

DISS 725 – System Development: Research Paper 2
Software Requirements Engineering – Best Practices

by

Ronald G. Wolak
wolakron@nova.edu

A paper submitted in fulfillment of the requirements
for DISS 725 Spring 2001 – System Development: Research Paper 2

School of Computer and Information Sciences
Nova Southeastern University

May 2001

An Abstract of a Paper Submitted to Nova Southeastern University in Fulfillment of the Requirements for DISS 725 Spring 2001 – System Development: Research Paper 2

DISS 725 – System Development: Research Paper 2
Software Requirements Engineering – Best Practices

by
Ronald G. Wolak

May 2001

The paper that follows was submitted to satisfy the requirements of DISS 725 Spring 2001 – System Development: Research Paper 2. Software requirements engineering lies at the heart of system development life cycle (SDLC) methodologies. The key objective of the software requirements engineering process is to specify a system that in the end will be successful. Software requirements engineering acts as a conduit between the needs of users and the capabilities of software technologies. In the following pages, the paper began with an introduction to the software requirements engineering process. In this discussion, the five phases of software requirements engineering were discussed: elicitation, analysis, specification, verification, and management. This was followed by an in-depth look at current issues and best practices in software requirements engineering. In addition, software requirements engineering methodologies and tools were explored. The paper concluded with a summary of current best practices in software requirements engineering along with recommendations for their use.

Chapter 1 Introduction

Software Requirements Engineering (RE) lies at the heart of system development life cycle (SDLC) methodologies. The primary objective of the requirements engineering process is to specify a system that in the end will be successful. Requirements engineering acts as a conduit between the needs of users and the capabilities of software technologies. One of the major reasons for the failure of software projects is the lack of a complete requirements specification (Connolly, Begg, & Strachan, 1999). The following introductory sections describe the problem to be investigated and the goal to be achieved. In addition, the introduction provides an analysis of the relevance of the research and discusses the paper's five-chapter format.

Problem Statement and Goal

The problems that result from inept, inadequate, or inefficient requirements engineering are expensive and plague most software systems and software development organizations (Sawyer, Sommerville, & Viller, 1999). In response to this problem, the software industry is exhibiting an increased interest in requirements engineering and the benefits it yields. Despite the current trend to develop applications on "Internet time," companies are realizing that the consequences of inadequate requirements engineering are too great (Wieggers, 2000). The goal of this paper is to provide an overview of current requirements engineering best practices to be used by companies when selecting the most appropriate technique.

Relevance

This research paper is relevant to the topic of system requirements engineering. The paper begins with an introduction to the software requirements engineering process and the five phases of software requirements engineering: elicitation, analysis, specification, verification, and management. This is followed by a look into software requirements engineering issues and best practices.

Format

This research paper is a descriptive study formatted in five chapters. The first chapter covers the paper's problem statement and goal, relevance, and format. This is followed in the second chapter by a review of the literature relevant to the problem. In the third chapter, the research methods and online tools and resources employed during the completion of the paper are described. The fourth chapter presents the results of the research and provides an analysis of current requirements engineering issues and best practices. The fifth chapter begins with a summary of the requirements engineering process along with a review of major issues and best practices in the field. This is followed by recommendations for the effective use of these practices. Finally, the paper concludes with an overall summary.

Summary

Software requirements engineering is receiving renewed interest in the software industry. The failure of many software projects can be directly linked to the lack of a

complete requirements specification (Connolly, Begg, & Strachan, 1999). In the following pages, this paper provides a review of literature relevant to this trend, a description of research methods employed, results of the research, recommendations for effective use, and an overall summary.

Chapter 2 Review of Literature

The literature review that follows is organized by subject heading. Those subjects include requirements engineering subspecialties: elicitation, analysis, specification, verification, and management.

While system requirements engineering is concerned with the analysis and documentation of system requirements, software requirements engineering is concerned with the analysis and documentation of software requirements (Thayer & Dorfman, 1999). The primary difference between the two is that system requirements are derived from user needs and software requirements originate from system requirements and specifications. For example, a system requirements engineer works with customers and users while a software requirements engineer works with system requirements documents and engineers.

