Representation of User Stories in Descriptive Markup

Pankaj Kamthan

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
for the degree of
Doctor of Philosophy (Computer Science)
at
Concordia University
Montreal, Quebec, Canada

November 2011

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Pankaj Kamthan

Entitled: Representation of User Stories in Descriptive Markup

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________ Dr. Rachida Dssouli	Chair
______________________ Dr. Michael Weiss	External Examiner
______________________ Dr. Ferhat Khendek	Examiner
______________________ Dr. Peter Grogono	Examiner
______________________ Dr. Olga Ormandjieva	Examiner
______________________ Dr. Terrill Fancott	Thesis Supervisor

Approved by ____
Chair of Department or Graduate Program Director

_______ 20__ ____
Dean of Faculty of Engineering and Computer Science

ABSTRACT

Representation of User Stories in Descriptive Markup

Pankaj Kamthan, Ph.D.
Concordia University, 2011

The environment in which a software system is developed is in a constant state of flux. The changes at higher levels of software development often manifest themselves in changes at lower levels, especially its activities and artifacts. In the past decade, a notable change has been the emergence of agile methodologies for software development.

In a number of agile methodologies, user stories have been adopted as a style of expressing software requirements. This thesis is about theory and practice of describing user stories so as to make them amenable to both humans and machines. In that regard, relevant concerns in describing user stories must be considered and treated separately.

In this thesis, a number of concerns in describing user stories are identified, and a collection of conceptual models to help create an understanding of those concerns are formulated. In particular, conceptual models for user story description, stakeholders, information, representation, and presentation are proposed.

To facilitate structured descriptions of user stories, a User Story Markup Language (USML) is specified. USML depends on the requisite conceptual models for theoretical foundation. It is informed by experiential knowledge, especially conventions, guidelines, patterns, principles, recommended practices, and standards in markup language engineering. In doing so, USML aims to make the decisions underlying its development explicit.

USML provides conformance criteria that include validation against multiple schema documents. In particular, USML is equipped with a grammar-based schema document and a rule-based schema document that constrain USML instance documents in different ways.

USML aims to be flexible and extensible. In particular, USML enables a variety of user story forms, which allow a range of user story descriptions. USML instance documents can be intermixed with markup fragments of other languages, presented on conventional user agents, and organized and manipulated in different ways. USML can also be extended in a number of ways to accommodate the needs of those user story stakeholders who wish to personalize the language.

Acknowledgments

I am grateful to Dr. Terrill Fancott for his guidance, patience, and support throughout the course of this thesis.

I am thankful to Dr. Peter Grogono, Dr. Ferhat Khendek, and Dr. Olga Ormandjieva for agreeing to be on the thesis supervisory committee, and for their feedback and suggestions for improvement.

I learnt the basics of research from Dr. Abraham Boyarsky (Department of Mathematics and Statistics), Dr. Eusebius Doedel (Department of Computer Science and Software Engineering), and Dr. Michael Mackey (Department of Physiology, Physics, and Mathematics, McGill University), and they remain memorable.

I received much encouragement and support from Dr. Leslie Cohen and Mrs. Karen Taillon (CUPFA) over the years. It was instrumental in my teaching and research, beyond the realms of this thesis.

I am thankful to the students who took courses with me over the years. I have benefited from their curiosity and enthusiasm to learn.

I am obliged to the administrative staff of the Department of Computer Science and Software Engineering, especially Mrs. Halina Monkiewicz and Mrs. Hirut Adugna, for their professionalism and timely help.

I am indebted to the Faculty of Engineering and Computer Science for the financial support during the program.

Last, but not the least, certain gratitude can not be labeled. I have learnt much over the years from the anonymous reviewers of my submitted work, and that is immeasurable.

Table of Contents

List of Conventions

General Conventions

In this thesis, a concept is called ‘first-class’ if it can be studied independently of other concepts.

The term ‘author’ is used to denote the author of this thesis, unless otherwise stated.

The thesis includes remarks at a number of places. These remarks are used for different purposes, including contextualizing or lending support to an argument, pointing to related work, and so on.

Editorial Conventions

The thesis follows a number of editorial conventions for the sake of brevity, clarity, and consistency.

The thesis uses only two typefaces for text, a manifestation of the use of the TWO TYPEFACES pattern [Rüping, 2003].

A pair of single quotes (‘…’) is, in general, used to indicate colloquial words or phrases. A pair of double quotes (“…”) is, in general, used to indicate verbatim text from a citation.

A punctuation such as period (.), comma (,), or semi-colon (;) follows the single end quote (’) or double end quote (”), as appropriate[1].

There are places in the thesis, such as certain definitions, where there is text from an external source. In some cases, the text has been modified by the author. A modification by the author of text from an external source is included in square brackets ([…]).

The name of a pattern [Alexander, 1979] is indicated in uppercase to distinguish it from surrounding text.

A descriptive markup fragment is presented in constant space (Courier New) typeface with single line space.

A presence of the following text anywhere in markup means the presence of other elements and perhaps attributes (details of which has been suppressed for brevity):

```
<!-- ... -->
```

A presence of an ellipsis (. . .) anywhere in markup means the presence of information in an element, presence of information in an attribute, or other attributes (details of which has been suppressed for generality).

[1] This convention is inspired by Peter G. Neumann.

The references are structured using a technique by the author, and are ordered alphabetically.

There are certain resources available exclusively on the World Wide Web (Web). These are indicated in a footnote by including their corresponding Uniform Resource Locator (URL).

List of Abbreviations

In the following, the alphabetical list of all abbreviations, along with their expansions, that appear in the thesis are given.

ATDD	Acceptance Test-Driven Development
BABOK	Business Analysis Body of Knowledge
CMMI	Capability Maturity Model Integration
CMMI-DEV	CMMI for Development
COSMIC	Common Software Measurement International Consortium
CSS	Cascading Style Sheets
DCMES	Dublin Core Metadata Element Set
DSDL	Document Schema Definition Languages
DSDM	Dynamic Systems Development Method
DSL	Domain-Specific Language
DTD	Document Type Definition
FOAF	Friend of a Friend
FSM	Functional Size Measurement
GML	Generalized Markup Language
HTML	HyperText Markup Language
IANA	Internet Assigned Numbers Authority
MathML	Mathematical Markup Language
MLUS	Markup Language for User Stories
MSV	Multi-Schema XML Validator

NVDL	Namespace-based Validation Dispatching Language
OHCO	Ordered Hierarchy of Content Objects
OSS	Open Source Software
OWL	Web Ontology Language
RDF	Resource Description Framework
RESPECT	Requirements Engineering and Specification in Telematics
RTF	Rich Text Format
RELAX NG	REgular LAnguage for XML Next Generation
SGML	Standard Generalized Markup Language
SMM	Story Card Maturity Model
SRS	Software Requirements Specification
UCAP	User-Centered Agile Process
UML	Unified Modeling Language
USDM	User Story Description Model
USIM	User Story Information Model
USML	User Story Markup Language
USMS	User Story Management System
USPM	User Story Presentation Model
USRM	User Story Representation Model
USSM	User Story Stakeholder Model
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
Web	World Wide Web

WML	Wireless Markup Language
XHTML	Extensible HyperText Markup Language
XLink	XML Linking Language
XML	Extensible Markup Language
XP	Extreme Programming
XPath	XML Path Language
XSDL	XML Schema Definition Language
XSLT	Extensible Stylesheet Language Transformations

Chapter 1 Introduction

To be a person is to have a story to tell.
— Isak Dinesen

The idea of a ‘story’, and the use of a ‘story board’ for telling a story and soliciting feedback on a story, has its origins in the production of animated movies of the 1920s [Canemaker, 1999]. It inspired similar efforts in later fields, such as user interface design [Buxton, 2007], that involved production. The notion of a ‘user story’ was introduced to software engineering in the late 1990s [Beck, 2000].

The significance of user stories in software development calls for their suitable description. The aim of this thesis is to provide a means for describing user stories in a manner that is acquiescent to both human and machine consumption. It is important that such means rests on a theoretical foundation and is practical in its application.

The rest of the chapter motivates the software development environment in which user stories are practiced, states the thesis problem, explores previous work towards solving the problem, suggests an approach to a solution, and outlines the organization of the thesis.

1.1. The Need to Manage Expectations of Software Project Stakeholders

There are number of viewpoints from which the outcome of a software project can be assessed. From a delivery viewpoint, there can be three types of assessment of the outcome of a software project: successful, challenged, and failed.

The outcome of a software project is related to expectations of its stakeholders. It has been suggested that lack of managing expectations of stakeholders is one the primary reasons for software project failures [McManus, 2004, Page 3].

There are a number of stakeholders of a software project, each with their own set of expectations. For example, a project manager expects that the software project is completed within allocated resources, a software engineer expects that a software requirement can actually be implemented, a customer expects that the software system is worth the payment, and a user expects that the software system matches his/her goals.

The expectations of stakeholders need not be mutually exclusive; in fact, they can compete. For example, a software requirement proposed by the customer may be deemed infeasible by software engineers. It is such variability among expectations of stakeholders that calls for their adequate management.

A necessary condition for a software project to be successful is that there is equilibrium in managing expectations of (at least) high-priority stakeholders. However, in general, software engineers are not clairvoyant, and therefore it is not always possible for them to anticipate and be aware of the needs of customer/users [Stober, Hansmann, 2010, Chapter 1].

1.2. Non-Technical Stakeholders as Participants in a Software Project

There are a number of ways of managing expectations of high-priority stakeholders, including their explicit involvement in a software project [Whitehead, 2007]. Indeed, human-centered design methodologies[2] [Schuler, Namioka, 1993] and surveys [Chow, Cao, 2008; The Standish Group, 1995] have, over the years, underscored the need for involvement of customer/users in a software project for it to be successful. The involvement of customer/users as integral participants in a software project increases the likelihood that the resulting software system reflects their needs during development, and meets their expectations upon completion.

An agile project is a software project based on the Agile Manifesto[3], and an agile methodology[4] is a software development methodology used by an agile project for the development of a software system. There is emphasis on involvement of customer/users in agile methodologies. In agile methodologies, it is expected that software development is 'not rational' [Parnas, Clements, 1986], and the requirements emerge during development. In certain agile methodologies, software requirements are usually expressed, verbally or otherwise, as user stories.

[2] In this thesis, 'human-centered design methodology' is being used as a single encompassing term for 'user-centered design', 'user experience design', and other similar terms, unless otherwise stated.
[3] URL: http://agilemanifesto.org/ .

Figure 1.1 summarizes the previous discussion. It illustrates a number of related managerial concerns that motivated the inception of user stories.

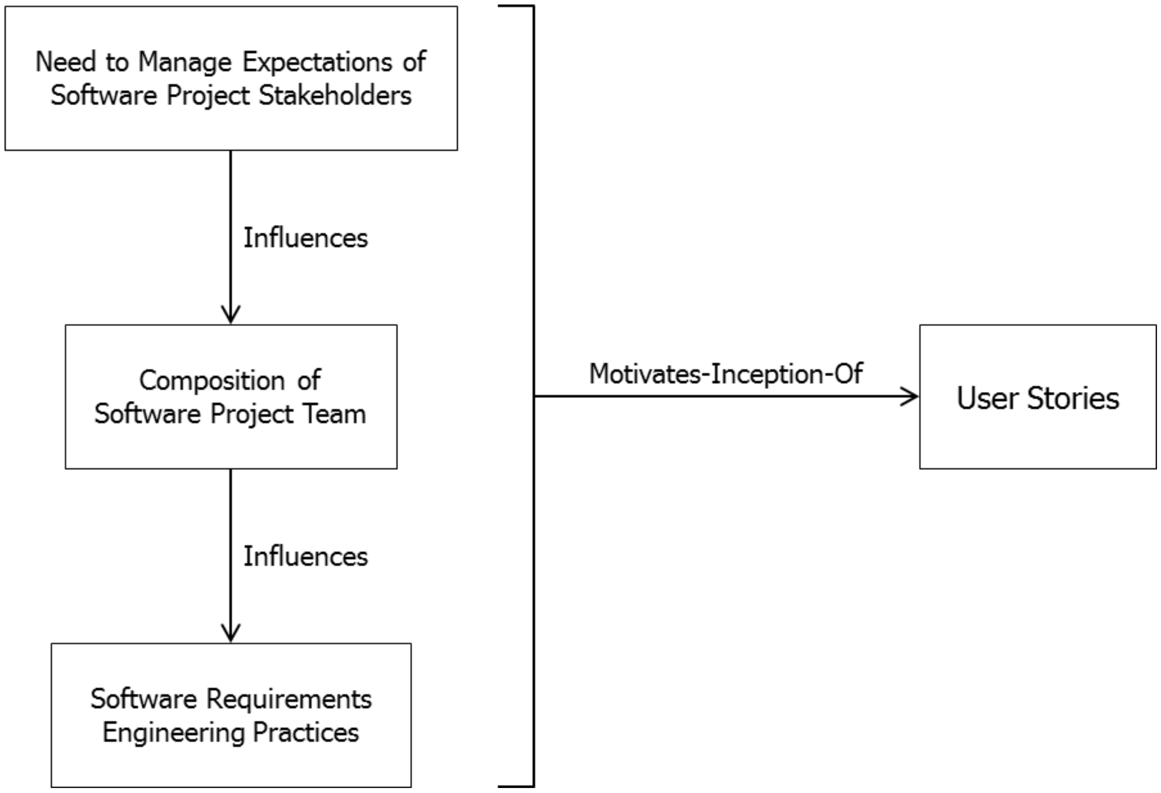

Need to Manage Expectations of Software Project Stakeholders
Influences
Composition of Software Project Team
Influences
Software Requirements Engineering Practices
Motivates-Inception-Of
User Stories

Figure 1.1. The inception of user stories as a consequence of managerial concerns.

1.3. User Story Practices in an Agile Project

The inclusion of non-technical stakeholders in a software project has ensuing cost. It is known that a productive involvement of all stakeholders, including customer/users, in a software project relies on effective communication [Coughlan, Macredie, 2002; Hoover,

[4] It is common in the current literature to associate (specifically, postfix) the term ‘agile’ with ‘method’, ‘methodology’, and ‘process’. In this thesis, these terms are not considered synonymous and, following the IEEE Standard 730.1-1995 [IEEE, 1995], the term ‘methodology’ is adopted and used.

