Wiki Based Requirements Documentation of Generic Software Products

Clara Silveira [1], João Pascoal Faria [2,3], Ademar Aguiar [2,3], Raul Vidal [2]
[1]Escola Superior Tecnologia Gestão do Instituto Politécnico da Guarda, Portugal
[2] Faculdade de Engenharia da Universidade do Porto, Portugal
[3]INESC Porto, Portugal
{silveira, jpf, aaguiar, rmvidal}@fe.up.pt

Abstract

Organizations that develop generic software products, such as ERP or CRM products, and implement them in customers with varying needs, are faced with the problem of relating each customer requirements to the generic product requirements and characteristics, in a way that accelerates product implementation and supports product evolution decisions. To help attain these goals, we present a requirements documentation approach, comprising a documentation model and a XML and Wiki-based documentation infrastructure. The relationship above mentioned is established mainly via configuration parameters and associated configuration questions. Variability in the requirements and characteristics of the generic product is described based on the configuration parameters. The requirements of each implementation are described by answering the configuration questions and, when needed, by documenting variations to the base requirements and characteristics. Linking and viewing facilities support traceability analysis, instantiation of base requirements and characteristics for actual parameter values, and variability analysis among actual implementations.

1. Introduction

Generic software products, intended to support the business processes of organizations with varying needs, such as Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) systems, are characterized by high modularity and configurability. The customer can choose a subset of modules according to his needs, conveniences or budgets. Each module can be adapted to the customer's specific needs, by configuration (e.g. by setting configuration parameters) and, when needed, by customization (customer specific developments).

The process of adapting a generic software product to the specific needs of an individual customer, and integrating it in the customer environment, is sometimes called an *implementation* process, while the process of building the product is called a *development* process. Both the development and implementation processes comprise requirements engineering (RE) activities. RE at the product development level is concerned with capturing the common and variable requirements within the domain. RE at the product implementation level is concerned with checking if the product meets the customer needs, and determining the configurations and customizations required.

When the development and implementation processes are performed (or, at least, managed) by the same organization, product implementation can be accelerated and decisions about product evolution can be better supported, if RE activities and artefacts at both the product development and implementation levels are closely related. Configuration choices determined during the RE activities of an implementation process are necessarily related to the product scope and variability determined during the RE activities of the development process. Such relationship can be made explicit if the product variability is described, from the beginning, as a function of configuration parameters used in the implementation process. This requires that the configuration parameters are identified early, during the RE activities of the development process.

The main contribution of this paper is a requirements documentation approach, targeted for organizations that want to manage in an integrated and agile way the software requirements in the development of generic software products and their implementation in customers with varying needs. The approach comprises a requirements documentation model and a collaborative XML and Wiki-based infrastructure, supporting:

(1) the structured documentation of requirements and characteristics of the generic product, based on templates, with variability defined as a function of configuration parameters;
(2) the definition of the configuration parameters and associated configuration questions;
(3) the documentation of the requirements of each implementation, firstly, by answering the configuration questions (black-box reuse), and, secondly, by documenting variations and extensions to the base requirements and characteristics (white-box reuse);
(4) the integration of the previous artefacts, supporting navigation, traceability, instantiation of the base requirements and characteristics for actual parameter values, and the analysis of variability among actual implementations.

This approach is part of o more complete RE process, for the development and implementation of generic software products, to be described in [25].

The rest of the paper is organized as follows: section 2 describes the requirements documentation model proposed, independently of concrete tools and platforms, section 3 describes the tool support, section 4 discusses related work, and section 5 presents some conclusions and areas that deserve future work.

To illustrate the approach, a running example, related to the management of marketing campaigns in a CRM system, will be used along the paper. This example comes from a re-documentation experiment (of a real world product) that was conducted to validate the approach.

2. Requirements documentation model

Figure 1 presents a top-level view of the requirements documentation model proposed to support, in an integrated way, the main RE concerns at the product development and implementation levels.

A major concern at the product development level is to capture the common and variable requirements within the domain. In order to allow explicitly relating the configuration choices determined at implementation time with the product scope and variability defined at development time, we propose that variability is described, from the beginning, as a function of configuration parameters used in the implementation process. Starting from a list of high-level requirements (P1), including variability requirements (configurability requirements for black-box reuse and customizability requirements for white-box reuse), detailed product requirements and characteristics can then be developed and described (e.g. through use case and entity models) (P3), expressing variability as a function of configuration parameters defined at the same time (P2).