Subspecialties

The field of software requirements engineering has undergone significant change since 1990 (Thayer & Dorfman, 1999). One example was the addition of requirements engineering as a key process area in the Capability Maturity Model for Software (published by the Software Engineering Institute). Another was the breakdown of requirements engineering into subspecialties: elicitation, analysis, specification, verification, and management.

Elicitation

Software requirements elicitation is the process by which all parties involved in the development of a software system discover, review, and understand user needs and the limitations of the development activity and the software (Thayer & Dorfman, 1999). Playle and Schroeder (1996) addressed issues associated with software requirements elicitation. In their article, they discussed elicitation problems, tools, and methods. Methods included formal (specification languages), informal (group), and prototyping. Prototyping was considered one of the most effective means of requirements elicitation. Both requirements and model fidelity improved as groups developed system prototypes. Formal specification environments that were coupled to graphical interfaces were also discussed. These interfaces typically included entity-relationship and data-flow diagramming tools.

Palmer and Field (1992) also described an integrated environment for gathering requirements that could be traced to specifications using specialized tools. The first tool was an object manager that handled information objects and passed them through to other tools. The second tool was a lexical analyzer that used natural language to classify requirements and relate them to the objects. Also included were automated meeting facilitation tools that facilitated online elicitation of software requirements.

A related article by Goguen and Linde (1993) surveyed and evaluated elicitation techniques. Special attention was paid to social issues related to the process. Methods discussed included introspection, interviews, questionnaires, and protocol, interaction, and discourse analyses. The authors recommended the “zooming” method of

requirements elicitation. Zooming selectively employed more expensive but detailed methods for problems determined to be of higher importance.

Analysis

Software requirements analysis has the goal of evaluating customer and user needs in order to create a definition of software requirements (Thayer & Dorfman, 1999). Methodologies for performing structured analysis were detailed in an article by Svoboda (1990). The term “structured analysis” applied to methodological approaches that guided the developer in the analysis of software requirements. These approaches were concerned with answers to the question “What?” (e.g. What must the system produce? What must the system do?). Traditional structured analysis employed four major tools to identify software requirements. These were the data flow diagram, the data dictionary, the primitive process specification (i.e. MiniSpec), and the structured walkthrough. Svoboda concluded that while the methodology constrains creativity, it helps developers maintain their focus and direction during the system development process.

In a related article, Bailin (1996) discussed object-oriented requirements analysis (OORA). OORA was a method of developing software system requirements in terms of objects and their interactions. The method sought to decompose a system into interacting parts that represented user requirements in a way that was in agreement with object-oriented design. In addition, this technique facilitated the transition from the analysis phase to the design phase and also simplified requirements traceability. Bailin also contended that the use of OORA improved the reusability of requirements analysis products from one system to the next.

Vitharana and Zahedi (1997) recognized the importance of software requirements analysis in building quality software systems. They pointed out that the involvement of users and developers created many communication difficulties. In their research, they formalized the stages of requirements analysis and identified group decision support systems (GDSS) that were effective in improving the process.

Specification

Software requirements specification is the process of developing a document that accurately records all the requirements of a software system (Thayer & Dorfman, 1999). In a related article, Davis and Leffingwell (1999) discussed the importance of the software requirements specification document. They contended that a quality requirements specification exhibited the following characteristics: lack of ambiguity, completeness, consistency, traces to origins, absence of design, enumerated requirements. Numerous software specification standards were described. One example was the IEEE/ANSI 830-1993, IEEE Recommended Practice for Software Requirements Specifications (SRS).

Wiegiers (1995) also discussed the use of the IEEE SRS to provide a well-structured framework for recording functional system requirements as well as the miscellaneous information required to create a quality specification. Specification processes often failed to investigate various quality attributes. The IEEE SRS addressed this problem with a section that dealt with software quality attributes. Wiegiers also discussed two common pitfalls of the requirements specification process. At one extreme was the “back-of-the-napkin” specification that provided a crude approximation of the

software system. The other extreme was the exhaustive specification that led to “analysis paralysis.”

Boehm (1984) also addressed the attributes of a good software requirements specification. They included the following:

- Complete
- Clear
- Correct
- Understandable
- Consistent
- Concise
- Feasible

Boehm pointed out that although the evaluation of a requirements specification with respect to these attributes was highly subjective and qualitative, their importance dictated that the validation be conducted.