Rosso-Llopart, Taran, 2009, Page 270; Nuseibeh, Easterbrook, 2000]. The same applies to agile projects [Cockburn, 2007, Chapter 3].

The development of user stories depends on a number of practices to facilitate communication. The user stories are elicited from conversation with customer/users. This conversation takes place during meetings and interviews. The language of communication needs to be sensitive to all stakeholders, including customer/users [Cohn, 2004, Page 3]. Therefore, the user stories are expressed in natural language and, in doing so, technical terms are avoided. The scope of the user stories is decided through collaboration and negotiation. It is understood that user stories are only mementos of conversation [Jeffries, 2001], and it is expected that customer/users are available in person if clarification is sought. For a number of reasons [Brown, Lindgaard, Biddle, 2008; Spinellis, 2007], including low cost, ease of exchange across a table, and non-reliance on any special skills, paper-based index cards are used as the medium for authoring user stories. These paper-based index cards are discarded after use [Langr, Ottinger, 2011]. The close proximity and conversing/listening to each other leads to an improvement in the relationship between technical and non-technical stakeholders [Alexander, Maiden, 2004, Page 268] that, in turn, contributes to building necessary mutual trust [Bang, 2007; Kovitz, 2003].

1.3.1. Issues in User Story Practices

There are a number of issues that emanate from the current user story practices of relying on the implicit knowledge and paper-based index cards:

- **Reliance on Implicit Knowledge.** There are a number of issues in relying on the implicit knowledge of team members. It takes time and mutual trust to build reliable implicit knowledge. This can be challenging for software projects following an agile methodology in a short time-to-market environment with non-proximal (say, geographically dispersed) teams [Larman, Vodde, 2010; Šmite, Moe, Ågerfalk, 2010]. Furthermore, people are known to have short-term memories; people may, voluntarily or otherwise, leave a software project prematurely; and people may work on a number of different software projects, and may not entertain the fact that they are required to remember important details of each of them.

- **Reliance on Paper-Based Index Cards.** There are a number of issues in relying on paper-based index cards for expressing user stories, and discarding them after use [Breitman, Leite, 2002; Lewitz, 2004]. For certain reasons, including legality, an organization may be obligated to retain user stories. It is impractical to exchange or share paper-based index cards across long distances, which is necessary for geographically dispersed (or non-co-located) software project teams. A paper-based index card has fixed space, which has its consequences. There are limits to the number of times modifications can be made on paper-based index cards, and limits to maintaining a history of modifications. A paper-based index card is static medium, which has a number of implications. It is not possible to manipulate the text on a paper-based index card. (For example, such a facility can be helpful for searching or sorting text in some manner.) The reader of a user story on a paper-based index card has to rely on the handwriting skills of the writer. It is not possible to display the text on a paper-based index card any differently than the original. (For example, such a facility can be helpful for accessibility.) There is no ‘dynamic’ between paper-based index cards, so it is not possible to ‘navigate’ between cards, or from one card to another software process artifact. (For example, such navigation can be helpful for clarity or traceability.)

To recapitulate, in relying on the implicit knowledge and paper-based index cards to be discarded, there is potential for irreversible loss of collective organizational memory.

1.4. Problem Statement

There is need for a means for describing user stories that overcomes the aforementioned issues, and is amenable to both humans and machines. In particular, such means must have a number of properties, including the following, categorized and listed in no particular order of significance:

- **[P1] People.** The means is such that a user story description can explicitly incorporate the needs of software project stakeholders. In particular, the means enables a user story description to include information that is relevant to different stakeholders. For example, such stakeholders include project manager, software engineer, customer, and user.

- **[P2] Proliferation.** The means is such that it provides support for a user story description to be used in the current electronic communication infrastructure. In particular, such infrastructure includes the Internet and the Web. It should be possible to disseminate a user story description essentially anywhere, at any time. For example, such dissemination is relevant to geographically dispersed software project teams.

- **[P3] Processing.** The means is such that it is possible to manipulate a user story description in a number of ways. In particular, such manipulation include interfacing a user story description with other software process artifacts; navigating between user story descriptions, or between a user story description and other software process artifacts; presenting a user story description, at different levels of abstraction, on different devices; organizing multiple user story descriptions in different ways. For example, such manipulation is relevant to authors, reviewers, and readers of user stories.

1.5. Previous Work on Describing User Stories

The proliferation of agile project management systems[5] is a step in the direction of addressing the disadvantages inherent to implicit knowledge and paper-based index cards.

There are certain agile project management systems that use operating system-specific and proprietary means, such as Microsoft Excel, for describing user stories. There have also been a few initiatives in the direction of using the Extensible Markup Language

(XML) for describing user stories. In the following, these efforts are analyzed briefly and chronologically.

[PW1] In [Breitman, Leite, 2002], a schema for a "scenario structure to capture and record use stories" is given. A user story and a scenario are considered equivalent. In the schema, a user story is decomposed into tasks, and a task is represented by one or more episodes of the scenario. However, this initiative has the following limitations: an instance document based on this schema corresponds to a user story that is specific to one agile methodology, the granularity of the schema is coarse, the typing of the schema is weak, and there is no support for estimation or acceptance criteria.

[PW2] In [Rees, 2002], a Markup Language for User Stories (MLUS) has been proposed. MLUS has certain meta-information and information elements of a user story, and is intended to be used with a proof-of-concept user story tool called DotStories. However, this initiative has the following limitations: the specification of MLUS, if it exists, is not publicly available; apart from a single instance document, the details of MLUS, such as a schema, are not given; user story information is expressed in `CDATA` sections that may be ignored or may not preserved by an XML processor [Harold, 2003, Item 15]; and consists of a user interface-specific element that belongs to the solution domain.

[PW3] XP Studio[6] is an open source software development environment for supporting XP practices. It includes a schema, namely `user-story.xsd`[7], which expresses

[5] URL: http://www.userstories.com/products/ ; URL: http://www.agile-tools.net/ .
[6] URL: http://xpstudio.sourceforge.net/ .

certain meta-information and information elements of a user story. However, this initiative has the following limitations: the constructs of `user-story.xsd` are not always rationalized; the definition of a user story used by `user-story.xsd` is incorrect ("… system requirement …"); and `user-story.xsd` includes certain elements (such as for a task) that are related to but are outside the domain of user stories.

[PW4] IceScrum[8] is an open source agile project management system that provides XML export of user stories. However, the specification corresponding to the export, if it exists, is not publicly available.

[PW5] Rally Community Edition[9] is a commercial agile project management system that provides XML export of user stories. However, the specification corresponding to the export, if it exists, is not publicly available.

In other words, the problem of a suitable means for describing user stories remains, and provides the motivation for this thesis.

1.6. Solution Approach

This thesis takes the following approach to the problems outlined in Section 1.4:

[7] URL: http://xpstudio.sourceforge.net/xpml/user-story.xsd .
[8] URL: http://www.icescrum.org/ .
[9] URL: http://www.rallydev.com/agile_products/editions/community/ .

- **Theory.** There must be a general understanding of the foundation of user story description, including that of the information it can contain. There must also be a general understanding of the people involved potential targets of a user story description, that is, of the user story stakeholders. This requires a construction of appropriate conceptual models, independent of any software development methodology or technology.

- **Practice.** There must be a general means for expressing arbitrary user story descriptions that are readable by both humans and machines. This means must also be minimally constrained in the sense that it must be independent, as far as possible, of any specific hardware or software, and it must be open (that is, it must be non-proprietary and independent of any vendor). This requires the construction of a system based on a general model of text, namely descriptive markup.

1.7. Organization of the Thesis

The rest of the thesis is organized as follows. There are 5 chapters, namely Chapters 2–6, that form the theoretical framework of the thesis. There are 5 chapters, namely Chapters 7–11, that form the practical framework of the thesis, and 5 appendices that lend integral support to these chapters.

Theoretical Framework

In Chapter 2, user stories are placed in the context of software process engineering, in general, and requirements engineering, in particular.

There are a number of definitions of model, including the following:

Definition 1.1 [Model]. A simplification, with respect to some goal, of a thing.

A model could be physical or conceptual. In this thesis, the interest is in conceptual models.

The rest of the chapters, although presented linearly by necessity, are not necessarily mutually exclusive.

In Chapter 3, a User Story Description Model (USDM) is proposed. The purpose of USDM is to provide an understanding of a user story description.

In Chapter 4, a User Story Stakeholder Model (USSM) is proposed. The purpose of USSM is to identify and classify the people involved in an activity related to a user story.

In Chapter 5, a User Story Information Model (USIM) is proposed. The purpose of USIM is to scope the information contained in a user story description.

In Chapter 6, a User Story Representation Model (USRM) is proposed. The purpose of USRM is to provide a model for representing a user story in descriptive markup, and to initiate technological commitments.

The aforementioned conceptual models, as illustrated in Figure 1.2, collectively provide an input to the User Story Markup Language (USML).

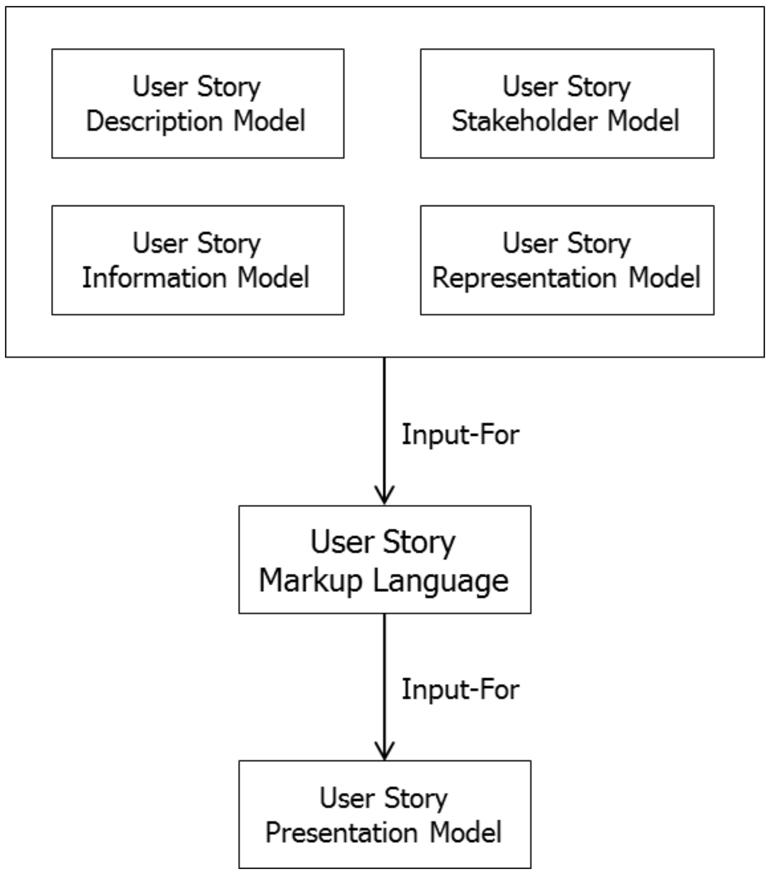

User Story Description Model
User Story Stakeholder Model
User Story Information Model
User Story Representation Model
Input-For
User Story Markup Language
Input-For
User Story Presentation Model

Figure 1.2. The relationship between user story-related conceptual models and USML.

Practical Framework

In Chapter 7, the scope and limitations of USML are highlighted. In particular, the motivation, purpose, and major decisions underlying USML are provided.

In Chapter 8, the elements and attributes of USML are identified and defined. The set of elements and the set of attributes determine, in part, the capabilities of USML.

In Chapter 9, certain scenarios of using USML are presented, and supported by examples. These scenarios demonstrate, in part, the breadth, depth, and flexibility of USML. In doing so, a conceptual model for presenting a user story description, namely User Story Presentation Model (USPM), is proposed. USML is an input to USPM, as illustrated in Figure 1.2.

In Chapter 10, a number of potential extensions of USML are proposed, and are demonstrated through examples. USML has been designed for extensibility, both by necessity and by choice.

In Chapter 11, different approaches for evaluating USML are suggested.

In Chapter 12, concluding remarks are given, and directions for future research are outlined.

In Appendices A and B, a grammar-based schema for USML and a rule-based schema for USML, respectively, are given. These appendices lend support to Chapters 7 and 8.

In Appendix C, a collection of USML instance documents, for the purpose of demonstration, are given. In Appendix D, style sheets associated with the USML instance documents of Appendix C are included. These appendices lend support to Chapter 9.

Finally, in Appendix E, the tools used in the thesis are listed. This appendix lends support to Chapters 7–10 and Appendices A–D.

Chapter 2 Background

[…] design a thing by considering it in its next larger context: a chair in a room, a room in a house, a house in an environment, an environment in a city plan.
— Eliel Saarinen

This chapter places user stories in the overall context of software process engineering, in general, and requirements engineering, in particular. In doing so, it highlights the current environment consisting of standards and similar efforts that supports user stories, and inherent limitations of user stories in practice. It also provides a comparison between use cases and user stories. The chapter concludes with a position on user stories.

2.1. Situating User Stories in Software Process Engineering

In software engineering, there are a number of different visions of how the development of a software system should take place. These visions have, over the past few decades, led to a number of different methodologies for software development.

The Agile Manifesto is one such vision for software development. It consists of a collection of values and principles for software development. There are a number of agile methodologies [Boehm, Turner, 2004; Williams, 2010] that vary in their realization of the Agile Manifesto. There are comparisons of agile methodologies available [Boehm, Turner, 2004; Williams, 2010].

In agile software development, user stories are supported explicitly in the Agile Project Management Delivery Framework [Highsmith, 2009], are supported explicitly in Behaviour-Driven Development (BDD)[10] [Chelimsky, Astels, Dennis, Hellesoy, Helmkamp, North, 2010], are supported explicitly in Extreme Programming (XP) [Beck, 2000; Beck, Andres, 2005], are supported explicitly in Scrum [Schwaber, 2004; Schwaber, Beedle, 2002], and are supported explicitly in the User-Centered Agile Process (UCAP) [Beyer, 2010, Chapter 5].

There is support in software development community for XP and Scrum in a number of different ways. It has been reported via surveys, such as [Hussain, Slany, Holzinger, 2009], that XP and Scrum are the most broadly used agile methodologies in the industry. XP and Scrum have also been extended in different directions. For example, UCAP builds upon Contextual Design [Holtzblatt, Beyer, 1995], XP, and Scrum to provide a "sound understanding of the user".