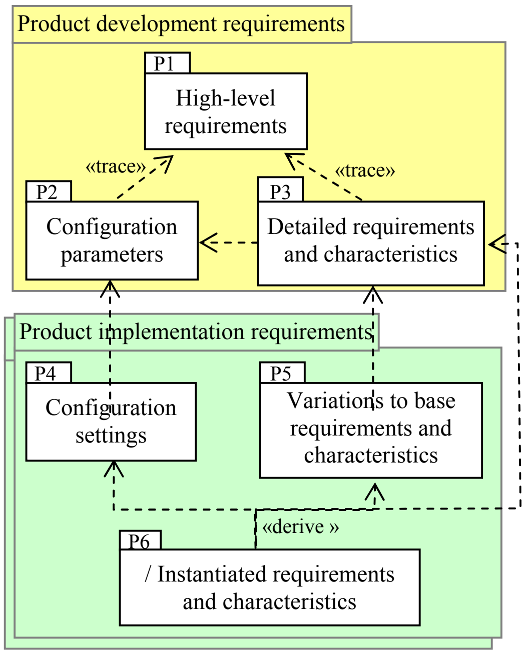

Product development requirements
P1
High-level requirements
«trace»
«trace»
P2
Configuration parameters
P3
Detailed requirements and characteristics
Product implementation requirements
P4
Configuration settings
P5
Variations to base requirements and characteristics
«derive »
P6
/ Instantiated requirements and characteristics

Figure 1. UML package diagram [12] showing the main requirements documentation packages and dependencies.

At the product implementation level, main concerns are to check if the product meets the customer needs, and to determine the configuration settings and customizations required. If a configuration question is prepared for each configuration parameter, the configuration settings required can be conveniently obtained by answering a configuration questionnaire (P4). Usually, in order to meet customer needs, customizations (custom specific developments) are also required. In some cases, the product itself has to be enhanced. In general, the customer needs that cannot be satisfied by mere configuration, can be satisfied by adding, removing or modifying features from the base product, and can be described as variants (additions, removals and modifications) to the base product requirements and characteristics (P5).

Identifier	Configuration parameter	Configuration question	Type of answer	Options
QCRM1	Use segments	Do you want to be able to use an existing segment	single choice	Yes, No

		as the target of a marketing campaign?		
QCRM2	Add contacts manually to marketing campaigns	Do you want to be able to add contacts manually as the target of a marketing campaign?	single choice	Yes, No
QCRM3	Marketing channels	From the following list, what marketing channels do you want to use: e-mail, letter or sms?	multiple choice	e-mail, letter, sms
QCRM4	Additional contact attributes	What additional attributes do you want to use to describe a Contact?	user defined list	name, description
QCRM5	Campaign statuses	What are the possible statuses of a marketing campaign?	user defined list	name, description

Figure 2. Example definition of configuration parameters.

By applying the configuration settings and variants to the base product requirements and characteristics, a detailed, self-contained, description of the product requirements and characteristics for a specific implementation can be automatically derived (P6). Information about actual product variability (not shown in Figure 1), can also be automatically derived.

The next sections describe with further detail the packages shown in Figure 1, and give some concrete examples. The definition of high-level requirements is not addressed, because it follows a straightforward structure.

2.1. Definition of configuration parameters

Figure 2 illustrates the definition of some configuration parameters and associated configuration questions for the running example (management of marketing campaigns). Each configuration parameter has a name, an identifier, a corresponding question (to be answered in each product implementation), a type (single choice, multiple choice, user defined list, number, date, text, etc.) and a list of options (interpreted according to the type).

The configuration questions are to be answered in the initial phases of each implementation process. The configuration parameters are used in the description of the product requirements and characteristics (and defined at the same time), to express variability.

Supporting tools should allow the definition of configuration parameters based on templates or forms, and should automatically provide navigation links between the definition of each configuration parameter and all the related artifacts, for forward and backward traceability. To answer the configuration questions, an appropriate questionnaire should be presented automatically to the user, based on the definitions provided.

Usually, configuration parameters are not fully independent. Dependencies among configuration parameters can be defined as configuration constraints. Since each constraint relates multiple parameters, constraints are better defined separately. Figure 3 illustrates the definition of a configuration constraint for the running example. Both a human readable and a machine readable description are provided. The machine readable descriptions should be used to validate or guide the user answers to the configuration questions.

