JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 1, March-April 2003

Modern Requirements Specification

Donald Firesmith, Firesmith Consulting, U.S.A.

Abstract

Requirements specification is the requirements engineering task during which analyzed requirements are properly documented for use by their intended audiences. Traditionally, this involved the requirements team using a word processing program to produce a single requirements specification document during an initial requirements phase of a project. However, trends in system development have made the numerous problems with this approach abundantly clear. Improvements in requirements tools have not only enabled better requirements management; they have also enabled the automatic generation of consistent, current, audience-specific requirements specifications that far better meet the needs of their individual audiences.

1 TRADITIONAL REQUIREMENTS SPECIFICATION

During the 1970s and 1980s, requirements specification seemed at first glance to be a relatively simple task. During the initial requirements phase of a project, the requirements engineers would elicit functional requirements from the various stakeholders using a functional decomposition method such as structured analysis. Then they would use a simple word processing program to manually document these requirements in a single requirements specification document, which after a review and some minor iteration would be approved, placed under configuration control (for all practical purposes frozen), and published to its audiences. Everyone would base their work on the same requirements specification, and everyone could rest assured that the specification would not significantly change during the following design, coding, and testing phases. Thus, requirements specification was a manual task that had its place during the initial phase of the waterfall development cycle, and it was basically complete early in the project. Requirements specification was also a paper-document-based process with little if any real tool support. Unfortunately, many projects still use a similar approach to perform requirements specification.

Challenges to Address

However, the previously described approach to the requirements specification task has numerous well-known problems. Being the result of a manual process, the single requirements specification document was time consuming and expensive to produce and was rarely maintained beyond being frozen and placed under configuration control. The resulting requirements specification was also error prone, and thus typically incomplete, inconsistent (both internally and externally), ambiguous, hard to read, and rarely up-to-date. It was definitely much too expensive to create multiple versions of the requirements specification for multiple audiences so one size had to fit all readers, providing much too much detail for some readers (e.g., executive management) and much too little detail for others (e.g., independent testers). The requirements specification documents also typically did not contain any metadata about the requirements (e.g., scheduling information, assignment to developers, status), thereby making requirements management very difficult.

Whereas these problems were observable from the very beginning, requirements engineers had little choice but to live with them because alternatives were either unknown or impractical. However, over time, trends in system development have both made the original approach infeasible as well as enabling new approaches that much better achieve the goals of requirements engineering: the production of requirements specifications that are correct, complete, internally and externally consistent, current, and audience-appropriate (i.e., supportive of the role-specific tasks of its numerous audiences).

2 TRENDS IMPACTING REQUIREMENTS SPECIFICATION

Modern Development Cycles

Perhaps the most significant trend affecting the requirements specification task is the replacement of the traditional waterfall development cycle with modern iterative, incremental, parallel, and timeboxed development cycles. In the classic waterfall cycle, the vast majority of the requirements were specified and frozen by the end of the requirements phase, which occurred before significant amounts of architecting, design, implementation, integration, and testing had occurred. However, this approach was never very successful because it was based on false assumptions such as the requirements being well known and stable near the beginning of the cycle. As these two preconditions are rarely true, requirements engineering took a major leap forward when modern development cycles were introduced. Today, the most effective development cycles have the following characteristics:

- **Iterative Requirements Engineering**. A development cycle is iterative when it recognizes that non-trivial work products (such as requirements specifications) are tentative and will initially contain numerous defects due to human errors and ignorance. These work products must be iterated (i.e., fixed in an ongoing manner) as their defects are identified and corrected. Thus, parts of the development process (e.g., the requirements elicitation, analysis, and specification tasks) are repeated on existing work products (e.g., the requirements, requirements models, and requirements specifications) to improve them.
- **Incremental Requirements Engineering**. A development cycle is incremental when it recognizes that many work products are too large and complex to be

produced all at once in a big-bang manner. A non-trivial application may literally have hundreds of requirements, which take a significant amount of time to elicit, analyze, and specify. Thus, the requirement elicitation, analysis, and specification tasks will typically occur incrementally, with new requirements being added on a daily or weekly basis over a significant portion of the development cycle. Thus, incremental development means that parts of the development process are repeated to add additional work products or more to existing work products.