Verification

Software requirements verification is the process of confirming that the requirements specification agrees with the system requirements (Thayer & Dorfman, 1999). In addition, it ensures that the specification adheres to the document standards of the requirements phase and provides a sufficient basis for the preliminary design phase. Wallace and Ippolito (1996) discussed the use of software verification as an aid in determining that software requirements were implemented correctly and were traceable to the system requirements. In addition, they described how the software requirements verification activity checked that system requirements were appropriately allocated to software and that software requirements were adequately specified (e.g. correct, complete, concise, and testable). Wallace and Ippolito also discussed the three major classes of verification techniques: static, dynamic, and formal analysis.

In a related article, Palmer (1996) discussed the importance of traceability to verification and validation. Traceability was needed for information abstraction, quick access to information, and insight into the techniques used for system development. Traceability also promoted change control, development process control, and risk control. The article discussed commercially available CASE tools for traceability. These included Cadre TeamWork for Real-Time Structured Analysis (CADRE), Requirements Traceability Manager (RTM), SLATE, and DOORS. These tools employed an entity-relation-attribute-like schema to capture information in a database. Other tools and techniques discussed were Software Requirements Methodology (SRM), and Problem Statement Language/Problem Statement Analyzer (PSL/PSA).

Davis et al. (1993) provided the beginnings for definitions of metrics suitable for qualities related to software requirements specifications (e.g. unambiguous, complete, correct, etc.). The article explored the concept of software requirements specification quality, defined attributes that contributed to quality, and suggested techniques to measure these attributes.

Management

Software requirements management involves the control and planning of the requirements elicitation, specification, analysis, and verification (Thayer & Dorfman,

1999). Yeh and Ng (1990) discussed the problem of deriving software requirements at the management level. The paper proposed a new paradigm for software development – the abstraction-based software lifecycle model. The model emphasized requirements, specification, and design. In addition, the model incorporated evaluation and validation activities into the development process.

In another related article, Wiegers (1999) described the benefits provided by current requirements management tools. These tools automated requirements management tasks that included: managing versions and changes, storing requirements attributes, linking requirements to other system elements, tracking status, viewing requirements subsets, controlling access, and communicating with stakeholders. Tools reviewed in the article were DOORS, RequisitePro, RTM Workshop, and Caliber-RM. All of the tools had traceability features and allowed the set up of user groups with full permissioning. Most allowed users to incorporate non-textual objects such as images and Microsoft Excel spreadsheets.

Summary

The literature review presented above was organized by subject heading. The subjects reviewed included five software requirements engineering subspecialties: elicitation, analysis, specification, verification, and management.

Chapter 3 Methodology

Research Type

This paper was a research-based descriptive study. The key outcome of the investigation was the identification of software requirements engineering issues and best practices.

Research Methods Employed

The primary research method employed throughout the course of writing this paper was browser-based Internet searches. The literature reviewed included textbooks, journal articles, and magazine articles referenced by a select set of online resources. Relevant texts were located, ordered, and delivered using the IEEE Computer.org Internet site. The full text articles from journals and magazines were located and subsequently downloaded.

Online Tools and Resources

A variety of online resources were used to locate and download literature relevant to the goal of the paper. These resources included ACM Search (www.acm.org/dl/search.html), IEEE Digital Library (<http://computer.org/search.htm>), and ProQuest Direct (<http://proquest.umi.com/>). Perhaps the most powerful search tools to be employed were the intelligent search agents Copernic 2001 and LexiBot.

Copernic 2001 is a well-documented freeware search agent (Copernic, 2001). It uses predefined channel sets, which allows researchers to target inquiries to all major Web search engines and also search for relevant text in newsgroups. Copernic conducts fast, multithreaded, full Boolean searches with progress displays and customizable search depth. Once results are compiled, Copernic displays returns (including name, location, and introductory text) in a right-click-enhanced list box sorted by relevance.

Another search technology utilized to gather literature was LexiBot from BrightPlanet (LexiBot, 2001). The LexiBot desktop search client acts as a universal translator for all dialects of search engines and searchable databases. LexiBot is able to search 150 services at one time using a standard query format. In addition, LexiBot's search technology is capable of identifying, retrieving, qualifying, and organizing "deep" and "surface" content from the Internet.