There are a number of initiatives for software process maturity, including the Capability Maturity Model Integration (CMMI). In CMMI for Development (CMMI-DEV) for CMMI Version 1.3 [CMMI Product Team, 2010], user stories occur in the following process areas: Configuration Management, Requirements Management, and Verification.

[10] URL: http://behaviour-driven.org/ .

2.2. Situating User Stories in Software Requirements Engineering

There are a number of definitions of (software) requirement, including the following:

Definition 2.1 [Requirement] [IEEE, 1990].

(1) A condition or capability needed by a user to solve a problem or achieve an objective.
(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
(3) A documented representation of a condition or capability as in (1) or (2).

In the previous definition, (1) and (3) are relevant to this thesis. However, (3) does not separate the notion of a requirement and its representation.

There are a number of definitions of user story [Cohn, 2004; Leffingwell, 2011], including the following[11]:

Definition 2.2 [User Story]. A high-level [software] requirement that contains minimally sufficient information to produce a reasonable estimate of the effort to implement it.

In this thesis, user story is a 'first-class' concept.

From the perspective of software requirements, a user story can be seen as playing a dual role. A user story is a user-centered requirement due to its relevance to a user of the software system, and a management-oriented requirement due to its significance for people involved in the software project management. This is elaborated in Chapter 5, and illustrated in Figure 2.1.

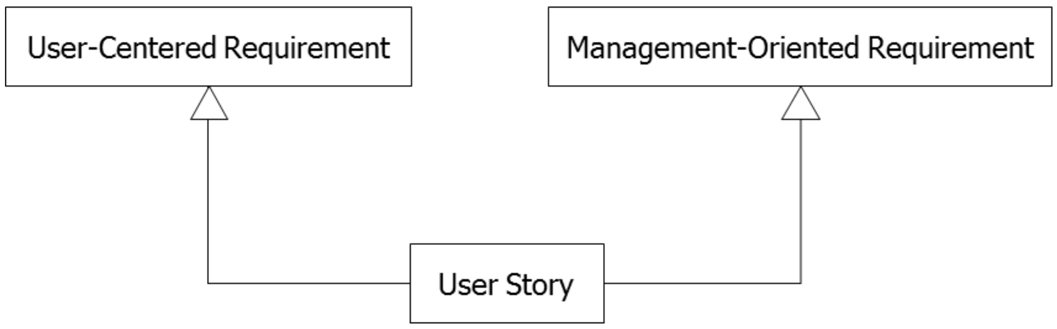


Figure 2.1. The relationship between user story and software requirements.

The Business Analysis Body of Knowledge (BABOK) 2.0 [IIBA, 2009] is “the collection of knowledge within the profession of Business Analysis and reflects current generally accepted practices”. The user stories have been covered in the Requirements Analysis Knowledge Area of BABOK.

2.2.1. Agile Requirements and User Stories

An agile requirement is a requirement pertaining to a software system based on an agile project. An agile methodology consists of, among other things, a process model that outlines the activities and artifacts that are to be produced as a result of those activities. In

[11] URL: http://www.agilemodeling.com/artifacts/userStory.htm .

process models of agile methodologies such as XP and Scrum, an agile requirement is a user story.

There has been previous work on situating user stories with respect to agile requirements. The user stories belong to the Story Development Phase of the Agile Requirements Generation Model [Soundararajan, 2008], and are part of an Agile Requirements Ontology [Ajmeri, Sejpal, Ghaisas, 2010].

2.2.2. 'Conventional' Software Requirements versus User Stories

The notion of a software requirement in agile projects is fundamentally different from that in 'conventional' software projects [Leffingwell, 2011, Page 16]. The difference between the two comes with respect to the degrees of freedom.

In 'conventional' software projects, the collection of software requirements, say, collated in a software requirements specification (SRS), determine the resources for the software project and the date of delivery of the software product. However, in agile projects, it is the converse: the resources and the date of delivery determine the scope of agile requirements.

Figure 2.2 is adapted from [Leffingwell, 2011, Figure 1-8; Stober, Hansmann, 2010, Figure 1.1], and shows a comparison between 'conventional' and agile requirements engineering.

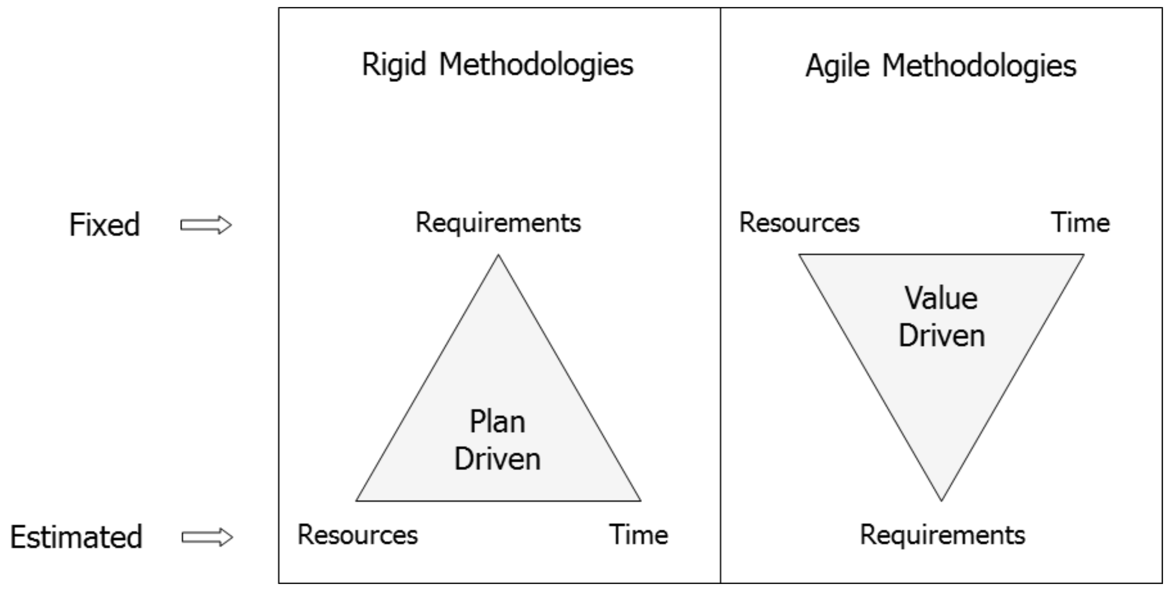


Figure 2.2. The difference between 'conventional' and agile requirements engineering.

The origin, style, and purpose of a user story are also in contrast with the standards for a 'conventional' software requirement such as the IEEE Standard 830 [IEEE, 1998]. For example, user stories are authored incrementally [Cohn, 2004, Page 52]; however, it is expected by the IEEE Standard 830 that all software requirements are provided upfront. A comparison of user story and 'conventional' software requirement as per IEEE Standard 830 is beyond the scope of this thesis.

2.2.3. Use Cases versus User Stories

It is suggested in theories of learning, such as constructivism [Piaget, 1952], that upon initial exposure to a new concept C, a comparison between C and other closely related and relatively more established concepts C' is inevitable. This comparison is a necessary prerequisite for creating an understanding of C through assimilation and accommodation.

In this section, C is user story and C' is use case [Jacobson, Christerson, Jonsson, Övergaard, 1992].

In the following, previous efforts that compare use cases and user stories are analyzed briefly and chronologically.

In [Beck, 2000], it has been pointed out that user stories are “simplified use cases”. However, the meaning of ‘simplification’ and exactly what is “simplified” has not been given.

In [Cohn, Paul, 2001], it has been stated that use cases are “structured” while user stories (in XP) are “unstructured”. In [Decker, Ras, Rech, Klein, Hoecht, 2006], it has been stated that “use cases are a structured representation of a user story”. In [Alexander, Maiden, 2004], it has been stated that “[use cases] are expressed using a constrained (semi-formal) syntax” and that “[user stories] are expressed using natural language prose”. However, there are a number of ways of expressing a use case [Cockburn, 2001], and not all of them need to be “structured” or follow a “constrained (semi-formal)” syntax.

In [Cohn, 2004], it has been asserted that “use cases are more prone to including details of the user interface” and in [Monochristou, Vlachopoulou, 2007] that “it is usual for use cases to include user interface details”. However, there are guidelines [Cockburn, 2001;

Wiegers, 2003] that explicitly suggest against the inclusion of any user interface specifics in use cases.

In [Gallardo-Valencia, Olivera, Sim, 2007], based on a controlled experiment, it is concluded that use cases can be useful as a complement to user stories.

In the rest of this section, use cases and user stories are compared systematically. The purpose of comparison is to facilitate understanding and appropriate use of each. The criteria for comparison are considered equally significant and are organized by different, basic viewpoints, as given in Table 2.1.

Viewpoint	**Criterion**
Project	• Estimate • Schedule
Process	• Methodology • Development
People	• Users • Value
Product	• Scope • Description
Resource	• Purpose • Maturity

Table 2.1. A framework for comparing use cases and user stories.

Use Cases versus User Stories: Project-Viewpoint

Criterion: Estimate

Use Cases: In [Karner, 1993; Mohagheghi, Anda, Conradi, 2005], use cases have been used as a basis for estimation. The purpose of estimation is to schedule the entire software project. This approach requires the presence of all use cases upfront, and the fact that they all are expressed in a certain manner, for calculating the estimate. However, a use case is not intrinsically related to estimation. Furthermore, software development methodologies that currently support use cases, do not use the estimates based on use cases for planning.

User Stories: A user story must be associated with an estimate. The estimation must be part of a user story development process. In other words, a user story description without an estimate is incomplete. A user story is estimated individually. In both XP and Scrum, the purpose of estimating user stories is short-term planning. In particular, the user story estimate is used for iteration planning and release planning.

Criterion: Schedule

Use Cases: The completion (design and implementation) of a use case is not explicitly time-bound. A single use case could be completed over a number of iterations.

User Stories: The completion of a user story is time-bound. A single user story must be completed in a single iteration.

Use Cases versus User Stories: Process-Viewpoint

Criterion: Methodology

Use Cases: The use cases are originally associated with Objectory. There is explicit support for use cases in Unified Process and its customizations, in ICONIX Process, and in Crystal Clear.

User Stories: The user stories are originally associated with XP. There is explicit support for user stories in other agile methodologies such as Scrum, and its extensions such as UCAP and U-SCRUM.

The author is not aware of any software development process that has explicit support for both user stories and use cases.

Criterion: Development

Use Cases: The development of a use case model can be based on introspection.

User Stories: The development of user stories relies on extrospection. The social aspect of the development is exhibited in form of proximity (meetings/interviews), communication, and collaboration, and is considered necessary for elaboration and clarification of user stories.

Use Cases versus User Stories: People-Viewpoint

Criterion: Users

Use Cases: In use case modeling, the users form a subset of human actors. The users elicited during use case modeling can be fictitious (or surrogate users [Alexander, Beus-Dukic, 2009, Page 39]). These users may be based on introspection, not necessarily on actual communication with the real users.

User Stories: There must be an actual person behind a (user) role in a given user story. Indeed, a user story is initially expressed for a concrete person, and is subsequently generalized to a user story for an abstract (user) role.

Criterion: Value

Use Cases: The value provided by a use case to an actor is implicit.

User Stories: The value provided by a user story to its (user) role must be explicit. This value serves as a rationale for the user story.

Use Cases versus User Stories: Product-Viewpoint

Criterion: Scope

Use Cases: In general, a use case is larger in scope compared to a user story. The scope of a use case is comparable to that of an epic [Collier, 2012, Page 99]. (An epic is a user story so large that it can not be completed in a single iteration, and therefore must be split [Cohn, 2004, Page 6].)

User Stories: In general, the scope of a user story is comparable to that of a use case scenario.

Criterion: Description

Use Cases: A use case can be described at different levels of details [Cockburn, 2001]. The description can be expressed in different modalities, namely text and graphics. It is expected that technical terminology is avoided in the description to make it amenable to non-technical stakeholders. However, doing so is unavoidable in a use case model expressed in the Unified Modeling Language (UML). The exceptional cases are dealt with in recovery and failure scenarios. The possible types of relationships between use cases are known [Adolph, Bramble, Cockburn, Pols, 2003]. The description of a use case is not intrinsically related to the existence of test cases. Indeed, the test cases are separated from use cases and, for example, may appear in a test plan, or otherwise in some form of test documentation. A use case description is a relatively permanent artifact that persists throughout the development and maintenance of the software system.

User Stories: A user story has single level of description. The description has a single modality, namely text. It is expected that technical terminology is avoided in the description to make it amenable to non-technical stakeholders. The exceptional cases are dealt with in acceptance tests, not in the user story statement. There is currently no clear understanding of the relationships between user stories. A user story must be associated with acceptance criteria. The acceptance criteria must be part of a user story development process. In other words, a user story description without acceptance criteria is incomplete. The tests are part of the description of a user story. The tests, by being on the same index card, are intimately related to a user story[12]. A user story description, at least in the classical sense [Beck, 2000], is intended to be transient: its life is limited to the iteration for which it is formulated[13]. Therefore, user stories need not be archived for long-term use. Indeed, index cards are used as a medium for user stories, and these cards are usually discarded after use [Langr, Ottinger, 2011].

[12] This is in agreement with the 'T' of INVEST [Wake, 2002] and INSERT [Patel, Ramachandran, 2009], where 'T' expands to Testability, one of the desired quality attributes of a user story.

[13] This is in accord with Travel Light, one of the principles of XP. This principle has also been adopted by Agile Modeling [Ambler, 2002].

Use Cases versus User Stories: Resource-Viewpoint

Criterion: Purpose

Use Cases: A use case is a model of the external behavior resulting from the interaction between an actor and the system.

User Stories: A user story is a statement of the desired functionality, from the perspective of a user. This desired functionality is aimed for user's benefit, but it may not be related to user's interaction with the system.

Criterion: Maturity

Use Cases: The notion of a use case was introduced in the middle of 1980s. There is established, experiential body of knowledge for use cases in form of guidelines [Cockburn, 2001], patterns [Adolph, Bramble, Cockburn, Pols, 2003; Issa, Odeh, Coward, 2006; Övergaard, Palmkvist, 2005], and anti-patterns [El-Attar, 2009].