- **Parallel Requirements Engineering**. A development cycle is parallel when it recognizes that numerous activities and tasks must happen concurrently if the application is to be completed within any reasonable time frame. Thus, we cannot wait for the completion of the requirements engineering activity before starting architecting, design, implementation (e.g., prototyping), and testing (e.g., test planning, test case development). Similarly, we perform multiple requirements engineering tasks such as requirements elicitation (including both discovery and invention), analysis, specification, and management in parallel. This has the advantage of bringing more teams, roles, and persons to bear on the problem earlier, thereby improving overall endeavor productivity. However, this parallel development of requirements, architectures, designs, implementations, and tests leads to increased iteration (and thus improvement) of the requirements.
- **Timeboxed Requirements Engineering**. A development cycle is time-boxed when either its tasks or its work products are scheduled so that they must be completed by specified deadlines. Instead of the waterfall development cycle's traditional milestones based on the completion of major activities such as requirements engineering, an iterative, incremental, parallel, timeboxed development cycle tends to have new time-based milestones based on new phases (not activities) as well as numerous, regularly-scheduled short-duration "inch-pebbles," possibly based on parts of work products such as parts of the requirements specifications (e.g., specific use cases, normal paths through a single actor's use cases, and security requirements).

An iterative, incremental, parallel, and timeboxed development cycle significantly affects requirements specification. Such a development cycle recognizes the need for (and encourages) constant changes to individual requirements and requirements models, which in turn results in constant changes (e.g., improvements, corrections, additions) to the associated requirements specifications. And this results in significant difficulties in keeping the requirements specification consistent and up-to-date, especially if they are traditional manually produced paper documents.

Numerous Stakeholders with Different Needs

When requirements specifications were manually produced paper documents, requirements specification was very labor intensive. Resources (e.g., staffing, schedule) were inadequate to keep even a single requirements specification complete and up-to-date. Thus, there were definitely insufficient resources to produce multiple versions of these specifications. So every stakeholder had to read and use the same requirements specification, no matter what his or her role and responsibilities were. The resulting specifications were typically too large and detailed for management to read and understand, and they were typically too incomplete and at too high of a level for the independent testing team to use to develop adequate test cases.

It is important to recognize that the requirements specifications have many different stakeholders and that different kinds of stakeholders have very different needs. Thus, different stakeholders need different requirements specifications with different scopes, amounts of formality and rigor, levels of abstraction (detail), and metadata (i.e., attributes about the requirements). For example, the following roles have very different needs when it comes to requirements specifications:

- **Executives**. Executives typically use requirements specifications as the basis for executive decisions regarding funding and approval and to manage the scope of programs of related projects. They need executive summary level documents that are short, concise, and easy for non-technical readers to understand.
- **Managers**. Managers typically use requirements specifications to manage project scope and to estimate endeavor schedule and required resources. They need specifications that are somewhat more detailed than executive summaries, but still at a level of abstraction so that they can concentrate on the requirements forest and so avoid becoming lost among the individual requirements trees.
- **Subject Matter Experts**. Domain experts act both as sources of requirements (e.g., business object models, business process models) as well as reviewers of requirements to ensure that they are properly specified. They need requirements specifications that concentrate on their area of subject matter expertise.
- **Architects**. Architects need to rapidly identify the architecturally significant requirements. They need more information that is more detailed than managers do, but should not be bogged down with numerous low-level requirements that do not had any real architectural significance.
- **Designers and Implementers**. The actual developers need much more complete and detailed requirements than the architects do. They also need requirements that are organized so that they can concentrate on only those requirements that are relevant to the components that they are developing, even when there is no one-to-one mapping between the requirements and components which are based on completely different kinds of abstractions (e.g., functional requirements vs. object-oriented components). They also critically need to be notified when their requirements of interest change (iteration) or are added to (incremental development), possibly via some kind of "publish and subscribe" mechanism. Note that this notification must be fine grained so that they are only notified when relevant requirements change, not when the entire specification changes, which will probably be on a daily basis for the first two-thirds of the endeavor.
- **Testers**. Like designers and implementers, testers also need the most complete and detailed requirements if they are to produce "complete" test suites of test cases. For example, if managers and architects may get by with use case information and designers and implementers may get by with basic use case path

information, testers definitely need detailed use case path information including preconditions and post conditions.

