2528-9705

Örgütsel Davranış Araştırmaları Dergisi
Journal Of Organizational Behavior Research
Cilt / Vol.: 8, Sayı / Is.: S, Yıl/Year: 2023, Kod/ID: 23S0-1014

AN INTERACTIVE TRACING SOFTWARE REQUIREMENT AND RESOURCES TOOL TO REDUCE COST AND REDUNDANCY

Marziyeh ANDISHEH[1*]

[1]*Universiti Putra Malaysia, Serdang, 43400 Selangor, Malaysia.

*Corresponding Author
E-mail: faribaandisheh@gmail.com

ABSTRACT

Planning for Software development has been one of the challenging phases of Software Development Lifecycle (SDLC). Proper planning for software development can assure the quality of software product. Traditional SDLCs where development phase starts upon completion of order and setting deliverables by customers have replaced by procedural and iterative that delivery is on module-basis. These strategies are common as they are budget-saving, require less time to deliver and do not require having all the requirements ready at the beginning. In traditional way, planning and tracing software requirement took place by seniors and experts in product feasibility planning which was costly as well. Alternatively, in new strategies, planning is based on resource sharing and tracing among different projects by non-experts to reduce costs. However, available platforms in the market are having complicated menus and not interactive enough (mainly text-basis) to regular IT staffs to plan and trace for their projects requirements and resources and simply find the impactful ones.

Keywords:. *requirements traceability, interactive, cost, and redundancy*

INTRODUCTION

THROUGHOUT the software lifecycle, there are many decision situations involving limited resources(Boehm 1984). Therefore, planning for software development requires analysis based on systems software engineering. It is also requires condition and needs determination to ensure that the final product (software prototype) meet objectives and requested deliverables by customers. In the IT industry, the level of success in a software development project is highly depending on how well it has been planned. It is also necessary to properly plan for software's project requirements and resources considering project's limited budget.

Software requirement planning has been considered as one of the most critical phases in Software Development Life Cycle (SDLC). A proper planning for requirements of a software development project, not only can facilitate developers and assist them to meet the deadlines, but also can reduce project costs. It is also possible to avoid requirement request conflicts using proper planning and facilitate to reuse the requirements available in inventory or those which are engaged in other phases of project. It is possible to reuse requirements that they have used for projects with partially similar requirements to reduce the cost. However, requirement planning has been always a challenging task as currently available tools have complicated User Interfaces (UIs). They are not interactive enough to trace the requirements in a simple way and

find their impacts on the overall project completion in an easy way so a regular user can simply use it.

Tracing software requirement considers as one of the possible solutions to efficiently reusing software requirements and allocating resources. This area includes requirement management within software engineering. It includes documents showing the history of every individual of the resources and their sources. The term "Traceability" concerns with the percentage which is showing ability of tracing a particular resource(Ramesh and Jarke 2001). In other words, if a resource is highly traceable means, it would be possible to access and check the availability of that particular resource. Tracing tools are usually able to answer such questions: where requirements are derived from, how they are satisfied, how they are tested, and what impact will result if they are changed.

Requirement elicitation or requirement gathering is a practice to collect different requirements from users, customers and stakeholders (Sommerville and Sawyer 1997). However, it is not possible to only rely on the requirements asked to these three groups by asking them. By using requirement traceability, it is possible to trace back requirement from different people and through observation or questionnaire. It is also possible to prioritize the requirement using designed tools for this purpose and check the reasons that has required in first place.

As mentioned above, requirement traceability involves with relationships between different requirements and documenting the history of resources. It can overall enhance the quality of products and it is able to manage the changes. It is necessary to trace the relationships of requirements in addition to tracing requirements individually. It should also be possible to trace users and groups related to every particular requirement.

Interaction with computers can be done through a UI. Traditional UIs were text-basis where recent UIs are mainly designed on visual-basis. They are using graphics, charts and figures to show information and they may provide feedback if a user aims to interact with them. Interactive UIs are easier to learn and more user-friendly. They may give a non-expert user to understand and effectively use with the software. The remainder of this paper is organized as follows: Section II presents the background on some related papers about the requirement traceability and some developed traceability tools. Section III describes the program in which the method was applied. Section IV shows the design of system. In section V you will see the results and some insights. Finally, we draw our conclusion and future work in Section VI.