User Stories: The notion of a user story was introduced in the late 1990s. There is growing, yet undeveloped, experiential body of knowledge for user stories in form of 'smells' [Cohn, 2004, Chapter 14], guidelines [Cohn, 2004; Patel, Ramachandran, 2009a; Wallace, Raggett, Aufgang, 2002], and patterns [Leffingwell, 2011, Table 6-1].

Conclusion

The following conclusion can be drawn from the above comparison between use cases and user stories:

- The use cases and user stories have different, albeit overlapping, goals, and different means of achieving those goals.
- The use cases and user stories can co-exist and support each other. Indeed, in the rest of the thesis, there are explicit references to the body of knowledge on use cases, as necessary.

2.3. Support for User Stories

There are, as discussed previously, different avenues of increasing support for user stories, emanating from academia, standards organizations, and industry. The current environment of community support for user stories is summarized in Figure 2.3.

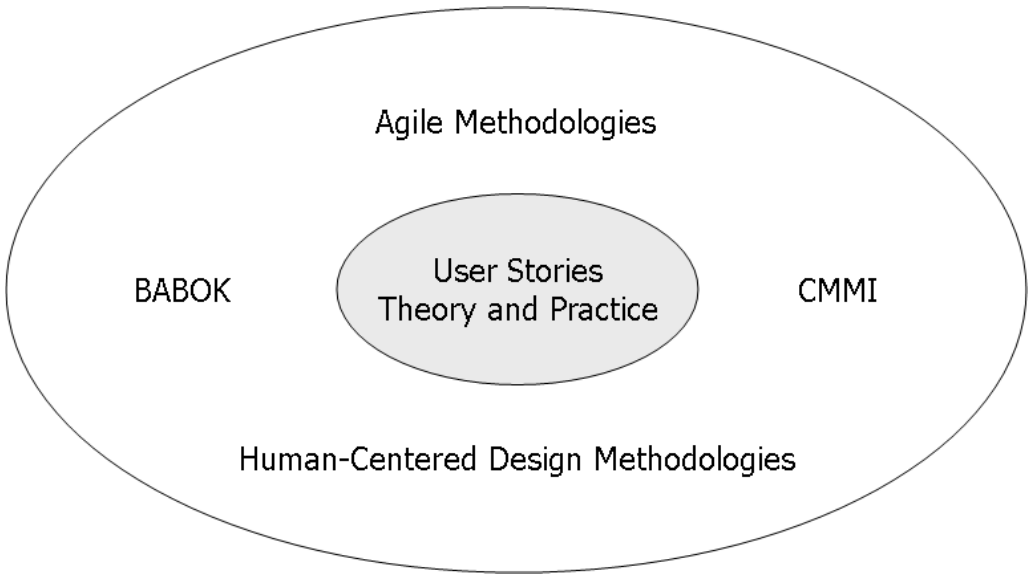


Figure 2.3. A panorama of the environment that currently supports theory and practice of user stories.

2.4. Limitations of User Stories in Practice

The support for user stories is not universal. There are inherent limitations of user stories that limit their applicability for certain types of software projects.

- **Involvement of Non-Technical Stakeholders.** There are a number of reasons, including the constraints of space or time, due to which the involvement of the customer or users in the development of user stories is not automatic [Hoda, Kruchten, Noble, Marshall, 2010]. Indeed, the advantages and disadvantages of customer involvement, especially if it is on a full-time basis, in XP-based software projects have been considered in [Mohammadi, Nikkhahan, Sohrabi, 2009]. In certain cases, such as software systems for children or for people with low literacy, the involvement of users may not be useful or feasible.

- **Expectation of Requisite Skills.** The development of user stories of 'high' quality depends on a number of social and technical skills including collaborating, negotiating, estimating, prioritizing, and authoring skills. It is not automatic that a person has, or can acquire these skills [Bunse, Feldmann, Dörr, 2004; Hoda, Kruchten, Noble, Marshall, 2010; Kovitz, 2003] at the level expected.

- **Presence of Abilene Paradox.** It has been reported in [McAvoy, Butler, 2006] that, after an initial commitment to user stories in a software development project, the interest of software engineers diminished significantly, ranging from initial commitment to skepticism, to virtual abandonment. The underlying reasons for the reduction in commitment are explained by the theory of competing commitments. In particular, it is shown that there was presence of the 'Abilene Paradox[14]' [Harvey, 1988].

2.5. Towards a Realignment of User Stories

The agile methodologies, as originally envisaged, are usually perceived as a significant departure from 'conventional' software engineering. However, it is increasingly being realized [Leffingwell, 2011; Rodríguez, Yagüe, Alarcón, Garbajosa, 2009] that the agile methodologies can not completely ignore the principles and practices of 'conventional' software engineering if they aim to deliver on their promise.

[14] The Abilene Paradox is a paradox in which a group of people collectively decide on a course of action that is counter to the preferences of any of the individuals in the group [Wikipedia].

The position of this thesis is that the theory and practice of user stories, especially as it appears in agile methodologies, can benefit from a better alignment with the state-of-the-art of ‘conventional’ software engineering, and cognate disciplines such as software requirements engineering and human-centered design methodologies. In particular, as illustrated in Figure 2.4, a ‘balance’ between agility and rigidity is necessary, and this should be reflected in the way user stories are developed, described, and disseminated.

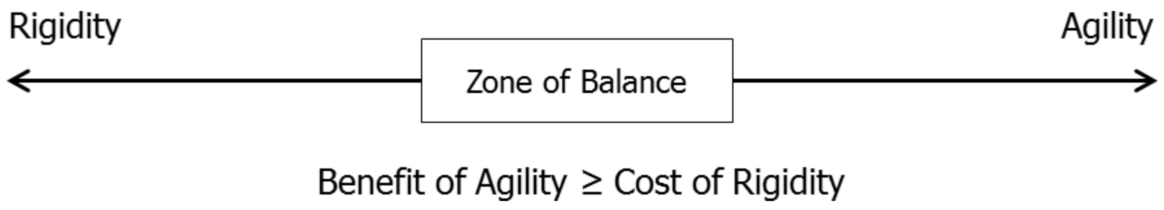


Figure 2.4. A spectrum of ‘balance’ between rigidity and agility in software development.

2.6. Summary

The agile methodologies are a class of software development methodologies that support iterative and incremental development. In agile methodologies, such as XP and Scrum, a software requirement is expressed as a user story. There is currently broad support in the community for user stories; however, there are also limitations of user stories in practice. There are stark differences between ‘conventional’ and agile requirements engineering. There are similarities, as well as differences, between use cases and user stories. There is need for cooperation, rather than competition, between user stories and ‘conventional’ software engineering.

Chapter 3 A User Story Description Model

It is a great misfortune in software development that the word 'model' has become so devalued. In common usage it means no more than 'description'.
— Michael A. Jackson

A user story can be made explicit through a user story description. There is currently no 'standard' way of describing a user story.

This chapter presents a conceptual model for user story description, namely USDM. The purpose of USDM is to help create an understanding of a user story description. USDM is also intended to be useful for subsequent deliberation.

3.1. Goals and Strategy

The goals of USDM are to create an understanding of a user story description, and to serve as an input to other conceptual models and USML. USDM aims to be independent of any specific medium, technology, or tool.

USDM is based on the following strategy:

- **Identify Concerns.** In describing a user story, there are a number of concerns. USDM identifies those concerns of a user story description that are relevant to humans and machines.

- **Identify Properties.** A user story description has a number of properties. USDM focuses on one such property, namely structure. It also highlights salient aspects of the structure of a user story description.

- **Identify Constructs.** A user story description is composed of certain information. To create an understanding of the composition of a user story description, USDM introduces certain atomic and composite constructs, and suggests a means to organize these constructs.

3.2. User Story Description

There are a number of definitions of statement, including the following:

Definition 3.1 [Statement]. A declarative sentence that is either true or false.

A sentence is a grammatical entity, whereas a statement is a logical entity.

Definition 3.2 [User Story Description]. A set of indicative, putative, and/or optative statements that specify a user story.

For the sake of this thesis, an indicative statement is about something that has already happened, a putative statement is about something that has not happened yet, and an optative statement is a wish for something to happen.

For example, in a user story description, the license (declaration), if any, is an indicative statement; the user story statement is a putative, as well as an optative, statement; and a constraint, if any, is a putative statement.

The purpose of a user story description is to make implicit knowledge pertaining to a user story explicit.

Remark 3.1. There can be different kinds of descriptions. For example, in contrast to a user story description, a pattern description consists only of indicative statements [Kamthan, 2010].

3.3. User Story Description: Information, Representation, and Presentation

The information contained in a user story description, the representation of that information, and presentation of that information are different concerns, as illustrated in Figure 3.1.

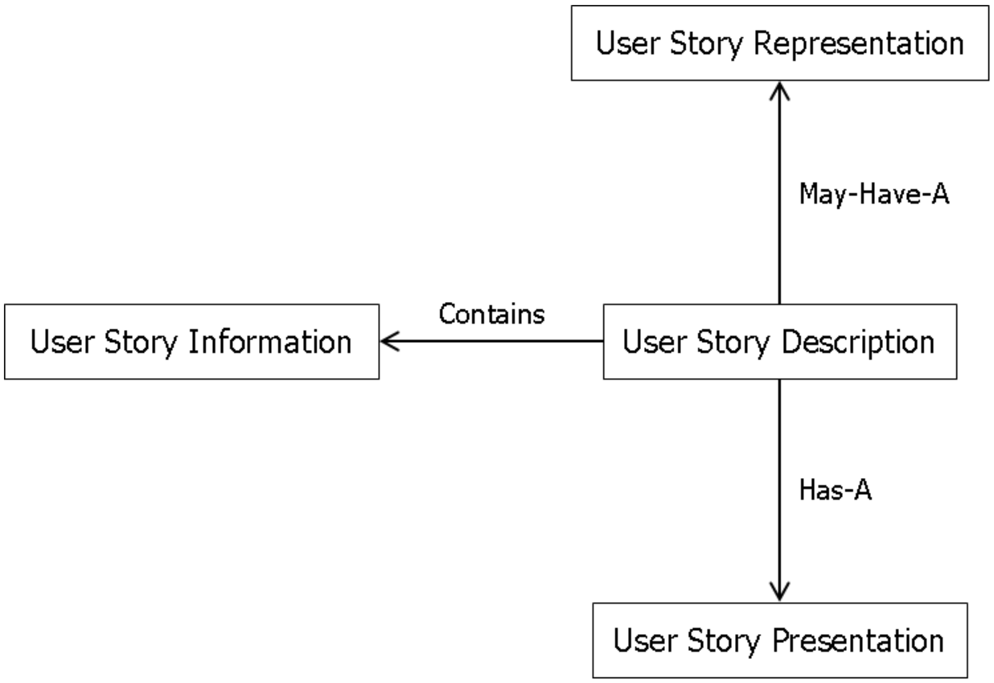


Figure 3.1. A high-level view of a user story description model.

In general, representation is intended for machine consumption, and presentation is intended for human consumption. For a number of reasons including, but not limited to, accessibility and modifiability of a user story description, the concerns of representation and presentation of information should be separated and treated separately.

Remark 3.2. Figure 3.1 is a manifestation of Separation of Concerns, one of the software engineering principles [Ghezzi, Jazayeri, Mandrioli, 2003].

Remark 3.3. There is a difference between a concept, such as a user story, and its representation. (In [Kaindl, Svetinovic, 2010], the distinction between a requirement and its representation has been emphasized.) Indeed, a user story can have one or more representations.

3.4. Structure of User Story Description

The structure of a user story description is of interest to both humans and machines. There are two aspects of structure of a user story description, the physical structure and the logical structure.

1. **Physical Structure of a User Story Description.** The physical structure of a user story description is concerned with the type of information contained in a user story description.

2. **Logical Structure of a User Story Description.** The logical structure of a user story description is concerned with the order of information contained in a user story description.

Remark 3.4. In [Highsmith, 2009, Figure 7-3], the necessity of (logically) structuring a user story has been emphasized. However, means of realizing such a description are not considered.

3.5. User Story Information Unit

There is certain information pertaining to a user story that can be considered as salient. This motivates the following definition:

Definition 3.3 [User Story Information Unit]. A labeled placeholder for information that accentuates a certain aspect of a user story.

For example, a user story, as an intellectual property or creative work, could be released under a license. Then, the license (declaration) is a user story information unit. There are a number of other user story information units, as given in Chapter 5.

3.6. User Story Information Unit and Structure of User Story Description

The structure of a user story description is intimately related to the notion of a user story information unit. In particular, the physical structure of a user story description is related to the occurrence of specific user story information units, and the logical structure of a user story description is related to the absolute or relative order of user story information units.

If a user story description is a 'molecule', then a user story information unit is an 'atom'. This analogy is illustrated in Figure 3.2. The variation in colors could be seen as a depiction of variety among atoms.

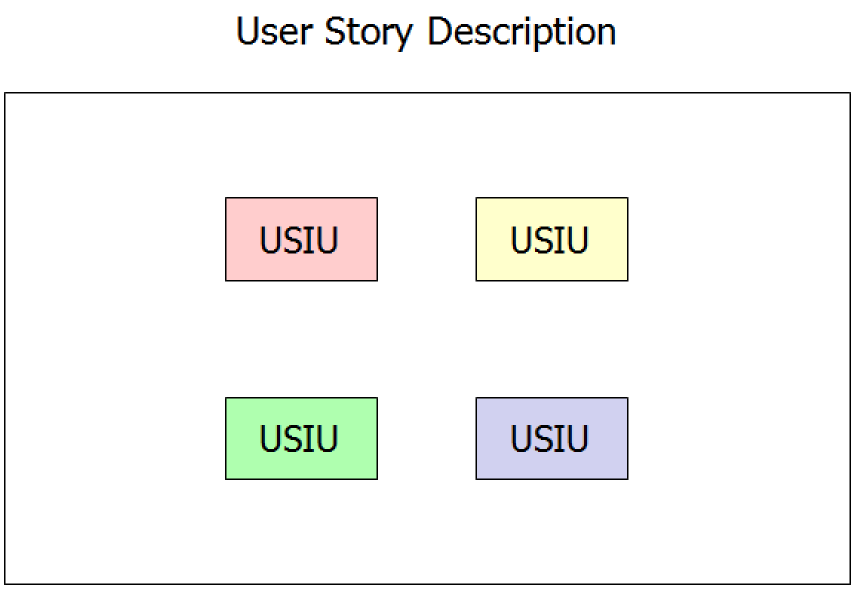


Figure 3.2. A user story description 'molecule' consists of user story information unit 'atoms'.