Identifier	Description	Formula
CCCRM1	QCRM1 and QCRM2 cannot be both 'No'	QCRM1 = 'Yes' or QCRM2 = 'Yes'

Figure 3. Example definition of a configuration constraint.

2.2. Parameterized descriptions of product requirements and characteristics

2.2.1. Parameterization mechanisms. In our approach, the detailed descriptions of product requirements and characteristics are parameterized by the values of the configuration parameters. Two main parameterization mechanisms are possible:

- substitution - a parameter reference in the middle of the documentation is substituted by its actual value in each product implementation;
- conditional inclusion (optional feature) - a part of the documentation (describing some optional feature) is tagged with a condition that references one or more configuration parameters; in each product implementation, that part of the documentation is excluded if the condition is false.

2.2.2. Use case descriptions. In general, functional requirements can be conveniently described with use cases [3]. In the case of a generic software product, the concrete behavior of a use case may vary from implementation to implementation. In our approach, those variations should be described together (to promote understanding), based primarily on the values of the configuration parameters. Figure 4 presents an example template to describe a use case at the product level, with variants based on the values of the configuration parameters. Such use cases can be called *parameterized use cases*, although the parameters are not defined locally as in [2], but globally. Our full approach accommodates both global configuration parameters and local parameters, but only global ones are of concern here.

Element	Description
Identifier	Unique use case identifier (e.g. module identifier + sequence number).
Name	Use case name (e.g. verb + nom).
Brief description	A short description of the main purpose of the use case.
Configuration parameters	List of global configuration parameters that affect the behavior and, in general, any part of the description of this use case (by substitution or conditional inclusion).
Actors	List of primary and secondary actors that interact with the system in this use case.
Inclusions	List of use cases that are included by this use case.
Extensions	List of use cases that extend this use case, together with the conditions under which each extension applies.
Basic and alternative flows of events	Description of the basic flow of events and the alternative flows of events, each comprising a sequence of steps.
Pre and post-conditions	Lists of pre-conditions and post-conditions.
Entities	List of domain entities manipulated by this use case, with the manipulation modes (Create, Retrieve, Update, Delete).

Figure 4. Example template to describe a parameterized use case. Highlighted elements can be conditional, based on the configuration parameters.

Figure 5 shows a use case diagram for the running example and Figure 6 shows a possible description of the use case "Create Marketing Campaign".

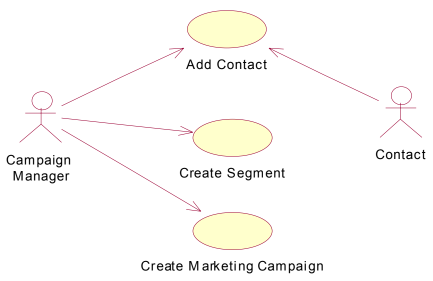


Figure 5. Use case diagram for the running example.

Element	Description
Identifier	UCCRM1
Name	Create Marketing Campaign
Brief description	This use case allows the definition of a marketing campaign and its target segment or set of customers.
Configuration parameters	QCRM1, QCRM2, QCRM3
Actors	Campaign Manager
Inclusions	None
Extensions	None
Basic and alternative flows of events	Basic flow: 1. The user inputs descriptive data of the campaign. 2. The user selects the campaign target, by one of the following methods: 2.1. [if QCRM1='Yes'] The user selects the option "Use existing Segment" and selects an existing segment. 2.2. [if QCRM2='Yes'] The user selects the option "Add contacts manually" and selects the contacts to be added to the campaign. 3. The user selects the marketing channel from the available list (QCRM3). 4. The user confirms the input data. 5. The system saves the campaign data.
Pre and post-conditions	Pre-conditions: 1. [if QCRM1='Yes'] The target segment was created with "Create Segment". 2. [if QCRM2='Yes'] The target contacts were created with "Add Contact". Post-conditions: 1. A new campaign is registered in the system.
Entities	Campaign (Create), [if QCRM1='Yes'] Segment (Retrieve), Contact (Retrieve), Channel (Retrieve)

Figure 6. Example use case description.

2.2.3. Domain entity descriptions. In general, information requirements can be captured in a domain model comprising domain entities and their attributes, relationships and constraints [5]. In the case of a generic software product, the relevant elements of the domain model may vary from implementation to implementation. In our approach, those variations should be described together (to promote understanding), based primarily on the values of the configuration parameters.