RELATED WORKS

Papers

There are many researches related to software requirement traceability. In these researches algorithms proposed to trace software requirements. For instance Gotel et.al (Gotel and Finkelstein 1994) proposed to define pre-requirements specification (pre-RS) traceability and post-requirements specification (post-RS) traceability to have better evaluation and track of changes. They proposed a system with two tracing engine.

In another research, Asuncion (Asuncion, Asuncion et al. 2010) proposed and designed a tool with topics modelling feature. In their system, topics can be grouped based on their keywords

and the system could enhance the requirement tracing strategy. Asuncion et.al (Asuncion, François et al. 2007) designed an end-to-end tracing software by focusing on both requirements traceability and process traceability.

Based on Winkler et al. (2010) software requirement tracing can be studied from five aspect including : Trace integration, Trace recording, Trace maintenance(Chen and Chou 1999), Economic aspects(Egyed 2006)and Human aspects(Alexander 2002). Researches such as (Maletic, Collard et al. 2005) were focused on tool integration where other researches (Pohl 1996, Pinheiro 1997, Mohan and Ramesh 2002, Munson and Nguyen 2005) were more on specialized tools (Process-centered, Object-oriented, Rationale-centered and Code-centered).

In regards to trace recording, researches can be categorized into 4 aspects which are Structural Rule-based (e.g. (Richardson and Green 2004, Egyed and Grünbacher 2005)), Linguistic Rule-based (e.g. (Kaindl, Kramer et al. 1999, Grechanik, McKinley et al. 2007)), Based on information-retrieval (e.g. (Antoniol, Canfora et al. 2002, och Dag, Regnell et al. 2002)) and Fully automated (e.g. (Whittle, Van Baalen et al. 2001, Gervasi and Zowghi 2005). Although, there are available solutions and software in the current market such as Doors(Hayes, Dekhtyar et al. 2006), Rational Requisite (Nawrocki, Jasiński et al. 2002)and CaliberRM(Wiegers 1999), they are lacking in interactivity of user interface evidences by researches such as (Garcia and Paiva 2016) and (Shah and Patel 2014). Garcia (2016) believed that low level of interactivity and simplicity in software traceability apps can be simply found by looking at their complex menus and designs. He believed that an interactive interface would enhance the simplicity and give the chance to non-professional software developers to trace software requirements as well.

Tools

There were also some available solutions and software in the current market that will be reviewed to explore the potential gap that this project aims to fill in. Some these products have reviewed below:

DOORS

Rational Dynamic Object Oriented Requirements System (DOORS) is a software requirement tracing, planning and management tool developed in early 1990s. Its front end can only run on windows and its back-end runs only on Linux. It works with a programming language that has specifically designed for this platform. It was initially designed in the ministry of defense in UK on 1991 and its first commercial version of that is released on 1993. DOORS is capable define links between requirements and trace them during the project(Zisman, Spanoudakis et al. 2003). Although DOORS's designed based on a tabular view, it is yet not much interactive and simple to understand. The traceability links for the objects (according to defined requirements) can be created.

Rational Requisite

It is software which is firstly released on 1985 based on Ada programming language. The goal of designing this system is to increase the productivity and can be run under different platform. This software can facilitate requirements tracing and supports different types of compilation. The architecture of this software is optimized and provides independent 64 bit data channels.

This is designed mainly to support iterative development. It can also support requirement management and can trace them based on experience.

CaliberRM

It is a requirement management system that is initially designed to assure that the software product meets the deliverables set by clients(Lu, Chang et al. 2008). It can enable managers and developers to increase software quality. It is also capable to provide communication and collaboration between system components. Furthermore, this system has additional features to what has provided by its competitors in the market including repository control and planning for requirement using different types of SDLC. It is also capable to trace and perform impact analysis on software requirements. Although this web-based software is designed to be visual and able to generate visual statistical reports on requirements.

