embeddedworld2016
Exhibition&Conference
...it's a smarter world

Requirements in the Flow

Strategy for Keeping Specifications Alive with Enterprise Architect

Jon Kowal
DSPECIALISTS
Digitale Audio und Messsysteme GmbH
Berlin, Germany
jon.kowal@dspecialists.de

***Abstract*—Proper requirements management is a crucial success factor for product development, especially in the long run. This paper focuses on solving some of the practical issues of requirements management, demonstrating how Sparx Systems' Enterprise Architect and its extension facilities may be used in a model based approach to master the task of maintaining usable specification documents throughout the entire product lifecycle. On top of the benefits gained from a model based approach, the quick and repeated production of documents, which are well readable by all stakeholders, is highlighted as a key ingredient to successful requirements management.**

Keywords—Requirements Management; UML; Enterprise Architect

I. INTRODUCTION

Requirements management has been a well researched topic for many years, if not decades, but practical experience shows that it is still a problem to be solved in many projects, teams and enterprises.

By now most people understand that requirements management is not a one-time act of requirements definition but rather the ongoing process of managing requirement changes which keeps affecting the product development and requires good integration into all phases of the product lifecycle. Many projects fail to keep that process running which results in outdated specification which is quickly not much more worth than having no specification at all.

After providing an understanding of requirements specification some key challenges to requirements management will be attacked in this paper through an integrated approach to requirements and design specification using Sparx System's Enterprise Architect as a platform to create model based specification documents.

Many tools exist to support the integration of requirements management and system design and some may even be more suited than Enterprise Architect. That said, it is beyond the scope of this paper to start a discussion on the pros and cons of different tools or praise Enterprise Architect as the ultimate solution. It is a platform, able to support the approach presented herein, not more and not less.

II. DEFINITIONS

A. *Requirements Specification*

A requirements specification is a set of written requirements which describe the functional and nonfunctional aspects of a system or system component.

A requirements specification is often requested as an input document to a specific design phase in the development process of the system to be described.

Requirements are usually the result of a requirements analysis in a specific phase of the underlying development model which defines the scope of the specified requirements.

Example: A system requirements specification is used to describe a system at large and may be followed by an architectural phase which defines components as well as requirements specifications targeted at the components. Some components may require software to be implemented which will need a software requirements specification, and so on.

Thus, depending on the abstraction levels of the system design, multiple requirements specifications with different scope may result from an iterative specification process, which are linked by design decisions made in the intermediate architectural design phases.

B. *Design Specification*

"Design is the general arrangement of the different parts of something that is made, such as a building, book, machine, etc." [1]

The design specification is the document or set of documents or artifacts which describe the design of a system and the decisions it's based on.

Despite not strictly considered part of the requirements definition phase, design may lead to new requirements to be defined for (a) the system components which are a result of the design process—as described above—and (b) the system itself, as a lack of proper requirements definition is often noticed within the design phase.

It is necessary to recognize the role of the design specification as part of the requirements management process as will be continually emphasized throughout this paper.

III. Requirements Management Challenges

A. Overview

There are certainly more challenges to requirements management than this paper will be able to discuss. This section therefor focuses on a few aspects which have been identified by the author as critical and difficult to master in combination:

- Traceability
- Change Management
- Keeping it simple for Editors
- Keeping it simple for Non-Editors
- Involving Third Parties

B. Traceability

Probably the key challenge to requirements management is to maintain traceability. The following questions have to be addressed:

- How has been taken care of each requirement in the design phase?
- What assumptions or external demand is a requirement based on, i.e. why does it exist?
- Is a design decision based on requirements or on assumptions made by the designer/developer?

While the first question usually comes to mind when thinking about traceability in a top-down development process, the latter are the ones which hunt developers in later phases of development or maintenance. They require bottom-up traceability.

In a perfect product development cycle, every design decision is explicitly linked to higher level requirements but this grade of completeness is rarely ever reached. It is simply too expensive to drag every design decision to a high level requirements definition. Often it is even impossible because technical circumstances imply certain design patterns which don't strictly relate to requirements.

Never the less it is important to know which design decisions are based on requirements and which are not. Especially when design is to be changed it is crucial to know which parts of the design may be discarded or modified without affecting the satisfaction of specific requirements.

C. Change Management

There may be many different kinds of requirements and only one thing all of them have in common: they are subject to change. If the change management has not been considered thoroughly when setting up the requirements management process it can quickly get overwhelming and costly to handle, especially with traceability in mind. That often leads to failure of the entire specification process.

D. Keeping it simple for Editors

Requirements management can only be kept within a reasonable budget if it is easy for editors to add, remove or change requirements. As part of the change management process this involves not only the editors of the requirements specification but also those of the depending design specification.

