. of Software Engineering, Advanced Informatics School (AIS), Universiti Teknologi Malaysia International Campus, Kuala Lumpur. He currently holds the post of Deputy Dean of AIS. He is an ISTQB certified tester and being appointed a board member of the Malaysian Software Testing Board (MSTB). He has published many articles in international conferences and international journals such as the International Journal of Web Services Practices, Journal of Computer Science, International Journal of Computational Science, Journal of Systems and Software, and Journal of Information and Software Technology. His research interests include software testing, requirements engineering, Web services, software process improvement, mobile and trusted computing.