Summary

In summary, this paper was a research-based descriptive study. Browser-based Internet searches were the primary research method employed. These searches queried databases that included the ACM, IEEE, and ProQuest Direct. Specialized client-based search technologies (i.e. Copernic 2001 and LexiBot) also aided in locating relevant literature.

Chapter 4 Results

Despite years of progress in the field, software development organizations continue to experience difficulty with the elicitation, analysis, specification, verification, and management of software requirements (Sawyer, Sommerville, & Viller, 1999). The following sections begin with an investigation into the issues related to the production of software requirements. This is followed by a discussion of current best practices in each subspecialty of software requirements engineering.

Issues

There are many issues related to the production of accurate, complete, and quality software requirements (Davis, 1990). These issues include the following:

- Lack of experienced software project managers
- Belief by managers that programming and testing are the major efforts of any software project
- Lack of trained system engineers
- Inability of software engineers to write correct software requirements specifications
- Customer inability to understand the purpose and importance of requirements specifications
- Inability to select the correct methodology and tool to use to develop software requirements specifications
- Unwillingness to acknowledge the importance of system requirements to the development of software requirements

The issues described above create the current environment in which the following are commonplace (Thayer & Dorfman, 1999):

- User requirements are not well identified and documented
- Development teams respond to incomplete project specifications
- Inadequate system analyses are performed during the study period
- Operation environments and system concepts are not completely understood
- Changing technology creates pressure to decrease project cycle times
- Point design acceptance increases as project cycle times decrease
- Budget and schedule estimates are not realistic
- System concepts and operational environments are not adequately understood

In fact, the source of most software requirement errors is the failure of developers to accomplish one of three primary goals (Faulk, 1996). The first is to understand exactly what the software is required to do. The second is to communicate their understanding of what is required to all people involved in the project. Third is to provide a way to control production and ensure the software system satisfies requirements.

Best Practices

The issues described above are best overcome by the application of established best practices (Sommerville & Sawyer, 1997). The following are best practices for each subspecialty of software requirements engineering.

Elicitation

Key practices in the elicitation phase of requirements engineering include defining business requirements, involving users, focusing on user tasks, and defining quality attributes (Wiegiers, 2000). Documented business requirements address the problem of scope creep by summarizing the major features of the initial and subsequent releases of an application. In addition, business requirements identify specific features that an application will not include.

Ensuring extensive user involvement is another best practice. Insufficient user participation is frequently the reason software projects fail (Gause & Lawrence, 1999). Determining which user classes will carry the most weight in resolving conflicting requests and selecting design choices is also important. The “product champion” model is one effective way to engage user representatives in the software development process.

The elicitation phase also benefits from focusing on user tasks (Collard, 1999). The use case approach is one method of accomplishing this. Use cases help capture essential functional requirements and also identify exception conditions. In addition, use cases assist in developing test cases of proposed systems early in the development process.

Another best practice is to define quality attributes (Wiegiers, 2000). Quality attributes are characteristics users are able to observe (e.g. system availability, integrity, efficiency, reliability, and usability). These frequently unstated user performance expectations often result in the lack of user acceptance of a finished system unless they are brought to surface and documented early in the requirements process.

Analysis

Requirements analysis involves the breakdown of high-level requirements into detailed functional requirements. However, resource limitations often dictate the prioritization of these functional requirements (Karlsson, 1996). Prioritizing requirements is another best practice that increases the success rates of software projects. Prioritization helps developers plan for staged releases and respond to requests for added functionality. In addition, prioritization helps avoid the loss of planned functionality late in a project when the only concern becomes getting the application out the door.

One method of prioritization is the grouping of requirements into three priority categories (Wiegiers, 1999b). These are high, medium, and low or essential, conditional, and optional. Keeping prioritization simple facilitates the process of updating priorities as a project progresses.

Specification

The use of requirements management tools is a best practice that is essential during the specification of software requirements (Wiegiers, 1999a). Requirements management tools automate tasks such as: managing versions and changes, storing requirements attributes, tracking status, viewing requirements subsets, controlling access,

and communicating with stakeholders. These tools also provide traceability links between individual requirements and other elements of the system.