Figure 7 shows a simplified class diagram for the running example, and Figure 8 shows a possible description of the entity "Campaign", with variants based on the values of configuration parameters.

Constraints on attribute values are also defined. Any supertype, attribute or constraint can be conditional, based on the values of the configuration parameters.

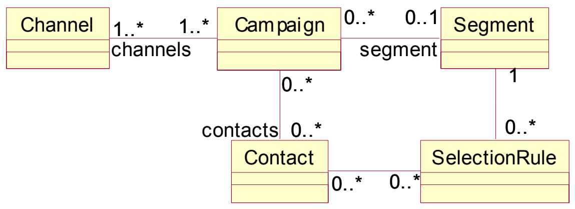


Figure 7. Class diagram showing domain entities and relationships for the running example.

Element	Description
Identifier	ECRM3
Name	Campaign
Brief description	Group of marketing actions directed towards contacts (customers) through communication channels.
Configuration parameters	QCRM1, QCRM2, QCRM3, QCRM5
Attributes (and associations)	**Condition** / **Name** / **Description**
	— / name / —
	— / description / —
	— / startDate / —
	— / endDate / —
	— / status / —
	QCRM1 = 'Yes' / segment / target segment
	QCRM2 = 'Yes' / contacts / target contacts
	— / channels / channels used (subset of QCRM3)
	— / ... / —
Constraints	1. EndDate > StartDate 2. status is in QCRM5

Figure 8 Example entity description.

2.3. Definition of configuration settings

Figure 9 illustrates a possible choice of configuration settings for the running example.

In a practical implementation, the configuration settings are obtained by answering a configuration questionnaire produced automatically based on the definitions of configuration parameters and constraints.

Identifier	Question	Answer
QCRM1	Do you want to be able to use an existing segment as the target of a marketing campaign?	No
QCRM2	Do you want to be able to add contacts manually to a marketing campaign?	Yes
QCRM3	From the following list, what marketing channels do you want to use: e-mail, letter, sms?	e-mail, sms
QCRM4	What additional attributes do you want to use to describe a Contact?	nationality language
QCRM5	What are the possible statuses of a marketing campaign?	plan, run, closed

Figure 9 Example configuration settings.

2.4. Definition of variants to the base product requirements and characteristics

Customer needs that cannot be satisfied by mere configuration, can be satisfied by adding, removing or modifying features from the base product, and can be described as variants (additions, removals and modifications) to the base product requirements and characteristics.

Additions and modifications can be described using the same structure as the base requirements and characteristics, with special tags to indicate where (in the base documentation) they should be inserted or what base element they should replace, respectively.

Removals from the base requirements and characteristics can be described by indicating the base element to be removed.

Figure 10 illustrates the definition of a variant for a hypothetical implementation in the running example. In this example, a post-condition is added.

Element	Description
Identifier	UCCRM1
Pre and post-conditions	Post-conditions: [Add] 2. An e-mail notification is sent to the marketing department head.

Figure 10 Example description of a variant to base requirements and characteristics.

2.5. Deriving instantiated descriptions of product requirements and characteristics

Figure 11 illustrates the result of applying the configuration settings of Figure 9 and the variants of Figure 10 to the base product requirements and characteristics described in Figure 6.

Element	**Description**
Identifier	UCCRM1
Name	Create Marketing Campaign
Brief description	This use case allows the definition of a marketing campaign and its target *segment or* set of customers.
Actors	Campaign Manager
Inclusions	None
Extensions	None
Basic and alternative flows of events	Basic flow: 1. The user inputs descriptive data of the campaign. 2. The user selects the campaign target, by *one of the following methods*: *2.2.* The user selects the option "Add contacts manually" and selects the contacts to be added to the campaign. 3. The user selects the marketing channel from the available list (e-mail, fax, sms). 4. The user confirms the input data. 5. The system saves the campaign data.
Pre and post-conditions	Pre-conditions: *2.* The target contacts were created with "Add Contact". Post-conditions: 1. A new campaign is registered in the system. *2.* An e-mail notification is sent to the marketing department head.
Entities	Campaign (Create), Contact (Retrieve), Channel (Retrieve)

Figure 11 Derived description of an instantiated use case for the running example.

2.6. Deriving actual variability information

2.6.1. Actual variability information at the configuration parameter/question level. This is a derived view that shows, for each configuration parameter, the actual values and frequencies that occur in existing implementations. This information is useful, for example, for product maintenance and evolution.