Clearly, the preceding has demonstrated that one size does not fit all when it comes to requirements specifications. Trying to have a single paper requirements specification often leads to confusion and disagreement. For example, managers may request that detailed information that is critical to developers and testers be removed because they do not see the cost-effectiveness in producing it and they often loose track of the requirements forest (endeavor scope) because of the overwhelming number of requirements trees.

Increasing Application Size and Complexity

Over the last quarter century, typical applications have consistently grown larger and more complex. Monolithic applications have been replaced by client-server applications which in turn have been replaced by n-tier applications. Stand-alone applications have given way to highly integrated and interoperable applications including enterprise application integration (EAI) applications. Software has been embedded an every conceivable type of hardware (e.g., phones, televisions, appliances, automobiles, automated teller machines) to the point where the average person interacts with literally dozens if not hundreds of computers each day. Applications have also increased in criticality with, for example, simple informational websites being replaced with eCommerce and eMarketplace websites.

As applications have become larger, more complex, more business and safety critical, and more ubiquitous, their requirements have grown in number, complexity, and type. Requirements engineers can no longer concentrate almost totally on functional requirements; instead, we must also adequately address data requirements, quality requirements, application programmer interface (API) requirements, and various types of architectural, design, implementation, and testing constraints as well as business rules and relevant laws and regulations. Thus, for example, increasingly stringent quality requirements such as operational availability, performance, interoperability, scalability, and security often have a bigger influence on architectures, cost, and schedule than the vast majority of functional requirements.

Larger, more complex applications have also traditionally meant larger, more complex requirements specification documents. Yet, as such documents have become larger, they have also become more difficult to understand, review, and use. No single documentation organization or level of abstraction scales to such large documents, especially when all of the different audiences for such documents are considered.

Better Methods and Notations for Requirements Engineering Tasks

When I started doing requirements in the 1970s, I would not dignify what we did with the term requirements engineering. The vast majority of requirements were inadequately analyzed and categorized, and most requirements engineers considered themselves lucky if they were allowed to use some Structured Analysis to do a little top-down functional decomposition and data flow analysis.

Requirements engineering has greatly advanced since those dark ages. We now recognize many useful requirements tasks such as business analysis, application visioning, requirements elicitation, requirements analysis, requirements specification, and requirements management. We have numerous requirements analysis methods, many of which (e.g., use case analysis) are widely recognized as being significant improvements over the older techniques. These requirements analysis methods use multiple types and levels of abstractions that provide different views of the requirements for their different audiences. We have more and better requirements models including more complete hierarchies of different kinds of requirements. Although neither perfect nor complete, we nevertheless have better modeling languages (e.g., UML), better formal and semi-formal requirements methods, and better-standardized content for our requirements specifications.

All of this significantly affects requirements specification by affecting the types of information specified, their content and format, and how the requirements specifications are organized. We must specify requirements in many forms including native language textual requirements, requirements models in graphical modeling languages, decision tables, formally specified requirements in specification languages, and more.

Better Requirements Tool Support

Whereas the requirements engineer's work has become more complex and challenging, the appearance of ever more powerful and user-friendly requirements tools has counteracted some of the previous negative trends and even made certain approaches feasible for the first time. Now we have integrated requirements tools that support more of the requirements engineering tasks such as requirements elicitation, analysis, specification, and management. These tools have been better integrated with other tools to support other related tasks outside of the requirements engineering activity such as scope management (management), version control and configuration control (configuration management), and quality assurance and quality control (quality engineering). These tools can also now support multiple, distributed users.

And perhaps, most importantly, these tools tend to be repository based (often built on an object database or extended relational database), so that they support a finer level of granularity. Thus, individual requirements can be entered into the requirements repositories in an incremental manner, individual requirements can be iterated, requirements metadata (e.g., priority, traceability information, assignment to components and teams) can also be stored with the associated requirements, developers can be notified when relevant individual requirements are modified, and requirements specifications of various types, contents, and levels of detail can be automatically generated from the requirements repository using appropriate selection criteria and templates.