Managing all dependencies of requirements can get tedious, so tools should be used to support the work of the editors. That in turn requires proper training of the editors in using the tools.

E. Keeping it simple for Non-Editors

Non-editors are stakeholders which depend on the specification (requirements or design) but don't intend to edit it. Their ease-of-use requirements are different from those of the editors and they may be used to use different tools than the editors—or no tools at all—to view a specification. Non-editors often don't spend nearly as much time with the specification as editors and they need a way to pull out the needed information without having to know the specialties of the editing process or tools involved.

Being also affected by changes to the specification, non-editors need an easy way to tell changes from one version of the specification to another and which parts of the specification are to be approved and which are still under construction.

F. Involving Third Parties

Third parties are non-editors which are only loosely bound to the developing organization but also require access to the specification, e.g. customers or external appraisers. They usually don't have access to or are familiar with the specific tools used in the development process.

IV. Solution: Model Based Specification Documents

A. Model based Specification

As has been pointed out already, requirements management cannot be done isolated from the design process. That raises questions on how to keep track of all the dependencies between requirements and design specification.

To keep it simple for the editor, an easy and robust mechanism to define dependencies is needed. An object based approach would be ideal where specification objects (requirements, design decisions, etc.) can be defined and easily be associated with each other.

A widely known approach to declare objects and express their relationships is using the Unified Modeling Language (UML) and based on it the Systems Modeling Language (SysML) which is tailored specifically for the use in systems engineering. Other than UML, SysML allows the definition of dedicated requirement objects, relationships and diagrams, which gives it a more native feel when dealing with requirements. But on an abstract level simple UML objects are just as good to fulfill the task.

Another benefit of using UML (or SysML[1]) to model requirements can be the easy integration with a design process which is often already done in—or at least supported by—UML. One unified model can be used to define requirements and the resulting design. This does not even take into account the benefits which may be drawn from using code generation or simulation in model driven development.

B. Documents based Specification

A downside to using a model based approach to specification is that the specification feels more like a database than a document.

Experience shows: most people know how to read a document, much fewer know their way around a database. Proper written documents provide a broad scope while guiding the reader through the theme, implying what should be read first and what later. Documents are easily exchanged and independent of specific tools, or at least can be reduced to require a very common set of tools.

Experience also shows: It is very difficult to generate a proper written document from a collection of database items and this is the real dilemma when dealing with model based specification. While most tools provide some way or another to generate documents from the model, those documents are not necessarily useful to anyone. Caught up by the detail of the entities and relations of the model, editors easily forget to put them into context. Having defined a rather loose set of model entities is not sufficient to print an understandable specification document which serves as a guide to the reader.

To be able to serve all readers which are not inherently familiar with the model, it is important to organize the model entities in a way they'd be organized in a document. Textual descriptions need to be aware of which document and maybe even which section of the document they are intended to appear in.

Focusing on documents when editing the model helps to later generate documents which present themselves to non-editors like natural written documents. The underlying model remains hidden in the final documents and is usually not of interest to non-editors.

C. Quick Iterations

"Release early, release often" is a philosophy known in software development which makes just as much sense when it comes to specification. Especially when developing a specification in collaboration with one or more third parties, frequent exchange of the current state of a specification can help tackling misunderstandings before drifting off too far in a wrong direction.

In traditional documents based specification, quick iterations are difficult to realize as the various dependencies within large specification documents have to be reviewed and kept up to date before their release, which is often only done in long intervals. Model based specification solves that problem, as dependencies are always at hand and easily updateable or can at least be marked as outdated.

In a model based specification it is easy to assign further attributes to specification objects to express which parts of the specification are to be discussed and which are already approved. This can significantly speed up the handling of large specification documents.

To support the frequent exchange of specification documents, those documents have to be generated directly from the model without any or with only little manual steps involved. That is a matter of proper tool support.

D. Tool Support

The approach of model based specification documents cannot be realized efficiently without the help of supporting tools. The following is required:

- Editor to model requirements and design;
- Document generator to produce deliverable documents from the model.

[3] contains a long list of UML tools, some of which might be suited to do the job. This paper will focus on using Sparx Systems Enterprise Architect, basically because that's the UML tool used at *DSPECIALISTS*, where the author is employed, and because it provides all facilities needed to realize the described approach.

V. Enterprise Architect

A. Overview

Enterprise Architect (EA) is a modeling and design tool by Sparx Systems based on UML, SysML and other standards. It supports the specification of requirements and provides specialized tools to support requirements traceability.