2.6.2. Actual variability information at the detailed requirements and characteristics level. This is a derived view that shows the detailed descriptions of product requirements and characteristics, with the optional parts rendered or annotated according to their frequency of inclusion in actual implementations (see dimmed elements in Figure 6). This information is also useful for product maintenance and evolution.

3. Tool support

To support requirements documentation and management in the development and implementation of generic software products, according to the documentation model and features presented in section 2, it was used an existing XML and Wiki-based software documentation tool - XSDoc [18].

In the next sections, after brief overviews of tools used in traditional requirements documentation approaches and of the XSDoc tool, we explain how the most important features of our requirements documentation model are supported by the XSDoc tool with appropriate configurations and extensions, some of which are the subject of ongoing work.

3.1. Tools used in traditional requirements documentation approaches

Requirements are usually documented with the help of a combination of tools: requirements management tools, like IBM Rational RequisitePro, to identify and describe lists of requirements; modeling tools, like IBM Rational Rose, to model requirements by UML or other diagrams; and word processing tools, like Microsoft Word, to provide supplementary descriptions and compose requirements documents that can delivered to stakeholders.

Requirements management tools are useful to gather requirements, control changes and versions, track status and maintain traceability links. Many requirements management tools integrate with other software engineering tools, including modeling tools and word processing tools. Overviews and comparative analysis of requirements management tools can be found for example in [13, 14, 16, 17].

Requirements management tools can be classified as database or document centric [17, 22]. Database-centric tools store all requirements (including their textual descriptions), attributes and traceability information in a database. Requirements can be imported from various sources, but they then reside in the database. Some tools support links to external files with supplementary information. Requirements documents are essentially reports from the database. By contrast, a document-centric tool treats a document created using a word-processing tool as the primary container for requirements. Selected elements of the document are stored as discrete requirements in the database and described with additional attributes and traceability information. For example, DOORS can be classified as database centric, while RequisitePro can be classified as document centric [22].

3.2. Collaborative approaches and the XSDoc tool

Recently, software documentation approaches based on WikiWikiWebs [19] are gaining popularity. These approaches privilege collaborative working, ease of change, accessibility, open formats and tools, and reduced up-front investments, and support some of the principals of agile methods for software documentation [23, 24] and requirements engineering [26, 27].

XSDoc [18] is an example of a software documentation infrastructure developed along these principles. XSDoc extends a traditional Wiki engine with several features to facilitate the edition, visualization, integration and validation of software documentation contents of different kinds (free text, XML documents, UML diagrams, source code, etc.). The automatic linking mechanism originally restricted to Wiki pages was enhanced to support also linking and inlining of source code fragments, UML diagrams, and structured contents, using simple naming conventions (e.g. prefixes, suffixes, and patterns). To enable content integration and extensibility, all contents are stored internally in XML format. For version control, XSDoc can access repositories of version control systems. New content types can be added using a plugin mechanism. XSDoc can be used standalone, in a web-browser, or integrated in an IDE such as Eclipse, via plugins. The XSDoc architecture is illustrated in Figure 12.

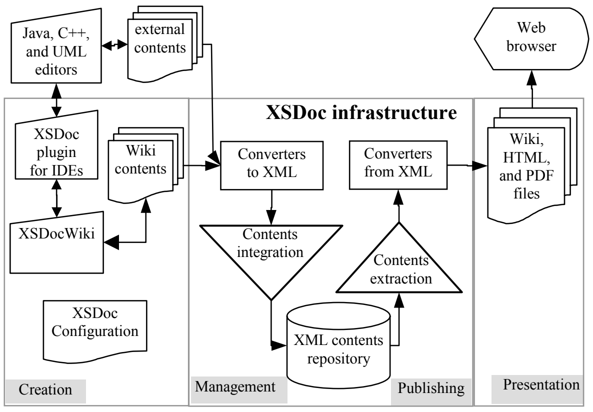


Figure 12 XSDoc architecture.

3.3. Templates

The requirements documentation model presented in section 2 defines templates for documenting use cases (Figure 4), entities, configuration parameters, configuration constraints, etc.

In XSDoc, a template can be defined as a combination of a document schema definition (XSD), a document formatter (XSL) and several exemplars (XML). Documents created based on a template are stored internally in XML format, and are rendered using the corresponding XSL formatter.

Naming conventions can be used to automatically associate Wiki documents, identified by Wiki names, with existing templates. For example, all Wiki names including "UseCase" can be associated with the template "UseCaseTemplate".