Cradle

Cradle is a tool that can manage software needs and requirements. The advantage of this software over its alternatives available in the market is its flexibility to users where they can define the functionalities on their own(De Gea, Nicolás et al. 2012). It can also perform analysis on users' needs and load information from external resources. It is fully supporting traceability across the entire lifecycle. This tool can support resource and requirement sharing between projects. However, it has complex menus requires trainings.

iRise

Is a software to trace the requirement which provides a cloud-based features(Schwaber, Leganza et al. 2006). It is initially founded on 1996. It has drag and drop features layouts for different scenarios. In this system users may create custom User Interface elements and components. Using iRise users are able to work on the same project simultaneously, collaborate and add comments. They are able to trace software requirements and see their impact on time-basis.

MATERIALS AND METHODS

This research aims to answer the research questions through a prototype and based on experiments' results. It will use quantitative method to collect the results and analyze those using numbers.

The prototype will be designed using Rapid Application Development (RAD) because:

RAD is iterative so it is a suitable choice for module-based software development.

RAD can give us the opportunity to make the necessary changes in design (if required) within the development phase. This research contains 5 phases as shown in **Figure 1** that would be done based on time plan as shown below:

Figure1. Project phases

To follow RAD cycle for designing and developing the proposed prototype, we will firstly plan and set our requirements such as what is necessary for software development, survey questionnaires and our target group of participants. Design Instrument:

As mentioned, we may change the design within the construction (development) phase benefiting from RAD advantages. Each module of the prototype will be designed separately and an integrated version will be released by combining modules together.

Data collection:

Data collection for this study will be quantitative (using questionnaire). The data collection part would be done using based on a questionnaire.

Data Analysis:

An analysis will be conducted on the data collected and the results presented using texts and visual elements (graphic and charts) where necessary. This phase will be done iteratively according to RAD cycle. The results will be statistically evaluated to show the research output.

The results of test for different cases of software development will be studied under two groups (demography) of participants:

1- Expert software requirement planners

2- Regular software developer

In order to narrow down the testing process we would focus more on the population sampling including people who are already in the IT industry. Due to the open nature of this study, the selection of respondents will not be based on any specific age or gender. The study will focus more on experts who were using traditional requirement tracing software and comparison to the proposed solution.

Result presentation:

Finally, we will combine and present the results and evaluate how they can support and answer the research questions.

System design

For detailed management, the resources of a project, as mentioned before, are as followed:

Time

Human Resources (Skills)

Computer Hardware Resources

Software Resources

Budget

The requirements of the project determine the time and required skills. The skills rely in the people, the project team members. On the other hand, the experience of the team members can affect the time. The team members require software to develop the project. And likewise, the

software need hardware resources to be executed. These last three resources require budget. This project should be able to trace and manage all of the resources and their relations to each other. Their inter-relations are show in **Figure 2**.

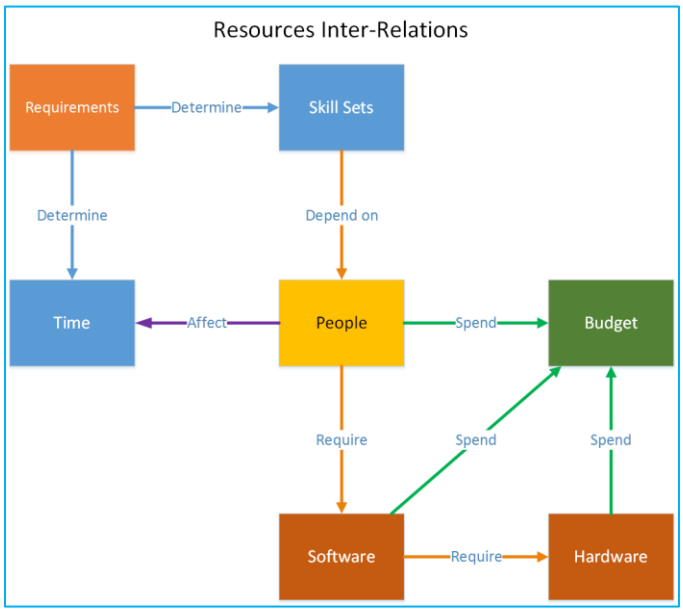

Figure2. inter-relations of project resources.

For keeping the track of the team members, software and hardware resources a tracing system is required on the workstations. This tracing system should record the software and hardware resources, which in return, shows how the team members are using the resources to finish the project.