Next to being a visual UML editor, Enterprise Architect is also a platform, providing a rich set of modeling tools on one hand and customization and extension through scripting and plugins on the other hand. Those customization possibilities are very useful as they allow us to add missing pieces to the program to achieve perfect specification documents.

B. Challenges

Being an open platform suited for any development process also means, Enterprise Architect is not providing strict guidance for any specific process. Though there are some model templates for different modeling patterns, they are not really helpful in our case. It is necessary to define an organizational strategy and stick to it when editing, in order to be able to collaborate without ending up in chaos.

[6] describes in detail how requirements can be specified and traced in EA as well as the specific support of the requirements management processes, including CSV import, reviewing, auditing, baselines, testing, and much more. Knowledge of those details is of value when realizing the strategy described in this paper, but will not be elaborated further as it falls in the category of training which should be completed by any editor attempting to use Enterprise Architect for requirements management.

[1] For the sake of simplicity the following text will stick to using UML as it is sufficient to express requirement and design dependencies and is more widely known and implemented in various tools.

Beyond [6], the following questions remain and need to be answered to define a strategy to create model based specification documents:

- How to declare/separate different documents within the model?
- How to include document content that is not strictly part of the model, such as an introductory chapter?
- How to declare which model content is to appear in the document?
- How to realize sectioning, numbering and inner document references?
- How to realize document versioning?
- How to realize model document dependencies and external references?

The following sections present a specific strategy on documents modeling in EA which attempts to provide answers to the questions above.

C. *Document based Model Structure*

"Packages provide the main generic structuring and organizing capability of UML. A Package is a namespace for its members, […]". [4] The semantics of packages correlate with those of "sections" in documents or folders in filing. Therefor, to organize documents in a model, packages can be used to separate document content sections as well as multiple documents.

Fig. 1 shows an example screenshot of the EA project browser, which is used to view and edit the model structure. It illustrates the definition of multiple documents below the package "Document Contents". The screenshot also shows that it is difficult to tell, which packages are document root nodes and which are document sections. Stereotypes, tagged values[2] or naming conventions may be used to diffentiate the different package types. Further document declaration will be done in the Document Directory as described in the following section.

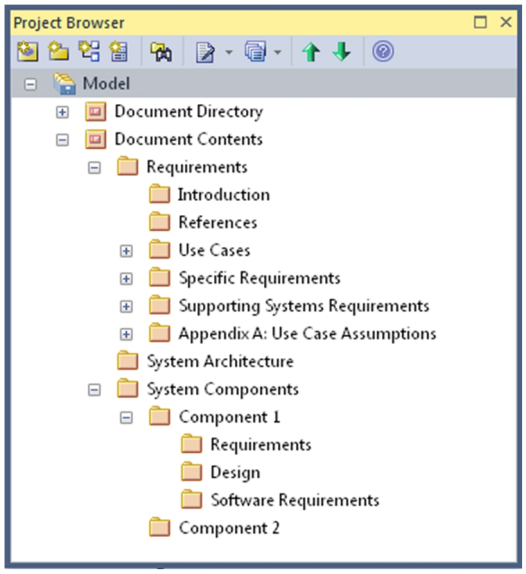

Fig. 1 Document Structure in Enterprise Architect Project Browser

Model elements such as use cases or requirements are placed within the appropriate section packages. Each element should be understood as a section or paragraph in the resulting document. Package names or element names serve as section headings. Tagged values can be used per element to tell the document generator if nested elements should appear as nested sections in the document or if they are to appear flat underneath each other. Generally, any element specific layouting option may be specified using tagged values and the document generator should be configured or extended to recognize those.

Enterprise Architect provides rich text notes fields for all elements and element dependencies which should be used to describe the elements as would be done in a textual document. Hyperlinks in the description can be used to insert references to other elements which are mentioned in the text or are related. Those hyperlinks will be resolved as cross referencing footnotes during document generation as is described in the section about *EA2Latex* below.

D. *Document Directory*

A package "Document Directory", kept separate from the specification content, serves as a central place to manage the model documents as well as externally referenced documents. EA provides specific model elements for that purpose:

- *Model Documents* are used to declare virtual documents based on packages in the model.
- *Document Artifacts* may be used to create links to external documents.

Both element types are intended to be used by the Rich Text Format (RTF) documentation generator built into EA, but they behave just like any other UML element, which allows us to create dependency connections to express which documents reference each other as depicted in Fig. 2. Those dependencies can be used by the document generator to fill in the references sections of the documents or to textually describe cross document references.

The Document Directory is structured with packages which represent the intended documentation file structure as to be delivered to the stakeholders. If document generation works well, the entire document tree can then be generated and packaged, archived, or revisioned on the fly based on the Document Directory structure.