Template based documents are currently edited directly in XML. An open source solution is being integrated into XSDoc to support document edition by filling in a form that is created automatically based on the XSD definition.

A set of templates have been created to support the different types of documents described in section 2. These templates can be easily adapted, and other templates can be easily created.

3.4. Parameterized documents

XSDoc is being extended to support the definition of parameterized documents, with the parameterization mechanisms described in section 2.2.1 (substitution and conditional inclusion).

Conditional inclusion is supported at the element level. Any element in a XML document can be annotated with an attribute *condition* that defines a condition, based on the values of configuration parameters, that indicates if the element should be included or excluded.

Appendix A illustrates the definition of the XSD part of a template for documenting parameterized use cases (according to Figure 4). A XML representation of the use case description of Figure 5, not included for space limitation reasons, can be found in www.fe.up.pt/~jpf/research/AWRE05/UCCRM1.xml.

3.5. Delta documents

An open source solution is being integrated in XSDOC to support the definition of variants to base documents via *delta documents*, using an approach similar to the one presented in [20] and implemented in [21]. A delta document is a XML document that defines changes to a base XML document at the element level, as illustrated in Figure 13. The delta document follows the same structure as the base document, and the elements changed are annotated with a special attribute (*delta*) that specifies the type of change (*add*, *delete* or *replace*).

```
<UseCase>
  <identifier> UCCRM1</identifier>
  <post_conditions>
    <post_condition number="2" delta="add">An e-
mail notification is sent to the marketing department
head.</post_condition>
  </post_conditions>
</UseCase>
```

Figure 13. Definition of the variants of Figure 10 by a delta XML document.

Alternative approaches exist that define delta documents as sequences of delta operations (described as XML elements). We favor the approach illustrated above for readability reasons.

3.6. Bidirectional navigation links

Navigation links for backward traceability (navigate to referenced documents) are automatically provided via Wiki references, as in any Wiki tool.

Navigation links for forward traceability (navigate to referencing documents) are automatically computed by the document viewer (see section 3.8).

3.7. Dynamically derived templates

Configuration questionnaires will be supported by dynamically derived templates that are being introduced in XSDoc.

Based on the definition of the configuration parameters and questions (as the ones illustrated in Figure 2), a template will be dynamically created for the corresponding configuration questionnaire (as the one illustrated in Figure 9).

3.8. Advanced viewing facilities

XSDoc provides an advanced document viewer. Besides the identifier (Wiki name) of the document one wants to view, the document viewer is being extended to accept additional optional parameters to:

- identify delta documents and parameter settings to be applied to the base document;
- render conditional elements according to their frequency of inclusion among a range of parameter settings;
- show the list of documents that reference the current document.

These parameters can be set on a per call basis, or globally. For modularity reasons, these features are implemented via specific formatters or transformers that are applied in pipeline.

4. Related work

A main issue in the design of the requirements documentation model is how to model requirements variability. A similar issue also appears in the context of the development of software product lines, which has deserved a lot of research effort [10, 11].

The dominant approaches for modeling requirements variability in the context of software product lines are based on feature models. Feature models capture commonalties and differences of applications in a domain by means of feature trees with common, alternative and optional features. For example, the Feature-Oriented Reuse Method (FORM) [6] extends the well-known Feature Oriented Domain Analysis method [7] with a 4-layered feature model (capability, operating environment, domain technology, and implementation technique layers) that is used to develop reusable domain architectures and components.

Approaches also exist to model variability directly in more detailed requirements models, namely use case and other UML models. There are also proposals that combine feature and UML models. For example, the FeatuRSEB method [8] expands the use case driven RSEB method [9] with an explicit feature model to provide a feature index into common and variable use case, design and implementation elements.

A different approach is proposed in the Variation Point Model (VPM) [5]. Variation points are first defined in the requirements view (as high-level requirements for variability) and realized in design views (component, static and dynamic views) of the core assets. This approach allows modeling variability, not only in scenarios where variants are known in advance, but also in scenarios where reusers may create their unique variants.

The main differences of our approach when compared with the approaches described above are the following:

- requirements variability is modeled directly in the use case and entity models, and not in a separate feature model (in our case, the feature model is not needed for feature selection purposes, because feature selection is achieved by answering configuration questions);
- variability is defined as a function of the configuration parameters that are used later in product implementation (this allows instantiating the detailed descriptions of product requirements and characteristics for each implementation).