Verification

Software requirements verification confirms that the requirements specification provides an adequate basis for design, construction, and testing (Wallace & Ippolito, 1996). An important best practice in this subspecialty is the use of formal inspection methods. Formal inspections are often slow and painful but reduce overall project costs significantly because costly requirements defects are discovered early in the development process.

Management

Successful requirements management requires a change control process that manages how changes are submitted, evaluated, chosen, and enacted (Wieggers, 2000). One best practice in this subspecialty is the use of a change management tool that automatically keeps track of changes to individual requirements, communicates the changes, and maintains revisions histories.

Summary

The results chapter presented above began with an investigation into the issues related to the production of software requirements. This was followed by a discussion of current best practices in the five subspecialties of software requirements engineering: elicitation, analysis, specification, verification, and management.

Chapter 5 Conclusion

Software requirements engineering has the goal of specifying a successful software system. In fact, one of the major reasons software systems are not successful is the lack of a complete requirements specification (Connolly, Begg, & Strachan, 1999). The problems created by inadequate or inefficient requirements practices are expensive and plague most software development efforts. This has led to a renewed interest in software requirements engineering within the software industry and the division of requirements engineering into five subspecialties: elicitation, analysis, specification, verification, and management.

Issues in the field are related to the production of accurate, complete, and quality software requirements (Davis, 1990). They include a lack of experienced project managers and system engineers, the inability of customers and managers to understand the importance and purpose of requirements specifications, and the inability to select the correct methodology and tool to use to develop requirements. These problems have created an environment in which user requirements and operational environments are not completely understood, changing technologies create pressure to decrease cycle times, development teams respond to incomplete project specifications, and budget and schedule estimates are not realistic.

The above issues are best overcome by the application of established best practices (Sommerville & Sawyer, 1997). Examples include defining business requirements, involving users, focusing on user tasks, and defining quality attributes. In addition, the prioritization of requirements, the use of requirements management tools, and the implementation of formal requirements inspections are effective in ensuring a successful software system.

Finally, software requirements engineering is primarily a communications (not a technical) activity (Faulk, 1996). Any approach to software requirements engineering that does not take both human and technical concerns into account will have limited success. In addition, any developer looking for an easy way out (i.e. new methods that make the process easy) must realize that the requirements engineering process is inherently hard. In addition to the need for discipline, numerous other characteristics are required to understand software requirements and their specification. While the cost of not having quality software requirements is often high, the benefits of having good requirements are not inexpensive. The task is difficult and cannot be performed by software developers without adequate experience, training, and resources. Successful requirement engineering requires the commitment to provide the time, resources, and means to do the job right.

Summary

The research paper presented above was a descriptive study formatted in five chapters. The first chapter covered the paper's problem statement and goal, relevance, and format. This was followed in the second chapter by a review of the literature relevant to the problem. In the third chapter, the research methods and online tools and resources employed during the completion of the paper were described. The fourth chapter presented the results of the research and provided an analysis of current requirements engineering issues and best practices. Finally, the last chapter provided a summary of

requirements engineering along with a review of major issues and best practices in the field.

References

- Bailin, S. (1996). Object-Oriented Requirements Analysis. In R. Thayer & M. Dorfman (Eds.), *Software Requirements Engineering* (Second ed., pp. 334-355). Washington, DC: IEEE Computer Society Press.
- Boehm, B. (1984). Verifying and validating software requirements and design specifications. *IEEE Software*, 1(1), 75-88.
- Collard, R. (1999). Test design. *Software Testing and Quality Engineering*, 1(4), 30-37.
- Connolly, T., Begg, C., & Strachan, A. (1999). *Database Systems: A Practical Approach to Design, Implementation, and Management* (Second ed.). Reading, MA: Addison-Wesley.
- Copernic (2001). Copernic 2001. Retrieved April 1, 2001, from the World Wide Web: <http://www.copernic.com>.
- Davis, A. (1990). *Software Requirements: Analysis and Specification*. Englewood Cliff, N.J.: Prentice-Hall.
- Davis, A., & Leffingwell, D. (1999). Making Requirements Management Work for You. *Software Technology Support Center*. Retrieved April 29, 2001, from the World Wide Web: <http://www.stsc.hill.af.mil/CrossTalk/1999/APR/davis.asp>.
- Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledebor, G., Reynolds, P., Sitaram, P., Ta, A., & Theofanos, M. (1993). Identifying and Measuring Quality in a Software Requirements Specification. In R. Thayer & M. Dorfman (Eds.), *Software Requirements Engineering* (Second ed., pp. 194-205). Washington, DC: IEEE Computer Society Press.
- Faulk, S. (1996). Software Requirements: A Tutorial. In R. Thayer & M. Dorfman (Eds.), *Software Requirements Engineering* (Second ed., pp. 158-179). Washington, DC: IEEE Computer Society Press.
- Gause, D., & Lawrence, B. (1999). User-driven design. *Software Testing & Quality Engineering*, 1(1), 22-28.
- Goguen, J., & Linde, C. (1993). Techniques for Requirements Elicitation. In R. Thayer & M. Dorfman (Eds.), *Software Requirements Engineering* (Second ed., pp. 140-152). Washington, DC: IEEE Computer Society Press.
- Karlsson, J. (1996). Software requirements prioritizing. *Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96)*, 110-116.
- LexiBot (2001). Our Technology - Results: The LexiBot Expression. *BrightPlanet*.