An interesting way of looking at this trend is to see how solutions have evolved over time to handle larger, more complex sets of requirements:

1. Word Processing Tools such as MS Word
2. Spreadsheets such as MS Excel
3. Word Processing Tools with embedded spreadsheets.
4. Databases with ad hoc report generation capabilities
5. Requirements Management Tools
6. Requirements Tools supporting more requirements engineering tasks than just requirements management
7. Requirements Tools that are properly integrated into an Integrated Development Environment (IDE)

Unfortunately, current requirements tools do not yet properly support all of these activities and tasks, and the amount of support that exists varies from tool to tool. Later in this column, I list several recommendations that can be used when evaluating requirements tools and when determining the workarounds that are required to achieve all of the benefits promised in this column.

3 IMPROVING THE REQUIREMENTS SPECIFICATION TASK

Based on the previously mentioned challenges to and trends affecting requirements engineering in general (and requirements specification in particular), what should we do? I would make the following recommendations designed to improve the requirements specifications produced by the requirements specification task.

Requirements Repository

Store your requirements in a requirements repository instead of a paper document. Keep the granularity of the repository small so that individual requirements can be entered, iterated, approved, placed under configuration management, published, managed, and traced. Thus, requirements should be considered to be individual objects and the last thing you want to do is to store a complete requirements specification as a binary large object (BLOB). Ensure that the requirements repository stores all kinds of requirements related information including individual requirements, requirements metadata (the attributes of requirements objects), and requirements models (aggregate objects) including diagrams and tables. The requirements repository should be based on an object database, XML database, or extended relational database. As examples, one could evaluate the requirements tools CaliberRM and DOORS, both of which are based on object databases. One should also not that this recommendation is critical and enables the remaining recommendations.

Automatic Specification Generation

Do whatever it takes (within reason) to enable the automatic generation of requirements specifications from the requirements repository. This goes beyond the simple use of international (e.g., IEEE830-1998), industry (e.g. OPEN or RUP), or business-internal templates for one or more requirements specifications. It also includes the creation of arbitrary requirements reports for requirements management and other purposes. This should also include the generation of the entire publishable requirements specification and not just the generation of a part of the requirements specification that then requires significant amounts of manual labor to complete. This enables the requirements specifications to be current with the official requirements in the repository. It also enables the generation of electronic specifications and reports that save a huge amount of paper in an iterative development cycle in which the requirements change on essentially a daily basis.

The first recommendation for a requirements repository as well as the current recommendation both recognize the separation of Model and View that has been so beneficial in the production of graphical user interfaces. They also recognize the importance of separating the requirements (the model) from potentially multiple views (the specifications) of the same model. Because the publication of requirements specification ideally should involve the publication of time-critical, audience-specific, and even personalized information, the requirements engineering community would also be well advised to consider the content management activity made popular by informational websites[1].

Different Specifications for Different Audiences

Once you have your requirements stored in a well-organized requirements repository and the ability to automatically generate requirements specifications from that repository, then you have the ability to produce multiple audience-specific versions of the requirements specifications. This is nothing more than the recognition that there is often a need for multiple views (specifications, reports) of different parts of the same model (requirements). This includes specifications and reports that differ in the types of requirements viewed (e.g., functional requirements vs. interface requirements), the different levels of requirements details (executive overview specifications vs. detailed requirements specifications for testers), predefined specifications vs. ad hoc reports, and specifications based on metadata (specifications or reports based on requirements due date, status, owner, last modified date, etc.).

[1] For example, look at the webpages concerning content management in my informational website at www.donald-firesmith.com.

Requirements Tools

Beyond having a repository from which requirements specifications and reports can be automatically generated, one actually needs one (or possibly more) requirements tools based on such a repository. Such a tool should have several critical properties:

- **User Interface**. The best way to enter requirements and their metadata is by using a user-friendly graphical user interface that allows one to easily enter and maintain individual requirements, their metadata, and requirements models including text, diagrams, and tables, etc. Notice that this is very different from asking requirements teams to first develop a single requirements specification using a word processing program such as MS Word and then inputting that specification into a database (although that is not a bad "nice to have" feature for reuse of existing requirements specifications). The user interface should understand the underlying requirements model including requirements types, their common and type specific metadata, the different types of models, the different types of diagrams, etc. The user interface should not only support requirements input and maintenance but also requirements specification and report generation including template creation, querying, and actual production.
- **Requirements Engineering Support**. The requirements tool should be complete in that it should support all tasks of requirements engineering including business analysis, cost/benefit analysis, application visioning, and requirements elicitation, analysis, specification, and management. It should support multiple requirements analysis approaches so that multiple types of models (e.g., use cases, decision tables, state models, context models) can be specified. It should also support the entire requirements model including all types of requirements including functional requirements, data requirements, quality requirements (a large list), interface requirements, and constraints (also a large list). Requirements management tools should also include requirements traceability to architecture, design, implementation, and testing work products and back.
- **Support for Related Activities**. The requirements tool should support tasks of other activities if those tasks are related to requirements engineering. This should include scope control, configuration management, and quality engineering. This includes interoperability with management, configuration management, quality engineering, modeling, and testing tools for forward and reverse engineering. Thus, a requirements tool should not be stand-alone, but a critical component of an integrated development environment.
- **Team Development**. Requirements engineering is best performed by a cross-functional requirements team that provides an adequate experience base to capture all of the requirements and to iterate them in a timely fashion. Although it is useful to have a technical writer versed in requirements modeling and the use of modeling tools to be the primary person to initially capture the requirement during joint requirements engineering sessions, all requirements team members should be able to work on the requirements simultaneously (another good reason for fine granularity in the requirements repository). Similarly, other members of the

endeavor team need to have simultaneous access to the requirements for purposes of learning, evaluation, and approval.

- **Security**. Requirements for many applications involve proprietary information, trade secrets, or even national secrets. Any requirements management tool should support the security of the requirements including the identification, authentication, and authorization to perform role-specific tasks of its users. It should also include privacy, integrity, and non-repudiation of requirements and their updates.
- **Other Quality Factors**. A requirements tool is an application in many ways like any other. Thus, requirements tools to be used for the specification of requirements should also have the appropriate amounts of other quality factors besides security and interoperability. Thus, when evaluating such tools, consider the typical quality factors such as completeness, internationalization, performance, scalability, usability, and user friendliness.
- **Distributed Development**. Today, it is not unusual for applications to be built by numerous teams and organizations that are geographically distributed. A requirements tool should support requirements engineering including specification by distributed users.
- **Requirements Reuse**. The requirements tools should enable the easy incorporation of existing requirements into its repository so that each endeavor need not start from scratch. Because these requirements were probably generated using traditional approaches, the requirements tool needs to be able to parse the old requirements specifications, recognize potential requirements and incorporate them in a manner that will be easy for them to be reviewed and either accepted as is, accepted with modification, or rejected. However, the recommendation is that all requirements (even reused requirements) be stored as individual objects at a high-level of detail.
- **Not Just a CASE Tool**. An adequate requirements tool is more than just a simple stand-alone modeling tool or requirements repository. It should be part of an Integrated Development Environment (IDE), but it is not merely a simple Computer Aided Software Engineering (CASE) tool in the traditional sense.

Although the subject matter of this article is modern requirements specification, it is now clear that one cannot talk about requirements specification without addressing many other tasks within requirements engineering. And it is senseless to talk about using a repository-based requirement tool to support the automatic generation of role-specific requirements specifications without addressing the many other properties of such a tool without which it would be useless or impractical.

4 CONCLUSION

This article has addressed several problems with traditional paper-based requirements specification approaches and how trends in software engineering have exacerbated these problems. As illustrated in the following figure, it has also recommended a requirements specification approach to solve these problems that is founded on the use of modern requirements tools based on fine-graned requirements repositories. Using an appropriate tool based on these recommendations, you can automatically, easily, and inexpensively generate various types of high-quality requirements specifications that are tailored to meet the individual needs of their various audiences.