3.6.1. Example

In order to satisfy the needs of different user story stakeholders (as given in Chapter 4), it should be possible to have user story descriptions at different levels of abstraction. This can be achieved by placing different kinds of constraints on user story information units.

In Figure 3.3, there are three user story information units, namely USIU-1, USIU-2, and USIU-3. For example, USIU-1 could be a license (declaration), USIU-2 could be the user story statement, and USIU-3 could be a user story author (name). Then, according to the constraints shown, USIU-1 must appear first, if at all, and not more than once; USIU-3 must appear second, and at least once; and USIU-2 must appear third, and once and only once.

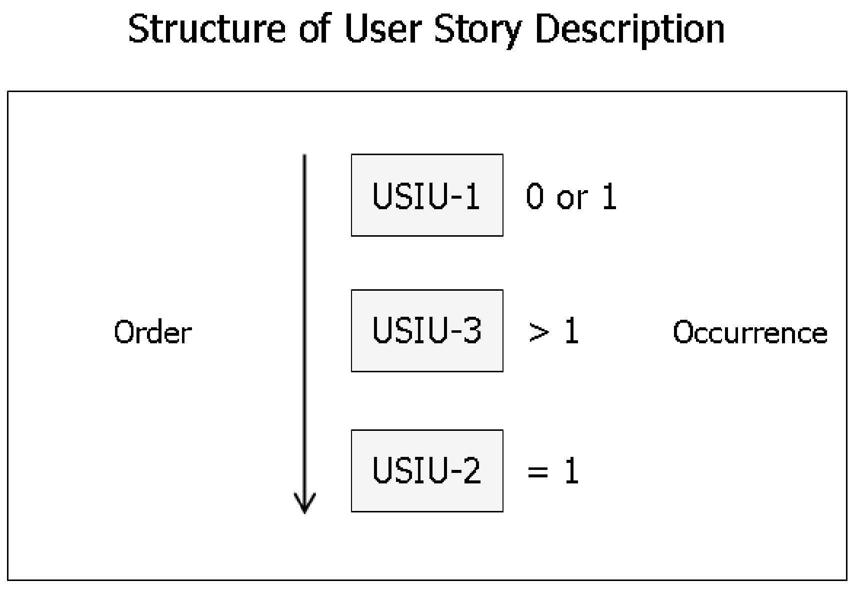


Figure 3.3. The physical and the logical structure of a user story description are governed by user story information units.

3.7. User Story Form

An arbitrary combination and permutation of user story information units may not be meaningful, and therefore may not be useful. For example, goal and value are two user story information units, as considered in Chapter 5. It is evident that a user story description with a goal, but without a value, is not meaningful.

Therefore, the interest in this thesis is in a meaningful collection of user story information units that appear together. This, in turn, leads to the notion of a user story form.

Definition 3.4 [User Story Form]. A prescription of a specific set of user story information units that are expected to appear in a user story description.

It follows from previous discussion and the definition of user story form that the order and occurrence of a specific set of user story information units in a given user story form is fixed.

A user story description, if structured, may have a user story form, as illustrated in Figure 3.4.

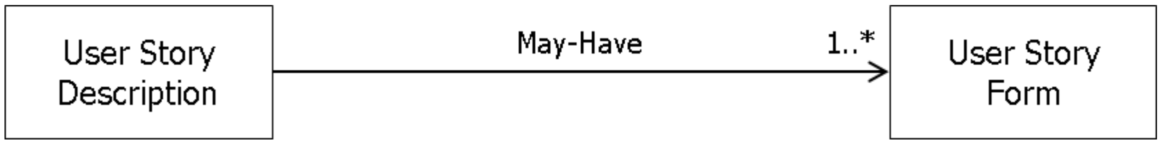


Figure 3.4. The relationship between user story form and user story information unit.

A user story form must consist of at least one user story information unit. There is currently no 'standard' list of user story information units. Therefore, it is possible to have a variety of user story forms, as illustrated in Figure 3.5.

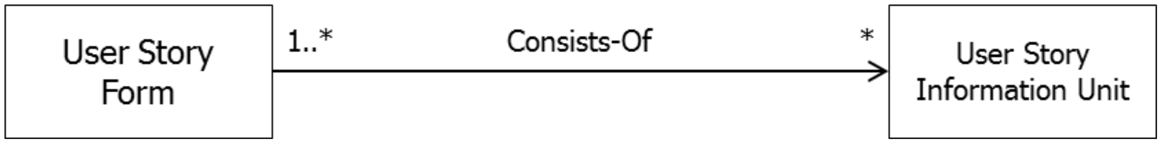


Figure 3.5. The relationship between user story form and user story information unit.

It follows from previous discussion that if a user story description is structured according to some user story form, then the statements therein correspond to one or more user story information units from a given collection, such as that of Chapter 5.

3.8. User Story Form and Process Maturity

The adoption and use by an organization of a user story form is conditional. There is, as illustrated in Figure 3.6, a direct relationship between the structure of a user story form and the level of process maturity of an organization.

For example, it can be expected that a user story form with an elaborate structure is applicable only to an organization with a high level of process maturity. The converse holds as well, that is, an organization with high level of process maturity can afford and aim to have a user story form with an elaborate structure.

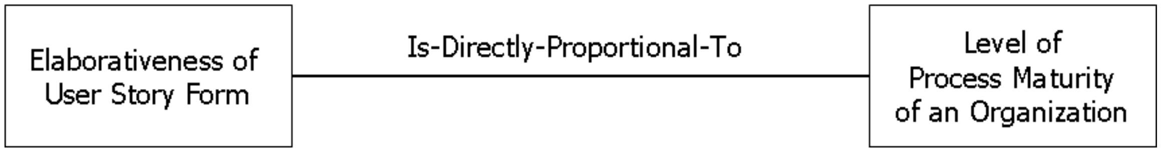

Elaborativeness of User Story Form
Is-Directly-Proportional-To
Level of Process Maturity of an Organization

Figure 3.6. The relationship between user story form and process maturity.

3.9. Limitations of USDM

The expectation of a rigorous structure on a user story description could be perceived as an impediment to creativity. In practice, an enforcement of structure invariably requires tool support. The reliance on a tool and the learning curve associated with the tool may not be acceptable to all.

3.10. Summary

A user story description is a way for making a user story explicit. In this thesis, the interest is in the physical and the logical structure of a user story description. A user story information unit provides a minimal starting point for such a structure. A user story description, if structured appropriately, typically consists of a number of user story information units. A user story form is about combination and permutation of user story information units, and it is possible to have multiple user story forms.

Chapter 4 A User Story Stakeholder Model

All the world's a stage, and all the men and women merely players.
They have their exits and entrances, and one man in his time plays many parts.
— William Shakespeare

It is expected that a user story, since its inception and for some reason, is relevant to some human [Kamthan, Shahmir, 2010]. The purpose of this chapter is to help identify and create an understanding of those who, in some manner, are directly involved with a user story, including its description. To do that, this chapter presents a conceptual model for stakeholders of a user story, namely USSM.

USSM is significant to markup languages such as USML. Indeed, in [Maler, El Andaloussi, 1996, Chapter 3], the need for identifying people, and their types of "interactions with documents" based on markup languages, has been emphasized.

4.1. Goals and Strategy

The goals of USSM are to create a basis for carrying out subsequent discussion regarding human involvement in user story engineering, in general, and pertaining to user story description, in particular. However, USSM is abstract, and does not consider personal characteristics of user story stakeholders.

USSM is based on the following strategy:

- **Select a Viewpoint.** A person may view a user story description from a certain perspective. There are number of possible perspectives, and the relationship between a person and a user story description must be based on a specific perspective.

- **Identify Roles.** A person plays a specific role with respect to a user story description. There are a number of such possible roles. The roles that are relevant to the scope of the thesis must be identified, and labeled (named).

- **Organize Roles.** The roles that are identified may, directly or indirectly, share a certain aspect. This commonality can be used to organize the roles in some manner, such as a hierarchy.

4.2. User Story Stakeholders

There are a number of definitions of a stakeholder, based on which the following is derived:

Definition 4.1 [User Story Stakeholder]. A person who has interest in a user story for some purpose.

In this thesis, user story stakeholder is a 'first-class' concept.

4.3. Situating User Story Stakeholders among Software Project Stakeholders

A user story stakeholder is a kind of software project stakeholder, as illustrated in Figure 4.1.

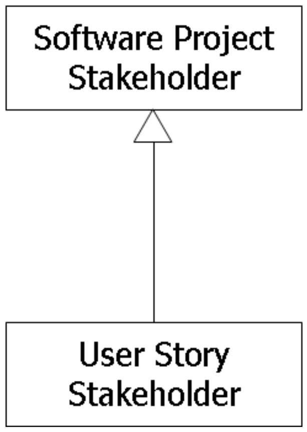


Figure 4.1. The relationship between software project stakeholder and user story stakeholder.

In [Power, 2010], ideas of stakeholder theory [Freeman, Harrison, Wicks, Parmar, Colle, 2010] are used to identify six “primary stakeholder groups” in a typical “agile product development organization”. These stakeholder groups are Product Owner Team, Product Delivery Team, Program Sponsor Team, Product Consumers, Product Council and Program Core Team. In such a case, the user story stakeholders are part of the Product Owner Team. This is illustrated in Figure 4.2.

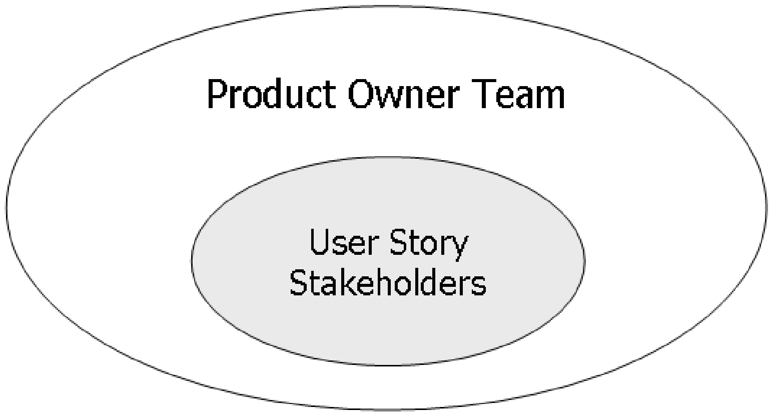


Figure 4.2. The relative positioning of user story stakeholders in an "agile product development organization".

Remark 4.1. It could be noted that, unlike a typical software project stakeholder model, USSM does not highlight any properties, such as importance or influence, of the user story stakeholders.

4.4. Classification of User Story Stakeholders

It is possible to have different viewpoints of a user story stakeholder. For the sake of this thesis, the developmental viewpoint is the most relevant among the possible viewpoints.

From a developmental viewpoint, the user story stakeholders can be classified into producers and consumers, as illustrated in Figure 4.3. This classification of stakeholders of a user story is role-based, and has a product-oriented view, the product being the user story itself.

There are number of reasons for a person to be interested in a user story. It is these reasons that inspire different possible roles. These roles are independent of any software development methodology, and therefore any agile methodology.

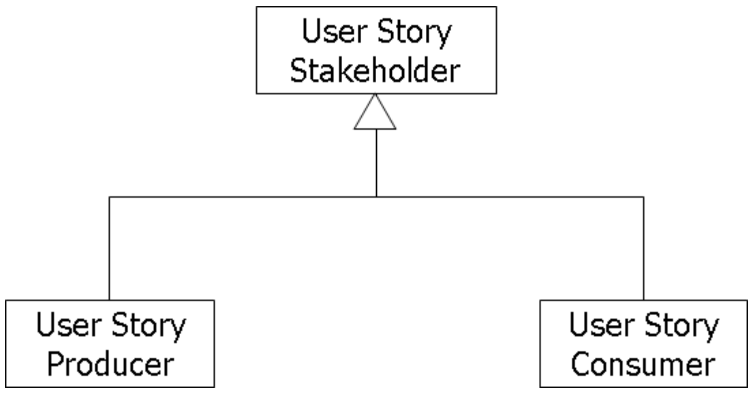


Figure 3.3. A high-level classification of user story stakeholders.

Remark 4.2. It is evident that the mapping between the set of roles and the set of user stakeholders is many-to-many. The same person can take upon different roles, and the same role can be taken upon by different persons.

Remark 4.3. A role-based classification of stakeholders has a rich history in software engineering, specifically, use case modeling [Adolph, Bramble, Cockburn, Pols, 2003; Jacobson, Christerson, Jonsson, Övergaard, 1992], organizational patterns [Coplien, 1994; Coplien, Harrison, 2005], and requirements engineering [Sharp, Galal, Finkelstein, 1999].

4.4.1. Classification of User Story Producers

The producers, as illustrated in Figure 4.4, are User Story Author, User Story Engineer, User Story Administrator, and User Story Evaluator.

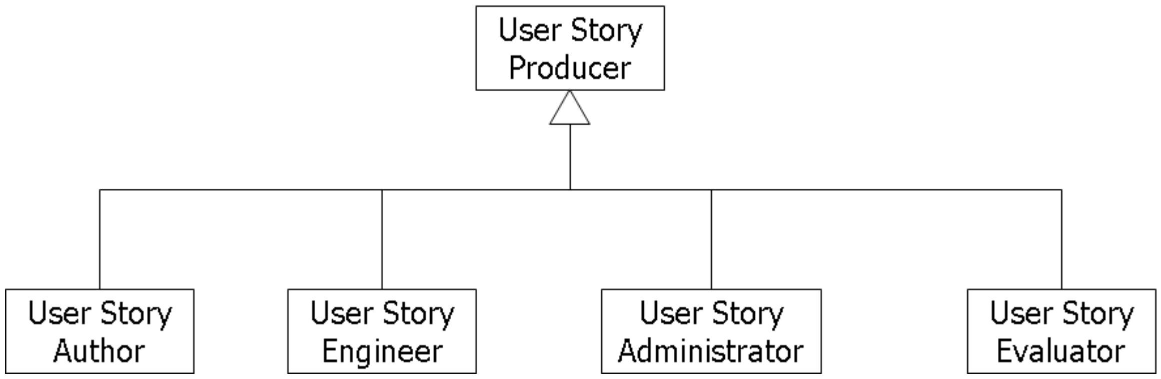


Figure 4.4. A classification of user story producers.