The overall architecture of the application being developed for this project is presented in Error! R eference source not found..

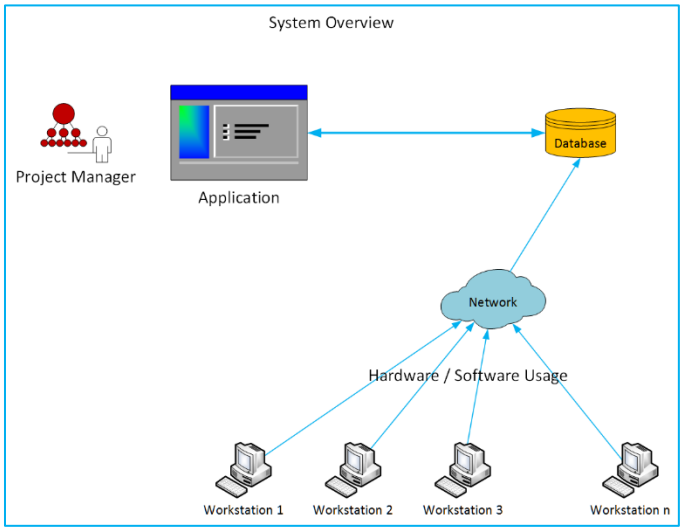

Figure 3. The overview of the system architecture

The system components are as followed:

- Main Application for managing and tracing the resources.
- A tracing application to be installed on the resource computers.
- The central database for storing and retrieving data.

Business Rules

- The user of the program should be able to define a project in the application.
- The team members' data like their names, skills, velocity of the skills and daily salary should be entered.
- Their salary can determine a percentage of the required budget of the project.
- The hardware for developing the project, like workstations or other types of needed hardware should be listed and their price should be entered in the application. Some of the project budget will be spent on the hardware which should be calculated precisely.
- The software that the team uses to develop the project that is available on each workstation should be entered. Another percentage of the budget spends here.
- The requirements of the project and the needed time and skills for implementing them should be entered separately.
- The people should be assigned to the requirements. This depends on the project manager preferences and each team member skills.
- The velocity of implementing the requirement is important, because it determines the required time. For instance, if 100 skill velocity-time are needed, it means 20 hours of work times 5 velocity of developing (100 skill velocity-time = 20 hours * 5 velocity). Using this formula, the higher the velocity, the less time required for finishing the requirement.
- The status of the requirements should be known any time during developing the project. They should have 3 stages: To do, In Progress, Done. After done, the actual time is calculated.
- The application should be able to monitor how the team members use their time and resources for implementing the project. Thus, a Windows Service application should be developing for sending the hardware status and list of running software on each workstation every 5 minutes.
- Any changes to the resources and their effect on other resources should be shown in the application.

- The reports of status and resulting calculations should be appealingly presented to the user.

Defining a Project

There are two aspects to be considered while defining the project. The time is represented in **Figure4**.

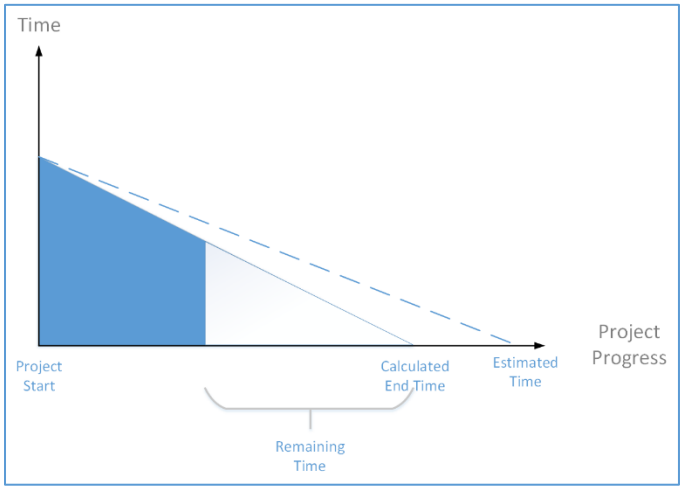

Figure 4. Project time management.

The different aspects of budget of the project is shown in **Figure 5**.