E. *Document Generation*

Unfortunately, as of version 12.1, the documentation generator build into Enterprise Architect is not sufficient to generate deliverable specification documents in the sense defined above. There are several configuration options in the RTF generator as

[2] "Tagged Values are a convenient way of adding additional information to an element, beyond what is directly supported by UML." That definition used in [7] is deprecated as of UML 1.4, but the intended extensibility of UML elements may still be achieved using the new understanding of tagged values as stereotype properties. [8]

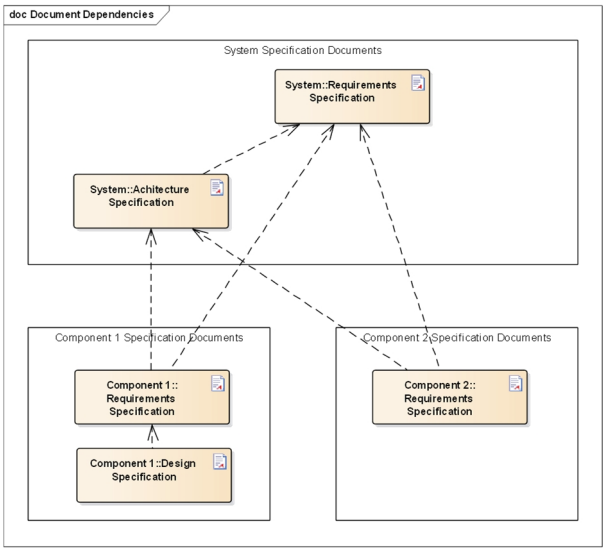

Fig. 2 Document relationships are expressed using standard UML techniques

of which model contents are to be selected for a specific model document and how they are to be laid out but the output still looks too much like data dumped onto a sheet of paper, missing too many features which are needed for a readable specification document. Just to name a few:

- Hyperlinks from within element notes are rendered as such and are useless in printed documents. There is no readable cross referencing such as “see section x.y”.
- There is no element specific formatting (e.g. based on tagged values), which we would need to treat complex objects as numbered sections and simple objects of the same type as paragraphs. That can lead to an extreme depth in section numbering, depending on the complexity of the model. Templates can only be customized based on element types.
- There are no cross document references. Hyperlinks to elements which are not part of the current document will just break. Cross document references are extremely useful to maintain traceability throughout multiple specification documents.
- To turn an EA generated document into a deliverable specification document requires too much manual tweaking afterwards (e.g. formatting, versioning, …), which defeats our goal of keeping it simple for the editor and enabling quick document releases.

The document generation feature in EA is more suited to generate smaller reports, displaying a filtered view on the model, e.g.: Create a list of all elements with status ‘approved’; which is also useful but not quite what we are looking for.

At this point the extensibility features of EA come in very handy. EA provides a plugin API with an easy to use interface to read and modify the entire model data from a custom provided plugin binary. There are some documentation plugins available for EA, already, such as [5], but those did not satisfy the needs discussed above at the time the issue was brought up at *DSPECIALISTS*. So *EA2Latex* was developed.

VI. EA2LATEX

A. Overview

EA2Latex is a plugin for Enterprise Architect which generates a TeX document from the UML model. It was created at *DSPECIALISTS* specifically to satisfy the in-house needs to generate deliverable specification documents from UML models.

EA2Latex uses LaTeX templates to format documents based on the model element types providing element specific template parameterization based on tagged values, stereotypes, and other element attributes. This reduces the plugin itself to create rather generic TeX code containing all the elements and their parameters, leaving the specific formatting to the LaTeX templates, which may differ depending on project or document type.

EA2Latex supports the entire documentation workflow by allowing the creation of draft documents from any model package (i.e. document section), providing integration of subversion for document management, and including versioning and referencing of specific document revisions. Fig. 3 and Fig. 4 show two states of the EA2Latex GUI Dialog to illustrate the workflow.

B. Features

The following is just a short list of the most important *EA2Latex* features, without going too much into detail as that would exceed the scope of this paper:

- Configuration of document properties (e.g. title, filename, table of contents depth, etc.) based on Model