Definition 4.2 [User Story Author]. A person responsible for authoring a user story description.

For example, in Scrum, the Product Owner is one of the user story authors.

Definition 4.3 [User Story Engineer]. A person responsible for creating means for describing or creating means for processing user stories.

For example, a user story engineer is responsible for creating a markup language such as USML for representing user stories, or creating a transformer, as defined in Chapter 9, for converting user story descriptions from one format into another.

Definition 4.4 [User Story Administrator]. A person responsible for managing user stories.

For example, a user story administrator can be responsible for managing a Kanban Board[15] [Hiranabe, 2007], syndicating and disseminating user stories, or administering a User Story Management System (USMS).

Definition 4.5 [User Story Evaluator]. A person responsible for reviewing and providing feedback on a user story.

Remark 4.4. The need for reviewing user stories as a preventative means to remove defects has been emphasized [Jones, Bonsignour, 2012, Page 60].

4.4.2. Classification of User Story Consumers

The consumers, as illustrated in Figure 4.5, are User Story Browser, User Story Reader, User Story User, and User Story Evaluator.

[15] In Japanese, kan mean a 'signal' and ban means a 'card' or a 'board'. The term kanban means a signal board or a billboard. The purpose of kanban, as a signaling system, is to trigger action.

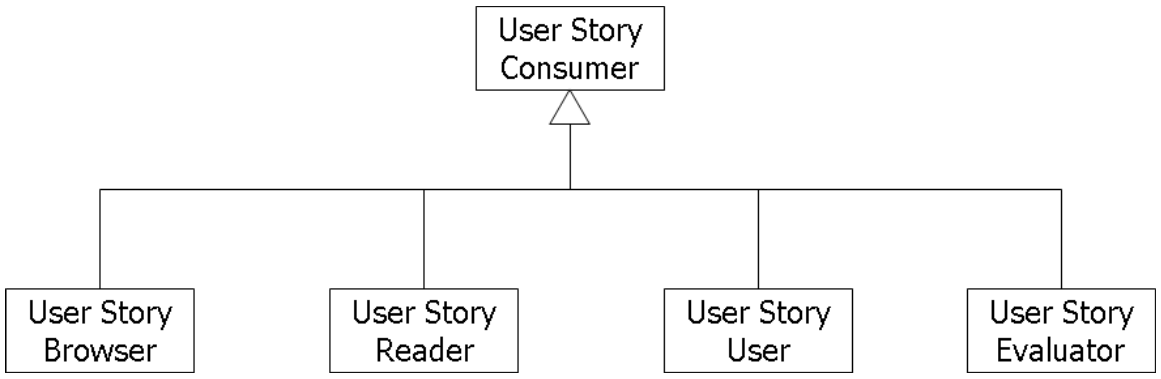


Figure 4.5. A classification of user story consumers.

Definition 4.6 [User Story Browser]. A person targeted for browsing (or scanning) a user story for some purpose.

The notion of browsing has gained special attention since the inception of the Web. There can be a number of reasons for browsing including, but not limited to, finding information. For example, a person, while reviewing a user story, may scan the acceptance criteria to see if a certain test has been included.

Definition 4.7 [User Story Reader]. A person targeted for reading a user story.

It is evident that a user story must be read before it is understood and, if deemed suitable, used subsequently.

A user story reader is not necessarily a user story user. For example, upon review, a user story may be rejected, and never be used. For another example, a person may read a user story from a previous software project seeking candidates/opportunities for potential reuse, however, for some reason, may decide not to use it.

Definition 4.8 [User Story User]. A person targeted for using a user story for some purpose.

There are number of potential uses of a user story. For example, a project manager, or a person in a similar role, may use the collection of user stories for planning a software project [Cohn, 2005], or a user interface designer may use part of the collection to construct a high-fidelity prototype.

4.4.3. Classification of User Story Authors

In this thesis, there is special interest in the notion of a user story author, and it therefore deserves further attention. It is possible to classify a user story author on different facets, as illustrated in Figures 4.6 and 4.7.

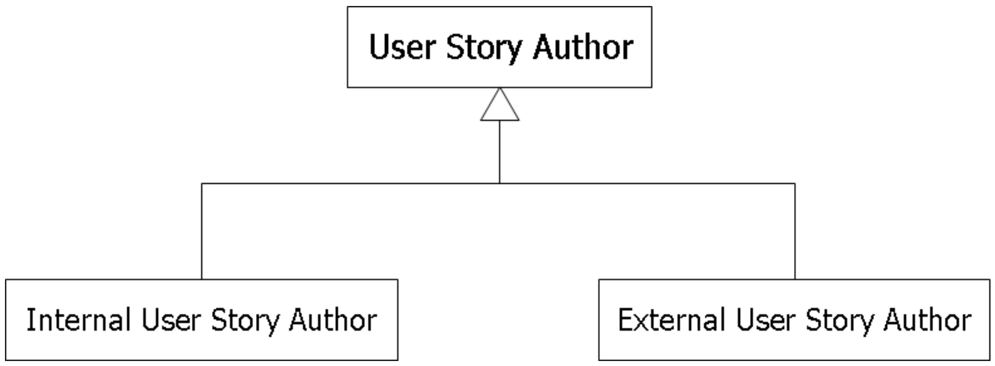


Figure 4.6. A classification of user story authors based on their relative positioning with respect to the organization corresponding to the software project.

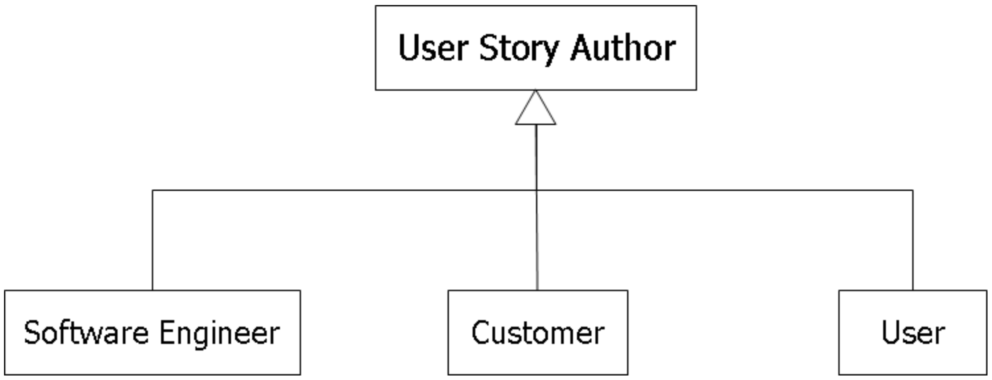


Figure 4.7. A classification of user story authors based on their role with respect to the software product.

It follows from Figure 4.7 that a user can be a user story author. For example, a representative user can be such a user. This is ‘empowering’ users, that is, involving those people in development of a product who will end up using that product.

Remark 4.5. The notion of ‘empowering’ users goes back at least to the use of patterns for urban architecture and design [Alexander, 1979].

4.5. Difference between User, Customer, and Software Engineer

Figure 4.8 emphasizes the fact that user, customer, and software engineer are different, and each is a source of certain type of knowledge necessary for software development.

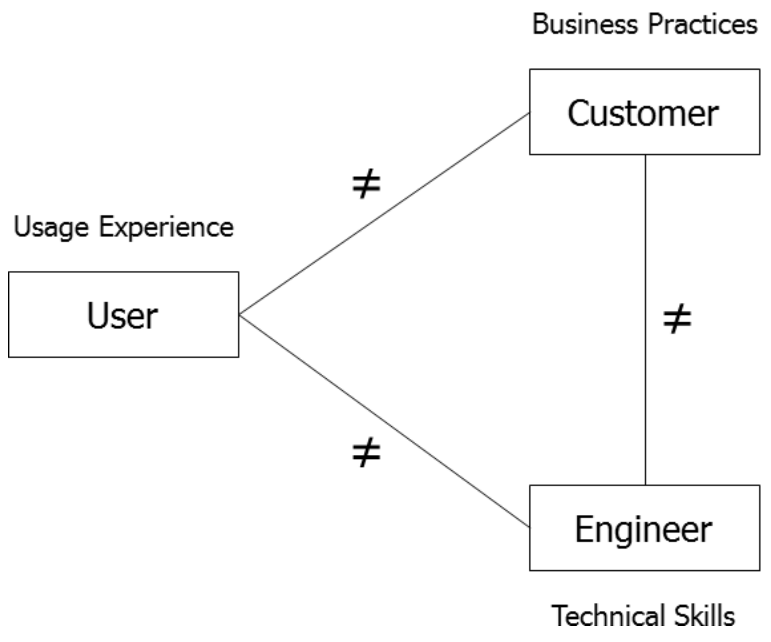


Figure 4.8. A user, customer, and software engineer are pairwise different.

4.5.1. User versus Customer

There are a number of definitions of a customer and user, including the following, respectively:

Definition 4.9 [Customer] [IEEE, 1998]. [A] person, or persons, who pay for the [software system] and usually (but not necessarily) decide the requirements.

Definition 4.10 [User] [ISO, 1998]. [A] person who interacts with the [software system].

In this thesis, customer and user are not synonymous, and there not interchangeable.

It follows from the above definitions that a customer may not be user of a software system. A customer is expected to know the needs of a user, and by able to communicate those needs to a software engineer. However, a customer is not a substitute for a user.

4.5.2. User versus Software Engineer

The mental models[16] [Young, 2008] of users and software engineers are different [Colborne, 2010; McKay, 1999, Chapter 6, Chapter 9; Nielsen, 1993, Page 12]. It is possible that during interviews, users might say or express one thing but actually do something else, also known as 'say-do' problem. The reason could be that certain type of knowledge (namely, tacit knowledge [Polanyi, 1966]) can not be articulated.

Indeed, the construction of prototypes, followed by feedback from users, allows software engineers to compare the mental models.

4.5.3. Customer versus Software Engineer

A customer is expected to be knowledgeable about, perhaps even an expert on, the application (or problem) domain. A software engineer may not have any knowledge, or only passing knowledge, of the application (or problem) domain. The mental models of customers and software engineers are different. It is possible that during interviews, a

[16] A mental model is a [description] of a [person's] thought process about how something works in the real world [Wikipedia].

customer might demand/expect one thing but, for a number of reasons, may change their mind upon presentation of the result.

For example, a customer may have initially demanded that the electronic commerce system under development must display the logos of the project sponsors. However, the organizational policy may change during development, or that doing so may appear out of context with the rest of the system, and the customer may decide to retract.

4.5.4. Implications of the Difference between User, Customer, and Software Engineer on Authoring User Stories

The differences between user, customer, and software engineer, as per the knowledge that each carries, have notable implications towards developing user stories.

It can not be expected that software engineers are completely aware of the application (or problem) domain, especially if it is new, or completely aware of the users' needs. It is also known that customer/users do not always know what they want [Colborne, 2010], and are not always right [Nielsen, 1993; Nuseibeh, 1996]. Furthermore, it can not be reasonably expected that customer/users are knowledgeable or skilled in expressing software requirements, specifically that they are aware of the issues related to software requirements quality, in general, and user story quality, in particular. Indeed, it has been pointed out that users can 'forget' non-functional aspects of user stories [Rodríguez, Yagüe, Alarcón, Garbajosa, 2009].

Therefore, the input of only software engineers, or only customer/users, is insufficient for authoring a user story. The involvement of all, evidently, requires negotiation. This is in contrast with the state-of-the-art that suggests that only customer/users, not software engineers, must author user stories [Alexander, Maiden, 2004; Cohn, 2004; Lui, Chan, 2008].

4.6. User Story Form and User Story Stakeholders

The presence of a user story form has both advantages and disadvantages for user story stakeholders. These are elaborated in the following.

- **User Story Form and User Story Administrator.** A user story form must include a certain number of user story information units. This helps the machine processing of a user story description in a variety of ways. For example, a user story description can be manipulated for different purposes, such as, extraction of text to create a user story thumbnail. This aids a user story administrator.

- **User Story Form and User Story Author.** A user story form imposes a definite structure on a user story description. This aids a user story author. For example, a user story author can use a specific user story form as a template for authoring user stories for a given software project. (This is a manifestation of the use of the DOCUMENT TEMPLATES pattern [Rüping, 2003].) The presence of multiple user story forms gives user story authors a choice. However, a commitment to a particular user story form may be perceived by user story authors as an impediment to ‘freedom’ of authoring, and thereby limiting their creativity.

- **User Story Form and User Story Reader.** If adopted and applied to all the user stories in a collection of user stories, a user story form can lend a consistent structure to the descriptions of user stories in that collection. This, in turn, contributes to familiarity of the structure on part of a user story consumer, and familiarity aids understandability. For example, upon reading different descriptions of user stories that belong to the same collection, a user story reader can be assured that certain information units are present, in a certain order, in each user story description.

- **User Story Form and User Story Browser.** A user story form can aid browsing. For example, if presented appropriately, a user story description can be scanned and skipped to desirable areas, such as to specific user story information units. This aids a user story browser.

4.7. Limitations of USSM

USSM is not exhaustive in the sense that does not take into consideration all possible stakeholders of a user story.

USSM also does not consider relevant relationships between the stakeholders of a user story. Resource Description Framework (RDF) and Web Ontology Language (OWL) are general-purpose ontology specification languages. There are RDF/OWL-based ontologies such as Friend of a Friend (FOAF)[17] and RELATIONSHIP[18] for describing different types of relationships between people.

Figure 4.9 illustrates certain types of relationships that can form between people, and between people and a user story. (The `rel:collaboratesWith` is a concept from the RELATIONSHIP ontology.)