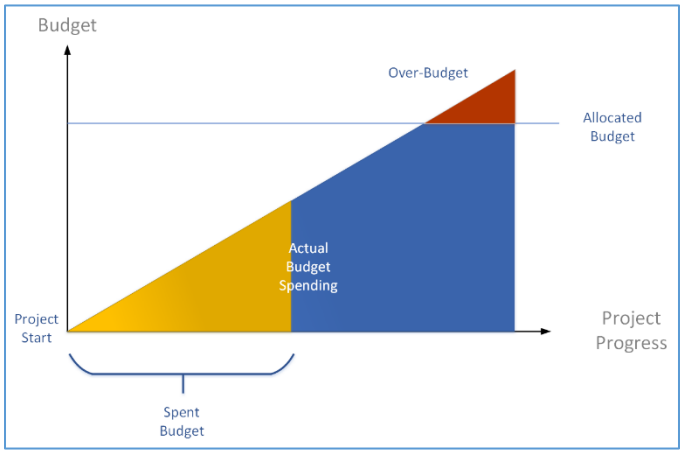

Figure 5. Project budget management.

1. Define project:
    a. Add name and description,
    b. Add allocated budget,

c. Add start time.

2. Define available people and their skills:
   a. Add project members' info,
   b. Add their skills and their level of experience in the skill (the more experience, the more agile they can implement the requirement),
   c. Add their daily salary.
3. Define available hardware and software:
   a. Add each available workstation,
   b. Add the available software packages on each workstation,
   c. Specify budget required for utilizing them.
4. Define project requirements:
   d. Define each requirement,
   e. Specify estimated time and skill experience for the requirement.
   f. Assign the requirement to the project members,
   g. Assign needed hardware and software to the member,

5. Calculate the overall budget and time:
   a. Budget based on recurring and one-time costs,
   b. Time based on the calculated time of each requirement.

Each requirement is divided to a few tasks. We keep track of each task in the project. This is shown in **Figure 6**.

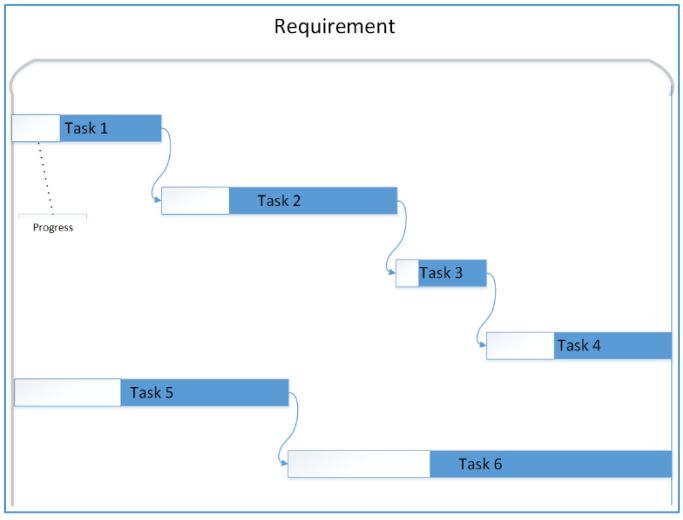

Figure 6. A Requirement and its tasks

RESULTS AND DISCUSSION

After collecting the answers from the participants, the results are as followed:

Average of question 1: 7.8

Average of question 2: 7.6

Average of question 3: 6.8

Average of question 4: 6.2

Average of question 5: 8.2

Average of question 6: 7.8

Average of question 7: 6

Average of question 8: 8

Average of question 9: 5.2

And the total average of 9 questions is **7.07** that shows is above the average value and is considerable. It should be considered that question 10 is open question and filled by the respondent's comments and based on these comments the limitation of the system can be found.

Products Comparison

Table 1. Product comparison

Product	Trace	Cloud-based	Real-Time	Collaboration	Resource Sharing	Monitoring Resources
DOORS	✓	×	✓	×	×	×
Rational Requisite	✓	×	✓	×	×	✓
CaliberRM	✓	×	✓	×	×	×
Cradle	✓	×	✓	×	✓	×
iRise	✓	✓	✓	✓	✓	✓
This Project	✓	×	✓	×	✓	✓

This product has the following features and abilities:

- Adding projects, requirements and tasks.
- Adding all resources including budget, estimated time, members, hardware and software.
- Monitoring the people and their usage of hardware and software.
- Real-time estimate of remaining time and budget.
- Resources sharing between the people working on the project.