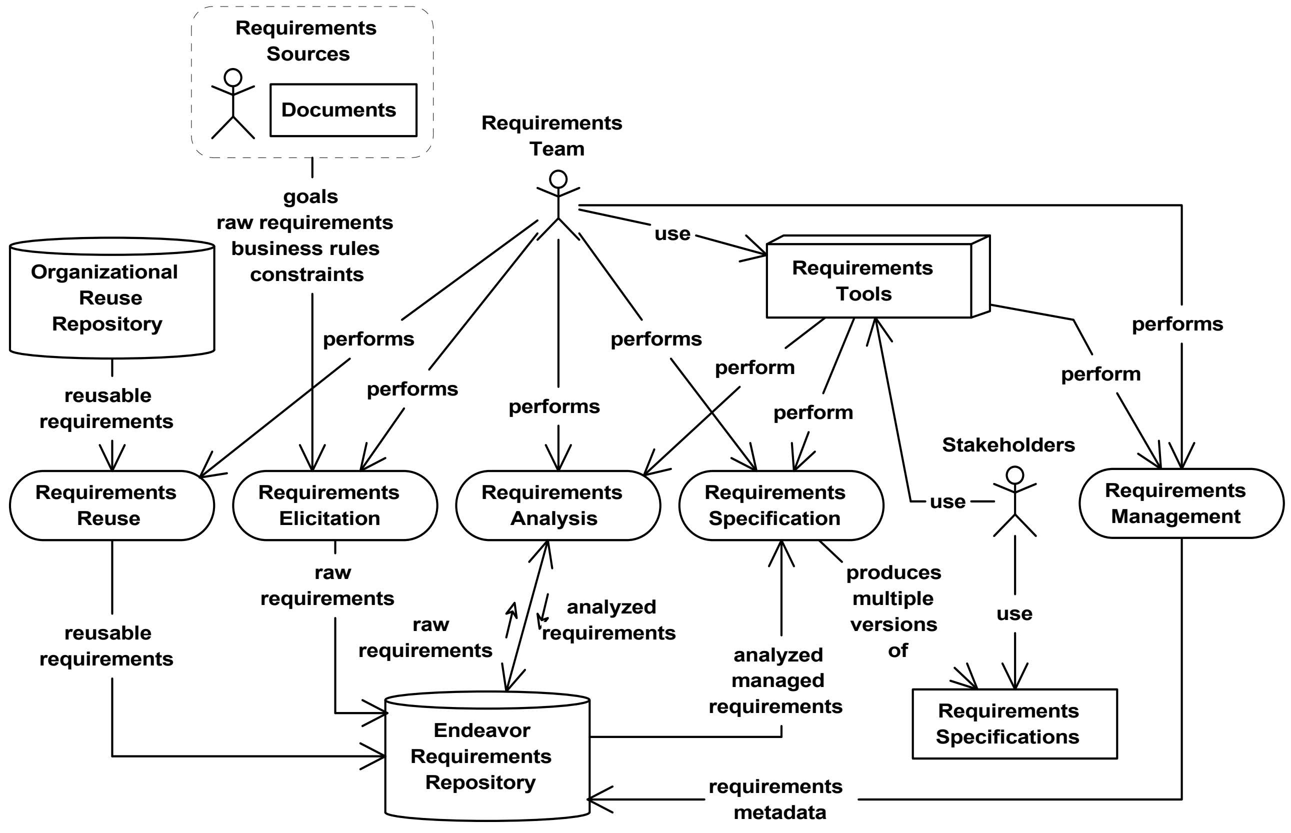

Fig. 1: Repository-Based Requirements Specification

ACKNOWLEDGEMENTS

I would like to thank Ivars Auzins, Keith Collyer, David Gelperin, and Hiam Kilov who reviewed this column and provided useful suggestions. Naturally, any remaining defects are my own.

REFERENCES

Further information on requirements specification can be found in the following two sources:

[Fires2001] Donald Firesmith and Brian Henderson-Sellers: *The OPEN Process Framework*, Addison-Wesley-Longman, 2001.

[Fires2003] Donald Firesmith: OPEN Process Framework (OPF) Website, http://www.donald-firesmith.com.

[Gelpe2002] David Gelperin: Maybe We Shouldn't "Write" Requirements, StickyMinds.com, 7 October 2002, http://www.stickyminds.com/r.asp?F=W5936.

About the author

Donald Firesmith is President of Firesmith Consulting, which provides consulting and training in the development of software-intensive systems. He has worked exclusively with object technology since 1984 and has written 5 books on the subject. He is currently writing a book on requirements engineering. Most recently, he has developed a 1000+ page informational website on the OPEN Process Framework at www.donald-firesmith.com. He can be reached at donald_firesmith@hotmail.com.