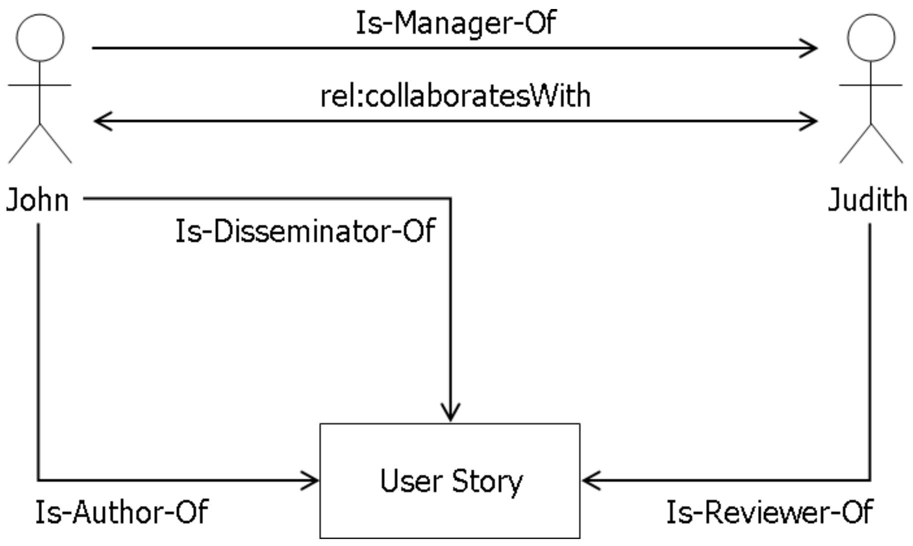


Figure 4.9. An example of possible relationships (1) between two concrete stakeholders of a user story, and (2) between each of the concrete stakeholders and the user story.

4.8. Summary

There are people who have stake in a user story, like they have in other software project artifact. The people related to a user story must be identified and classified, and USSM provides one way of doing so. There are different kinds of user story authors. A customer, user, and software engineer are pairwise different, which is relevant when

[17] URL: http://xmlns.com/foaf/spec/ .

authoring user stories. A user story form has implications towards user story stakeholders. In certain situations, such as on a social network [Palfrey, Gasser, 2008], the relationships between the stakeholders of a user story could be of interest.

[18] URL: http://vocab.org/relationship/ .

Chapter 5 A User Story Information Model

[T]he wealth of information means a dearth of something else: a scarcity of whatever it is that information consumes [and] it consumes the attention of its recipients.
— Herbert A. Simon

The type of information that can occur in a user story description needs to be identified and, if necessary, organized. To do that, this chapter presents a conceptual model for information contained in a user story, namely USIM. The user story information units in USIM are supported by rationale and a systematic literature review.

USIM is an essential prerequisite to USML. Indeed, the significance of information modeling in context of markup languages has been pointed out previously [Carlson, 2001; Maler, El Andaloussi, 1996, Section 5.2.3; Linton, 2007, Page 9].

5.1. Goals and Strategy

The goals of USIM are to lend support to USDM, and to be useful as an input to USRM and USML. The current coverage of user stories in literature is often aligned with one of the agile methodologies, namely XP or Scrum. USIM aims to be independent of any specific software development methodology.

USIM is based on the following strategy:

- **Concretize User Story Information Units.** The introduction of a user story information unit in Chapter 3 is theoretical and abstract and, to be useful, needs to be practical and concrete. To do that, USIM includes a concrete list of user story information units, and an organization of these user story information units.

- **Rationalize User Story Information Units.** USIM is largely based on discovery, not invention. The inclusion of each user story information unit in USIM is rationalized, using technical argument and current literature, as necessary.

5.2. The User Story Ecosystem

The development of every software system takes place in an ecosystem [Brooks, 1987], and the same applies to agile software development [Highsmith, 2002].

The user story ecosystem consists of animate or inanimate things related to the notion of a user story. For example, a person negotiating a user story, the organization responsible for the software project in which a user story appears, and the conditions under which a user story may be used, are all part of the user story ecosystem.

In this thesis, the relevant user story information units are elicited from the user story ecosystem, as illustrated in Figure 5.1.

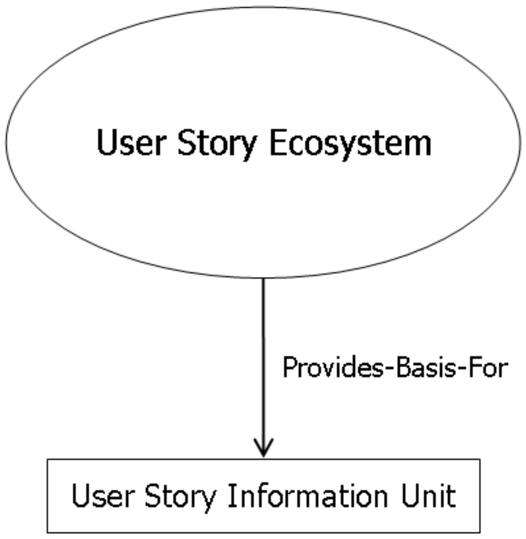


Figure 5.1. The user story information units are derived from the user story ecosystem.

The user story ecosystem needs scope. There is currently no body of knowledge, reference model, or 'standard' for user stories. Therefore, this thesis relies on a selected number of resources that can be considered as noteworthy, if not authoritative. Furthermore, if deemed necessary, this thesis supplements the knowledge from these resources with new concepts.

5.2.1. Systematic Literature Review as a Basis for User Story Information Units

In USIM, each information unit is supported by a rationale for inclusion, and optionally by results of a systematic literature review using some of the guidelines of [Kitchenham, Charters, 2007].

For the sake of this thesis, literature consists of resources. A resource could be in form of a book or a non-book (including, but not limited to, an article published in a conference, magazine, or journal).

The literature review is based on the following process:

1. **Step 1: Discovering Resources.** For discovery of relevant literature, the means deployed were navigating and searching on the Web using local and global search engines[19]. Concordia University Libraries[20] was used as the primary local search engine and Google Scholar[21] was used as the primary global search engine. For queries, "`user story`" was used as the primary query string and "`agile methodology`" was used as the secondary query string. The resources were also discovered by navigating through 'related articles' that are displayed alongside the 'full text' of an article on certain portals (such as SpringerLink and ScienceDirect).

2. **Step 2: Selecting Resources.** A resource was discarded if it was not written in English. A non-book resource was considered a candidate if its coverage on user stories was substantial (at least a paragraph), and (1) it was available openly at no cost to the commons, or (2) it was available at no cost to the Concordia University community through the Concordia University Libraries. A non-book resource was discarded if no evidence of a formal review for it was found.

[19] A local search engine is organization-specific, and its use is restricted; a global search engine is open to public.
[20] URL: http://library.concordia.ca/ .
[21] URL: http://scholar.google.ca/ .

3. **Step 3: Including Resources.** The following format is used to relate a user story information unit and results of literature review:

USIU	**Frequency**	**Resource(s)**
Name	Number	Citation(s), where | Citation(s) | = Number

5.3. Organization of User Story Information Units

In the current literature, there is no single encompassing term for a collection of user stories.

Definition 5.1 [User Story Book]. A collection of user stories for a specific software project.

The use of the term 'user story book' is metaphoric.

Remark 5.1. A user story book is not a 'specification' in the sense of [Bang, 2007; Monochristou, Vlachopoulou, 2007]. For example, a user story book is meant to be negotiable, not contractual.

The user story information units presented in the rest of the chapter are organized into two categories:

1. **Information Units for a User Story Book.** There is certain information that is general and applies to all user stories in a collection, that is, it applies to the user story book. For example, the software project to which the user stories belong as a collective is such information.

2. **Information Units for a User Story.** There is certain information that is specific to each user story. For example, the iteration to which a given user story belongs is such information.

5.4. Information Units for a User Story Book

The information units related to a user story book, as it pertains to this thesis, and ordered alphabetically, are:

- Domain
- License
- Maturity
- Methodology
- Organization
- Project

USIU: Domain

The notion of a user story is independent of any particular application (or problem) domain. However, a user story description encapsulates information of some application (or problem) domain. This domain must be made explicit.

Rationale. A user story description is about a problem, and therefore must include terms specific to application (or problem) domain. For example, there are reports of deployment of user stories in organizations providing products/services related to computer games [Keith, 2010, Chapter 5] and student welfare [Bang, 2007]. In such cases, the domain would be Computer Games and Student Welfare, respectively.

USIU: License

The development and release of a user story book is subject to legal constraints that need to be highlighted. To do that, a user story book can be associated with a license declaration. The license corresponding to a license declaration applies to a user story book, and is inherited by each user story therein.

Let P_1 be a software project that has user story book USB_1 with license L_1, and let P_2 be a software project that has user story book USB_2 with license L_2. Let US_1 be a user story from USB_1. If US_1 is reused in P_2, that is, included in USB_2, then it is assumed that L_2 overrides L_1 and applies to US_1.

Rationale. In this thesis, it is assumed that a user story, like other software process artifacts, is an intellectual property [Rosen, 2004, Chapter 2] to which its authors have rights. The availability of a user story to others is conditional, where the conditions can either be implicit or explicit. A license declaration makes the conditions explicit.

Remark 5.2. There are different kinds of software licenses [Rosen, 2004, Chapter 4; St. Laurent, 2004]. There are guidelines [Rosen, 2004, Chapter 10; St. Laurent, 2004, Section 7.3] and patterns [Kaminski, Perry, 2007; Perry, Kaminski, 2005] for selecting an appropriate license. This thesis does not recommend or endorse any specific license.

USIU: Maturity

A user story description is a result of some user story development process. The number and kind of activities included in a user story development process can vary and, as a result, the kind of user story descriptions can also vary. A user story description, depending on the information it contains, can belong to different levels of maturity.

Rationale. The demands in rigor placed on user stories as practiced in Scrum are higher than those in XP. These demands are related to process maturity. For example, in [Kähkönen, Abrahamsson, 2004], user stories pertaining to XP are placed at Maturity Level 2 of the CMMI Version 1.1; in [Diaz, Garbajosa, Calvo-Manzano, 2009], user stories pertaining to Scrum are placed at Maturity Level 3 of the CMMI Version 1.2. In [Patel, Ramachandran, 2009b], a Story Card Maturity Model (SMM) is presented. SMM has four maturity levels: Initial, Repeatable, Defined, and Managed. There are key process areas corresponding to each maturity level, and there are user story-related activities corresponding to each process area. However, at times, the notions of a user story and that of a user story card are mixed.

Literature Review.

USIU	Frequency	Resource(s)
Maturity	1	[Patel, Ramachandran, 2009b]

Remark 5.3. There have been a number of proposals for requirements engineering process maturity models [Li, 2005]. These proposals vary with respect to their alignment with software process maturity models such as CMMI. However, in these models, user stories are not considered.

USIU: Methodology

The notion of a user story is independent of any software development methodology. However, it can be expected that the development and description of a user story will usually take place within the realm of some software development methodology.

Rationale. There is explicit support for user stories in BDD, Scrum, XP, UCAP, and U-SCRUM, as pointed out in Chapter 2. Indeed, the notion of a user story was first introduced in XP [Beck, 2000].

Literature Review.

USIU	Frequency	Resource(s)
Methodology	3	[Beyer, 2010] [Cohn, 2004] [Williams, 2010]

USIU: Organization

It is expected that a user story book is being developed under the auspices of some organization. An organization may be a commercial or non-commercial entity (say, a University).

Rationale. The development of a user story book, is general, and user stories, in particular, is not an effort that is isolated from the rest of the world. Indeed, it is initiated by some entity. In a user story description, the information of such an organization must be explicit.

USIU: Project

A project is a temporary endeavor undertaken to achieve a specific and unique product or service [PMI, 2008]. It is expected that a user story book is being developed being developed for a software project.

Rationale. The development of a user story book, is general, and user stories, in particular, is part of a planned initiative. A software project is such an initiative. In a user story description, the information of such a software project must be explicit.

5.5. Information Units for a User Story

The information units related to a user story, as it pertains to this thesis, and ordered alphabetically, are:

- Acceptance Criteria
- Constraint
- Context of Use
- Estimate
- Goal
- Iteration
- Name
- Note
- Persona
- Priority
- Role
- Statement
- Status
- Theme

- Value

USIU: Acceptance Criteria

There are a number of definitions of acceptance criteria, including the following, which is a minor adaptation of the source.

Definition 5.2 [Acceptance Criteria] [IEEE, 1990]. The criteria that a [software] system must satisfy in order to be accepted by customer or user.

The acceptance criteria for a user story are a collection of conditions under which an implementation of that user story is accepted by a customer or user. These conditions can be expressed as a list one or more tests. Indeed, the acceptance criteria are a prelude to acceptance testing.

Definition 5.3 [Acceptance Testing] [IEEE, 2005].

(A) Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.

(B) Formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component.

Rationale. The closeness of requirements and tests, and the significance of expressing the tests as early as possible, has been discussed previously [Martin, Melnik, 2008]. Indeed, the tests clarify and amplify a user story. Testability is among the characteristics of a user story [Alexander, Maiden, 2004, Page 271; Jeffries, 2001; Patel, Ramachandran, 2009a; Wake, 2002; Winbladh, Ziv, Richardson, 2008]. The activity of authoring acceptance tests for a user story is included in the SMM Level 3, Key Process Area 3.2 and in the SMM Level 4, Key Process Area 4.6 [Patel, Ramachandran, 2009b].

Literature Review.

USIU	**Frequency**	**Resource(s)**
Acceptance Criteria	1	[Cohn, 2004]

Remark 5.4. In recent years, acceptance testing has found support in agile methodologies in the form of Acceptance Test-Driven Development (ATDD) [Adzic, 2009; Madeyski, 2010; Pugh, 2011]. In both XP and Scrum, automation of acceptance testing is encouraged. However, it is not always possible to automate certain tests (of a user story).

USIU: Constraint

A user story statement is not necessarily absolute, and therefore must be situated in its context wherever necessary. There may be one or more constraints on the user story statement. These are usually associated with software quality-related requirements.

Rationale. The activity of "identifying non-functional requirements" is included in the SMM Level 2, Key Process Area 2.1 and reviewing "non-functional requirements" is included in the SMM Level 3, Key Process Area 3.5 [Patel, Ramachandran, 2009b].

Literature Review.

USIU	Frequency	Resource(s)
Constraint	1	[Cohn, 2004]

Remark 5.5. The constraints provided in the aforementioned manner are local, not global. (A global constraint applies to the entire software system.) In [Cohn, 2004, Page 77], it has been suggested that global constraints should be listed separately on a constraint card. In [Rodríguez, Yagüe, Alarcón, Garbajosa, 2009], the notion of 'system story' is introduced to document global constraints.

USIU: Context of Use

The following definition of context of use has its origins in usability:

Definition 5.4 [Context of Use] [ISO, 1998a]. A description of the users, tasks and equipment (hardware, software and materials), and the physical and social environments in which a product is used.