Based on what has set as this research aims and objectives, it will be expected to get the results as below:

1. A software prototype with interactive features and UI. This software will be easier to understand and more learnable for users in compare to similar available software products in the market.
2. An evaluation and estimation on software development cost reduction using the designed prototype. In another word, the impact of using this software prototype on cost reduction for companies will show how impactful it can be in a tangible way.
3. The designing tool will avoid redundancy in software requirement planning. As a result, we will not see any redundancy as an outcome of using this software prototype.

CONCLUSION

In summary of what has mentioned in this proposal, this research aims to design an interactive software requirement tracing tool and investigate its benefit on companies cost reduction and redundancy avoidance. It will investigate the outcome through a set of questions to be answered by experts in requirement planning and regular software developers who have used non-interactive available requirement tracing tools available in the market comparing to what has developed as a prototype in this project.

Base on the objectives of this project, the resulting software should be able to trace the project requirements and its resources. As mentioned before, the resources of a project are: Time, Budget, Human Resources, Hardware Resources, and Software Resources. It is supposed that this project should manage all of these resources and their effect on each other. It is important to say after collecting the answers from the participants, the total average of 9 questions is 7.07 that shows is above the average value and is considerable. Although this software fulfill the mentioned objectives such as tracing the requirements and resources, resource sharing, monitoring resources and also is real time. The main target of this project was about user friendly that any people with low knowledge can also work with it, where is clear based on gained results in part V. The result software of this project is represented to a small team of the IT experts. The overall feedbacks are toward the positive and more satisfying values.

Advantages

The strongest feedbacks are on ease of use of the software, fulfillment of the purpose of the software, and the look and feel of the application.

Limitations

The weakest feedbacks are one the collaboration of other users with the software, the account setup experience, and integration with other applications.

In future work, based on comments mentioned in the questioner part, defining more than one user as a project manager in the application and customizable timing of the work stations in Software Information Collector can be considered.

ACKNOWLEDGMENTS: So many thanks to my family especially my father Abdolrahim Andisheh that supported me every moment of my life and my mother Effat Rahmani with her Compassionate heart , I truly don't know what I would have done without you

CONFLICT OF INTEREST: None

FINANCIAL SUPPORT: None

ETHICS STATEMENT: None

References

Alexander, I. (2002). Towards automatic traceability in industrial practice. Proc. of the 1st Int. Workshop on Traceability, Citeseer.

Antoniol, G., G. Canfora, G. Casazza, A. De Lucia and E. Merlo (2002). "Recovering traceability links between code and documentation." IEEE transactions on software engineering**28**(10): 970-983.

Asuncion, H. U., A. U. Asuncion and R. N. Taylor (2010). Software traceability with topic modeling. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, ACM.

Asuncion, H. U., F. François and R. N. Taylor (2007). 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, ACM.

Bakhitar, A., A. Hannan, A. Basit and J. Ahmad (2015). „Prioritization of value based services of software by using Ahp and fuzzy KANO model.". International Conference on Computational and Social Sciences.

Boehm, B. W. (1984). "Verifying and validating software requirements and design specifications." IEEE software**1**(1): 75.

Brown, R. B., G. Beydoun, G. Low, W. Tibben, R. Zamani, F. García-Sánchez and R. Martinez-Bejar (2016). "Computationally efficient ontology selection in software requirement planning." Information Systems Frontiers**18**(2): 349-358.

Chen, J.-Y. and S.-C. Chou (1999). "Consistency management in a process environment." Journal of Systems and Software**47**(2): 105-110.

De Gea, J. M. C., J. Nicolás, J. L. F. Alemán, A. Toval, C. Ebert and A. Vizcaíno (2012). "Requirements engineering tools: Capabilities, survey and assessment." Information and Software Technology**54**(10): 1142-1157.

Egyed, A. (2006). Tailoring software traceability to value-based needs. Value-Based Software Engineering, Springer**:** 287-308.