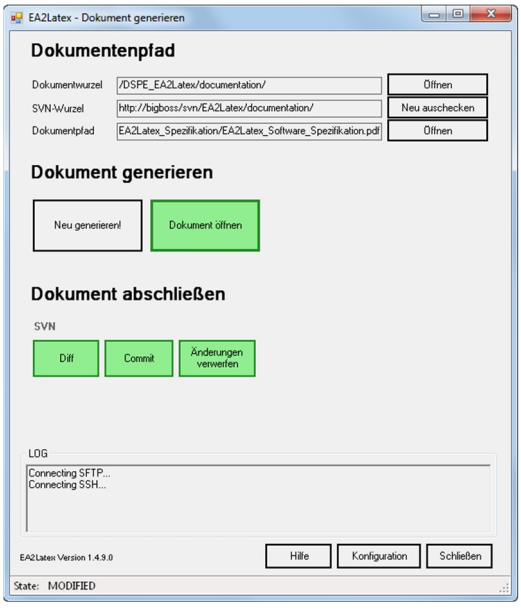

Fig. 3 *EA2Latex* GUI state after document generation, allowing to open, diff, commit, discard, or regenerate the document. Green color indicates next plausible steps.

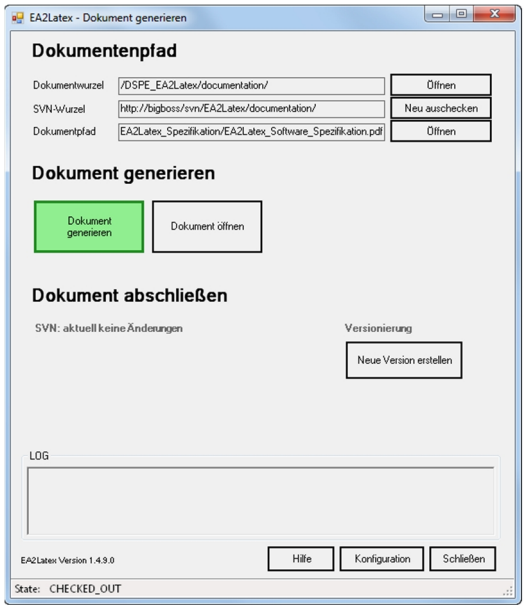

Fig. 4 *EA2Latex* GUI state after SVN commit, allowing to open or regenerate the document, or tag a new document version. Green color indicates next plausible steps.

Document attributes and tagged values;

- Optional integration of one SVN repository per Document Directory;
- Simple but powerful GUI, guiding through the document creation process, including document versioning and tagging;
- Support of inner document references using footnotes based on hyperlinks in element notes. (e.g. “see section x.y, Element A, page z”);
- Support of cross document references based on document dependencies and hyperlinks in element notes. (e.g. “see section x.y, Element A, page z, document B”);
- Support of all EA diagram types and template specific positioning of diagrams within sections or pages.
- Automatic creation of dependency connections in the model for elements which are hyperlinking each other to achieve traceability for hyperlinks;
- Hide certain packages or elements and their children from the document based on tagged values in the model;
- Definition of hierarchical or flat section structure on per element basis (default: hierarchical);
- Use of LaTeX as flexible template engine, including all of its benefits such as automatic section numbering, cross referencing, floating figures, tables, indices and beautiful type setting;
- Server based processing of LaTeX templates to reduce client side configuration overhead;
- Provide quick links to often used plugin features, e.g. one context menu entry for Model Document objects to instantly produce a deliverable PDF document;
- Output of high quality PDF documents, including hyperlinked cross references—even across multiple documents—, vector graphics, and table of contents.

VII. Conclusion

The presented method of *model based specification documents* provides well readable specification documents while maintaining traceability and easy modifiability in the underlying model. Those are the basic ingredients needed for successful requirements management.

Enterprise Architect proves to support the model based editing process greatly while leaving the structural organization of the model to the editor, allowing any individual specification process. Furthermore it provides the infrastructure to add the salt needed to fulfill custom needs. With the *EA2Latex* plugin it gained the capabilities needed for project specific document generation and management at *DSPECIALISTS*.

References

[1] Joanna Turnbull , “Oxford Advanced Learner’s Dictionary of Current English”, Oxford University Press, 2011

[2] Sparx Systems, Enterprise Architect, 2016, Available at: http://www.sparxsystems.com.au/

[3] Various Authors, “List of Unified Modeling Language tools”, 2016, http://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools

[4] OMG Unified Modeling Language (OMG UML), V2.5, March 2015, p.239

[5] eaDocX, 2016, available at: http://www.eadocx.com/

[6] Sparx Systems, “Requirements Management with Enterprise Architect”, 2014, http://www.sparxsystems.com.au/downloads/whitepapers/Requirements_Management_in_Enterprise_Architect.pdf

[7] Sparx Systems, “Tagged Values”, Enterprise Architect User Guide v12, http://www.sparxsystems.com/enterprise_architect_user_guide/12/modeling_basics/thetaggedvaluestab.html

[8] OMG Unified Modeling Language: Infrastructure, V2.0, March 2006, p.196