Retrieved April 1, 2001, from the World Wide Web:
<http://www.brightplanet.com/technology/results2.asp>.

- Palmer, J. (1996). Traceability. In R. Thayer & M. Dorfman (Eds.), *Software Requirements Engineering* (Second ed., pp. 412-422). Washington, DC: IEEE Computer Society Press.
- Palmer, J., & Fields, A. (1992). An integrated environment for requirements engineering. *IEEE Software*, 9(3).
- Playle, G., & Schroeder, C. (1996). Software Requirements Elicitation: Problems, Tools, and Techniques. *Software Technology Support Center*. Retrieved May 6, 2001, from the World Wide Web:
<http://www.stsc.hill.af.mil/CrossTalk/1996/dec/xt96d12e.asp>.
- Sawyer, P., Sommerville, I., & Viller, S. (1999). Capturing the benefits of requirements engineering. *IEEE Software*, 16(2), 78-85.
- Sommerville, I., & Sawyer, P. (1997). *Requirements Engineering: A Good Practice Guide*. New York: John Wiley & Sons, Inc.
- Svoboda, C. (1990). Structured Analysis. In R. Thayer & M. Dorfman (Eds.), *Software Requirements Engineering* (Second ed., pp. 303-322). Washington, DC: IEEE Computer Society Press.
- Thayer, R., & Dorfman, M. (Eds.). (1999). *Software Requirements Engineering* (Second ed.). Washington, DC: IEEE Computer Society Press.
- Vitharana, P., & Zahedi, F. (1997). Group Decision Support for Software Requirements Analysis. *Association for Information Systems*. Retrieved May 6, 2001, from the World Wide Web: http://hsb.baylor.edu/ramsower/ais.ac.97/papers/vith_zah.htm.
- Wallace, D., & Ippolito, L. (1996). Verifying and Validating Software Requirements Specifications. In R. Thayer & M. Dorfman (Eds.), *Software Requirements Engineering* (Second ed., pp. 437-452). Washington, DC: IEEE Computer Society Press.
- Wiegiers, K. (1999a). Automating Requirements Management. *Software Development Online*. Retrieved April 22, 2001, from the World Wide Web:
<http://www.sdmagazine.com/articles/1999/9907/9907d/9907d.htm>.
- Wiegiers, K. (1999b). First Things First: Prioritizing Requirements. *Software Development Online*. Retrieved April 22, 2001, from the World Wide Web:
<http://www.sdmagazine.com/articles/1999/9909/9909b/9909b.htm>.
- Wiegiers, K. (1999c). In Search of Excellent Requirements. *Process Impact*. Retrieved

April 22, 2001, from the World Wide Web:
http://www.processimpact.com/articles/exc_reqs.html.

Wiegers, K. (2000). When Telepathy Won't Do: Requirements Engineering Key Practices. *Process Impact*. Retrieved April 22, 2001, from the World Wide Web: <http://www.processimpact.com/articles/telepathy.html>.

Yeh, R., & Ng, P. (1990). Software Requirements - A Management Perspective. In R. Thayer & M. Dorfman (Eds.), *Software Requirements Engineering* (Second ed., pp. 425-436). Washington, DC: IEEE Computer Society Press.