These differences are motivated by the specific context we address in this paper: the development of generic software products.

5. Conclusions and future work

It was presented a requirements documentation approach, targeted for organizations that want to manage in an integrated and agile way the software requirements in the development of generic software products (e.g. ERP or CRM products) and their implementation in customers with varying needs.

The main aims of the approach are to accelerate product implementation and to provide better support for decisions about product evolution.

The approach comprises a requirements documentation model and a XML and Wiki-based documentation infrastructure (XSDoc). Product requirements and characteristics, determined at development time, and customer specific requirements, determined at implementation time, are explicitly related via configuration parameters and associated configuration questions. Requirements are structured based on customizable templates. Variability is supported by parameterized documents and delta documents. Advanced linking and viewing facilities support traceability analysis, instantiation of base requirements and characteristics for individual implementations, and variability analysis among actual implementations.

We are currently improving the tool support, as explained in section 3, and plan to introduce and experiment the approach in several product oriented software development companies.

6. References

[1] A. Bertolino, A. Fantechi,S. Gnesi, G. Lami, and A. Maccari, “Use Case Description of Requirements for Product Lines”, *REPL’02*, September 2002, pp. 12-18.

[2] Alistair Cockburn, *Writing Effective Use Cases*, Addison-Wesley, 2001.

[3] Kurt Bittner and Ian Spence, *Use Case Modeling*, Addison-Wesley, 2003.

[4] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley Longman, Inc., 1999.

[5] D. Webber and H. Gomaa, “Modeling Variability in Software Product Lines with the Variation Point Model”, *Science of Computer Programming*, Vol. 53, Nº 3, 2004, pp. 305-331.

[6] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, “FORM: A feature-oriented reuse method with domain-specific reference architecture”, *Annals of Software Engineering*, Kluwer Academic Publishers, Dordrecht, Holland, Vol. 5, 1998, pp. 143–168.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[8] M. Griss, J. Favaro, and M. Alessandro, “Integrating Feature Modeling with RSEB”, *5th International Conference on Software Reuse (ICSR-5)*, Victoria, Canada, 1998.

[9] I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse: Architecture, Process, Organization for Business Success*, ACM Press, Addison-Wesley, 1997.

[10] M. Griss, “Product-Line Architectures”; *Component-Based Software Engineering: Putting the Pieces Together*; Heineman, G. T.; Councill, W. T. (Eds.), Addison-Wesley, 2001, pp. 405-419.

[11] R. R. Lutz, “Extending the Product Family Approach to Support Safe Reuse”, *Journal of Systems and Software*, Vol. 53, Nº 3, 2000, pp. 207-217.

[12] “UML 2.0 Superstructure Specification”, OMG, October 8, 2004, www.uml.org.

[13] "Requirements Management Tools Survey", International Council on System Engineering (Incose), www.paper-review.com/tools/rms/read.php.

[14] "Requirements Tools", Volere, www.volere.co.uk/tools.htm.

[15] M. Hoffmann, N. Kühn, M. Weber and M. Bittner, “Requirements for Requirements Management Tools”, *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE’04)*, 2004.

[16] R. Wieringa and C. Ebert, “RE’03: Practical Requirements Engineering Solutions”, *IEEE Software*, Vol. 21, Nº 2, 2004, pp. 16-18.

[17] K. Wiegers, “Automating Requirements Management”, Process Impact, 1999. www.processimpact.com/articles/rm_tools.pdf.

[18] Ademar Aguiar, Gabriel David, Manuel Padilha, "XSDoc: an Extensible Wiki-based Infrastructure for Framework Documentation", *VIII Jornadas de*

Ingeniería del Software y Bases de Datos (JISBD 2003), 2003, pp. 11-24.

[19] Ward Cunningham, "The original wiki front page", 1999, http://c2.com/cgi/wiki.

[20] Robin La Fontaine, "A Delta Format for XML: Identifying Changes in XML Files and Representing the Changes in XML", *XML Europe 2001*, 21-25 May 2001, Berlin, Germany.

[21] www.deltaxml.com

[22] Karl E. Wiegers, *Software Requirements*, 2nd Edition, Microsoft Press, 2003

[23] Scott Ambler, "Agile documentation", http://216.239.59.104/search?q=cache:gdjxR2caYOsJ:www.agilemodeling.com/essays/agileDocumentation.htm+agile+documentation&hl=en.