Egyed, A. and P. Grünbacher (2005). "Supporting software understanding with automated requirements traceability." International Journal of Software Engineering and Knowledge Engineering**15**(05): 783-810.

Garcia, J. E. and A. C. Paiva (2016). "A Requirements-to-Implementation Mapping Tool for Requirements Traceability." Journal of Software**11**: 193-200.

Gervasi, V. and D. Zowghi (2005). "Reasoning about inconsistencies in natural language requirements." ACM Transactions on Software Engineering and Methodology (TOSEM)**14**(3): 277-330.

Gotel, O. C. and C. Finkelstein (1994). An analysis of the requirements traceability problem. Requirements Engineering, 1994., Proceedings of the First International Conference on, IEEE.

Grechanik, M., K. S. McKinley and D. E. Perry (2007). Recovering and using use-case-diagram-to-source-code traceability links. Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ACM.

Hayes, J. H., A. Dekhtyar and S. K. Sundaram (2006). "Advancing candidate link generation for requirements tracing: The study of methods." IEEE Transactions on Software Engineering**32**(1): 4-19.

Kaindl, H., S. Kramer and P. S. N. Diallo (1999). Semiautomatic generation of glossary links: A practical solution. Proceedings of the tenth ACM Conference on Hypertext and hypermedia: returning to our diverse roots: returning to our diverse roots, ACM.

Lu, C.-W., C.-H. Chang, W. C. Chu, Y.-W. Cheng and H.-C. Chang (2008). A requirement tool to support model-based requirement engineering. 2008 32nd Annual IEEE International Computer Software and Applications Conference, IEEE.

Maletic, J. I., M. L. Collard and B. Simoes (2005). An XML based approach to support the evolution of model-to-model traceability links. Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering, ACM.

Mohan, K. and B. Ramesh (2002). Managing variability with traceability in product and service families. System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on, IEEE.

Munson, E. V. and T. N. Nguyen (2005). Concordance, conformance, versions, and traceability. Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering, ACM.

Nawrocki, J., M. Jasiñski, B. Walter and A. Wojciechowski (2002). Extreme programming modified: embrace requirements engineering practices. Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on, IEEE.

och Dag, J. N., B. Regnell, P. Carlshamre, M. Andersson and J. Karlsson (2002). "A feasibility study of automated natural language requirements analysis in market-driven development." Requirements Engineering**7**(1): 20-33.

Pinheiro, F. d. A. C. (1997). Design of a hyper-environment for tracing object-oriented requirements, University of Oxford.

Pohl, K. (1996). PRO-ART: Enabling requirements pre-traceability. Requirements Engineering, 1996., Proceedings of the Second International Conference on, IEEE.

Ramesh, B. and M. Jarke (2001). "Toward reference models for requirements traceability." IEEE transactions on software engineering**27**(1): 58-93.

Richardson, J. and J. Green (2004). Automating traceability for generated software artifacts. Proceedings of the 19th IEEE international conference on Automated software engineering, IEEE Computer Society.

Schwaber, C., G. Leganza and M. Daniels (2006). "The Root of the Problem: Poor Requirements." IT View Research Document, Forrester Research, September.

Shah, T. and S. Patel (2014). "A Review of Requirement Engineering Issues and Challenges in Various Software Development Methods." International Journal of Computer Applications**99**(15): 36-45.

Sommerville, I. and P. Sawyer (1997). Requirements engineering: a good practice guide, John Wiley & Sons, Inc.

Van den Brand, M. and J. F. Groote (2015). "Software engineering: redundancy is key." Science of Computer Programming**97**: 75-81.

Whittle, J., J. Van Baalen, J. Schumann, P. Robinson, T. Pressburger, J. Penix, P. Oh, M. Lowry and G. Brat (2001). Amphion/NAV: Deductive synthesis of state estimation software. Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on, IEEE.

Wiegers, K. E. (1999). "Automating requirements management." Software Development**7**(7): 1-5.

Winkler, S. and J. Pilgrim (2010). "A survey of traceability in requirements engineering and model-driven development." Software and Systems Modeling (SoSyM)**9**(4): 529-565.

Zisman, A., G. Spanoudakis, E. Pérez-Miñana and P. Krause (2003). Tracing Software Requirements Artifacts. Software Engineering Research and Practice.