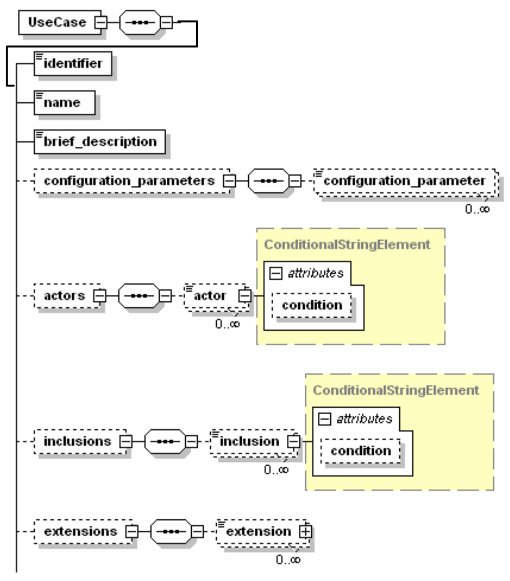
[24] Andreas Rueping, "Agile documentation", www.developerdotstar.com/mag/bookreviews/davis_agiledocumentation.html.

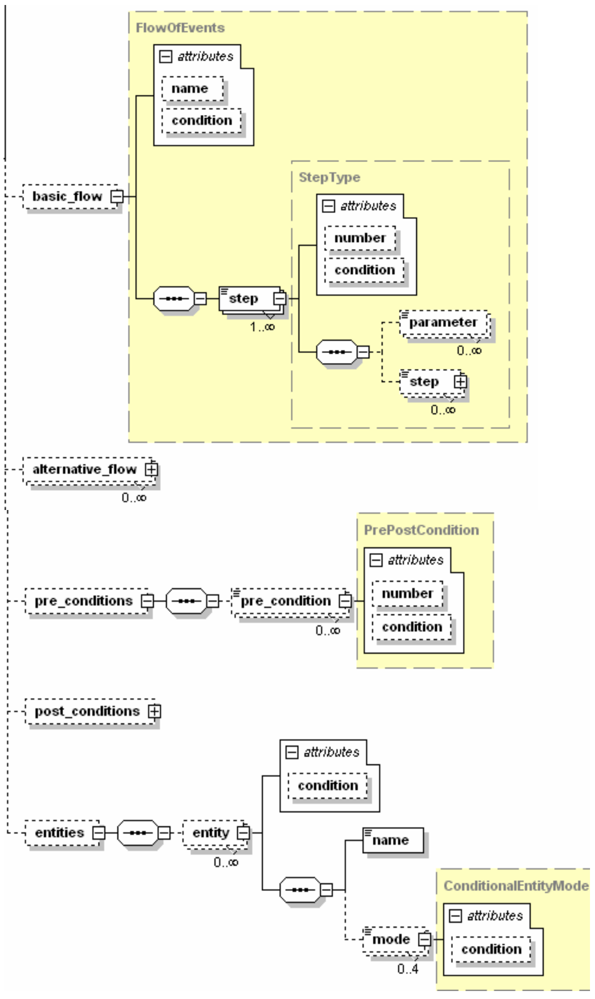
[25] Clara Silveira, *Reutilização de Requisitos no Desenvolvimento e Adaptação de Produtos de Software*, PhD thesis, FEUP, Porto, Portugal, 2005 (*to be published, in portuguese)*.

[26] A. Sillitti and G. Succi, "Requirements Engineering for Agile Methods", *Engineering and Managing Software Requirements*, A. Aurum and C. Wohlin, eds., Berlin, Springer, 2005, pp. 309-325.

[27] K. Kolehmainen, "Agile Requirements Engineering: Building tool support for XP", *VTT*, 2003, www.vtt.fi/moose/docs/agile_re.pdf.

Appendix A - Example definition of a use case template in XSD (schema view)


UseCase
identifier
name
brief_description
configuration_parameters
configuration_parameter
0..∞
ConditionalStringElement
attributes
condition
actors
actor
0..∞
ConditionalStringElement
attributes
condition
inclusions
inclusion
0..∞
extensions
extension
0..∞


FlowOfEvents
attributes
name
condition
basic_flow
StepType
attributes
number
condition
step
1..∞
parameter
0..∞
step
0..∞
alternative_flow
0..∞
PrePostCondition
attributes
number
condition
pre_conditions
pre_condition
0..∞
post_conditions
attributes
condition
entities
entity
0..∞
name
ConditionalEntityMode
attributes
mode
0..4
condition