The use of any software system by a human (user) takes place in a certain context. The information on the context of use needs to be explicit in a user story description.

Rationale. The significance of context of use in ubiquitous computing [Dey, 2001] and in relation to usability [Thomas, Bevan, 1996] is known. The RESPECT Project [Maguire, Kirakowski, Vereker, 1998] provides a framework for user requirements specification, and includes a comprehensive treatment of context of use.

Remark 5.6. It could be noted that an accessibility- or usability-related constraint associated with a user story depends on the context of use. For the sake of this thesis, accessibility and usability are defined as per [ISO, 2008] and [ISO/IEC, 2001], respectively. For example, consider a user story about a student searching a human resources information system for employment opportunities. The number of search results to be presented depends on the device being used. A mobile phone tends to have a screen with relatively small horizontal and vertical dimensions compared to that of a notebook computer. Therefore, a large number of search results presented on a screen of a mobile phone can lead to considerable vertical scrolling. This is prohibitive to operability, one of the dimensions of usability [ISO/IEC, 2001].

A further discussion of context of use, including formulation of a suitable context of use model, is beyond the scope of this thesis.

USIU: Estimate

A unique aspect of a user story, as implied by its definition in Chapter 2, is the inclusion of an estimate upfront.

Definition 5.5 [Estimate] [Fenton, Pfleeger, 1997, Page 428]. An estimate is a prediction that is equally likely to be above or below the actual result.

The approaches for user story estimation can be subjective or objective. The subjective approaches for user story estimation currently include use of Analogy, Expert Judgment, Planning Poker, and Silent Grouping [Cohn, 2004; Cohn, 2005; Power, 2011].

The estimate could be about size or time. A Story Point is used as a metric for size (that is, amount of work it will take to complete a user story). It has been pointed out [Cohn, 2004; Cohn, 2005] that, as the size gets larger, the estimates get poorer. Therefore, the suggested range for story points is either a Fibonacci sequence or a geometric sequence (specifically, powers of 2). An Ideal Day (of work) is used as a metric for time.

A story point is relative, while an ideal day is absolute. It is possible to convert story points to ideal days, for example, by setting 1 story point = 1 ideal day (of work).

The objective approaches for user story estimation currently include the use of Common Software Measurement International Consortium Functional Size Measurement

(COSMIC FSM)[22] [Fehlmann, Santillo, 2010; Hussain, Kosseim, Ormandjieva, 2010; Rule, 2009]. The estimate in COSMIC FSM is about functional size. A COSMIC Function Point is used as a metric for functional size.

Rationale. Estimatability is among the characteristics of a user story [Wake, 2002]. The activity of estimating user stories is included in the SMM Level 4, Key Process Area 4.2 [Patel, Ramachandran, 2009b].

Literature Review.

USIU	Frequency	Resource(s)
Estimate	7	[Beck, 2000] [Beck, Andres, 2005] [Cohn, 2004] [Patel, Ramachandran, 2009b] [Rule, 2009] [Wake, 2002]

USIU: Goal

There are a number of definitions of a goal, including the following, which is from the perspective of use (of a software system):

Definition 5.6 [Goal] [ISO, 1998]. An intended outcome of user interaction with a product.

[22] URL: http://www.cosmicon.com/ .

Rationale. A user has a goal for using a software system. This goal may be implicit or explicit. A user story statement can make the goal explicit.

Literature Review.

USIU	Frequency	Resource(s)
Goal	3	[Cohn, 2004] [Leffingwell, 2011] [Monochristou, Vlachopoulou, 2007]

USIU: Iteration

There are a number of definitions of iteration [Larman, Basili, 2003; Williams, 2010], including the following:

Definition 5.7 [Iteration]. A collection of activities and a specific length of time dedicated towards developing an approximation of the software system.

The length of an iteration may be a constant or a variable. An iteration is time-boxed if its length is pre-determined, that is, a constant. In agile methodologies, such as XP and Scrum, the iterations are time-boxed [Williams, 2010].

Rationale. The notion of iteration is motivated by the acknowledgement that it is not possible to anticipate everything in software development [Parnas, Clements, 1986; Stober, Hansmann, 2010, Chapter 1]. The notion is especially relevant to software development methodologies that are evolutionary in nature, including agile methodologies. A user story is incepted for and must be completed in a single iteration, an indication of which [Sliger, Broderick, 2008, Appendix B] can be useful for certain user story consumers.

Literature Review.

USIU	Frequency	Resource(s)
Iteration	9	[Beck, 2000] [Beck, Andres, 2005] [Beyer, 2010] [Cohn, 2004] [Leffingwell, 2011] [Monochristou, Vlachopoulou, 2007] [Sliger, Broderick, 2008] [Stober, Hansmann, 2010] [Williams, 2010]

Proposition 5.1 [Cardinality of User Stories]. Let P be an agile project that follows an agile methodology A. Let there be n iterations I_1, …, I_n in the software process corresponding to A. Let m_i be the number of user stories in iteration I_i, for some i, $1 \leq i \leq n$. Let $m \equiv \max \{m_i, 1 \leq i \leq n\}$. Then, n is the lower bound and $m \cdot n$ is the upper bound on the number of user stories in P.

Proof. Let x be the number of user stories in P. There must be at least 1 user story per iteration, and so $n \leq x$. It is evident that $x \leq m \cdot n$. Therefore, $n \leq x \leq m \cdot n$. □

USIU: Name

The name of a user story is shorthand for the user story statement of a given user story.

Rationale. The name of a user story is significant for a number of reasons:

- **Communication.** A project team may use the names of user stories to communicate. A user story statement can, in certain cases, be rather lengthy for stakeholders to remember for the purpose of communication. The Sapir-Whorf Hypothesis suggests that defining a term for a set of concepts enables people to think and talk about them.

- **Organization.** A user story statement may be rather long to be readily included in a concept map (or mind map) or placed on a sticky note, for some purpose, such as understanding or organization. The collection of user story statements, if large in number, may also be difficult to fit on a flipchart. In such cases, a user story name can be useful.

- **Location.** An information scent [Pirolli, 2007] is a cue to a person (that is, information seeker) in an information environment that indicates that the information environment has the information the person seeks. A user story name is information scent. A user story browser may attempt to locate user stories using their respective names. A computer program may index user stories or, upon request, locate user stories using their respective names.

Literature Review.

USIU	Frequency	Resource(s)
Name	2	[Beck, 2000] [Cohn, 2004]

Remark 5.7. The significance and implications of labeling in the context of information design have been highlighted previously [Morville, Rosenfeld, 2006, Section 6.1].

Remark 5.8. There are similarities between naming a user story and naming a pattern [Kamthan, 2011a; Meszaros, Doble, 1998]. However, a discussion of the issue of the 'quality' of a user story name is beyond the scope of this thesis.

Remark 5.9. If a use case model has been constructed, then the name of a use case can serve as the name for the corresponding epic.

Remark 5.10. The idea of naming need not be exclusive to a user story statement. Indeed, a test, if deemed long, could be named.

USIU: Note

A user story reflects shared understanding as a result of conversation, followed by negotiation with the customer or user. This fact must be made explicit wherever necessary.

A note can augment a user story in a number of ways. It can contain open, unresolved issues pertaining to a user story. In this sense, a note is reminiscent of the 'TBD' in an SRS [IEEE, 1998]. For example, a note may be about a pending issue that needs to be conferred with the customer or user before a decision takes place.

A note can also act as a placeholder for miscellaneous information, that is, information otherwise not included elsewhere in the user story description. In this sense, a note is reminiscent of the UML Note element.

Rationale. Negotiability is among the characteristics of a user story [Wake, 2002]. The concept of a note is introduced in [Cohn, 2004].

Literature Review.

USIU	Frequency	Resource(s)
Note	1	[Cohn, 2004]

USIU: Persona

There are different types of user models [Junior, Filgueiras, 2005], one of which is a persona.

There are a number of definitions of a persona, including the following:

Definition 5.8 [Persona]. A fictional but realistic user of a software system.

The term ‘persona’ is derived from the term ‘personification’. It has origins in cognitive psychology [Jung, 1971] and user-centered approach to interaction design [Cooper, Reimann, Cronin, 2007].

Rationale. It is known that a persona can help prevent 'self-referential design[23]', or more generally, false consensus effect.

Literature Review.

USIU	Frequency	Resource(s)
Persona	3	[Cohn, 2004] [Leffingwell, 2011] [Singh, 2008]

Remark 5.11. The U-SCRUM methodology [Singh, 2008], an extension of Scrum in the direction of usability, uses personas.

Remark 5.12. A negative persona is a fictional but realistic negative user of a software system. It may be relatively difficult to elicit negative personas via ethnography. Therefore, it is likely that a negative persona is based on introspection.

USIU: Priority

A prioritization is a kind of ordering, or ranking.

The prioritization of software requirements has been studied extensively [Berander, Andrews, 2005; IEEE, 1998; Wiegers, 2003].

[23] A software engineer may inadvertently project his/her own mental model on the software system that may be different from that of the target users.

The prioritization of user stories is based on the assumption that the user stories are comparable with respect to some property, and are on an ordinal scale [Fenton, Pfleeger, 1997, Section 2.3.2] with respect to that property. For example, such a property could be perceived degree of relevancy (or value) to users, risk, relevancy to other (dependent) user stories, and so on. It has been pointed out that prioritization aims to maximize value and minimize risk [Anderson, Schragenheim, 2004; Bakalova, Daneva, Herrmann, Wieringa, 2011].

The prioritization of user stories assists in the selection of user stories for a specific purpose. For example, user stories may need to be prioritized so that the most significant ones are met by the earliest product releases.

The current approaches for prioritizing software requirements vary in a number of ways, including their rigor and sophistication. In [Wiegers, 2003], the 'High-Medium-Low' approach is used for prioritizing software requirements. MoSCoW stands for MUST, SHOULD, COULD, and WOULD, and the approach has its origins in the Dynamic Systems Development Method (DSDM) [Stapleton, 2003], an agile methodology. In [Cohn, 2004], the MoSCoW approach is used for prioritizing user stories.

In this thesis, the 'High-Medium-Low' approach is called the Basic approach. Figure 5.2 illustrates a scheme for prioritizing user stories as 'High', 'Medium', and 'Low' using two different dimensions, namely Value and Risk.

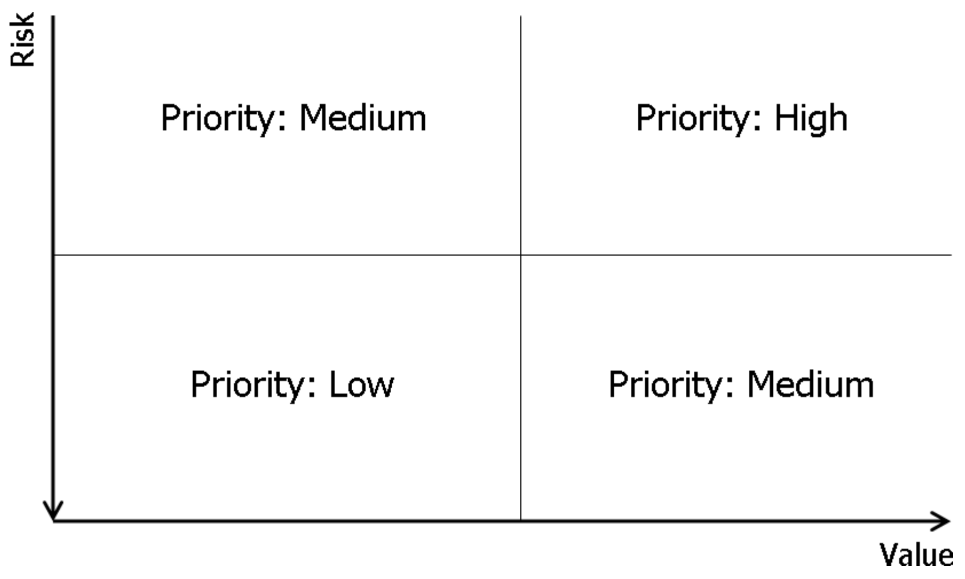


Figure 5.2. The prioritization of user stories on the Value-Risk plane.

A priority could be absolute or relative. For example, priority based on the MoSCoW approach is absolute, and priority based on the Basic approach is relative.

Rationale. Prioritizability is among the characteristics of a user story [Alexander, Maiden, 2004, Page 271]. The activity of "prioritization" of user stories is included in the SMM Level 3, Key Process Area 3.3 [Patel, Ramachandran, 2009b].

Literature Review.

USIU	Frequency	Resource(s)
Priority	7	[Alexander, Maiden, 2004, Page 271] [Anderson, Schragenheim, 2004] [Beck, 2000] [Cohn, 2004] [Collier, 2012] [Leffingwell, 2011] [Patel, Ramachandran, 2009b]

USIU: Role

There are different types of user models [Junior, Filgueiras, 2005], one of which is a (user) role.

Definition 5.9 [(User) Role]. A fictional character based on possible interactions with a software system.

The notion of a (user) role has its origins in usage-centered approach to interaction design [Constantine, Lockwood, 1999].

Rationale. A user, by virtue of the explicit mention of role, is 'first-class' in a user story statement. (This gives credence to the claim that a user story is a kind of user requirement.)

Literature Review.

USIU	Frequency	Resource(s)
Role	2	[Cohn, 2004] [Leffingwell, 2011]

Remark 5.13. A (user) role is abstract, while a persona is concrete. A persona and a (user) role can be seen as complementary. Indeed, a persona can be deployed for a verification of a (user) role: if no persona can be found for a given (user) role, then perhaps that user role should be deleted from the collection.

Remark 5.14. A user engaging in a negative use is called a negative user [Courage, Baxter, 2005]. A negative user story, as defined in Remark 5.16, has a negative (user) role. For example, a 'Malicious Hacker' is a negative (user) role.

USIU: Statement

A user story statement could be structured by basing it upon the primitive questions of who, what, and why. These can, in turn, be mapped respectively to a (user) role, a (user) goal, and a specific value to the (user) role by the realization of the user story in the software system.

In the following format, a user story statement is structured using role, goal, and value, with optionally interspersed text. It is based on an abstraction of a number of user stories given in [Cohn, 2004].

```
[text] [role] [text] [goal] [text] [value] [text]
```

Rationale. The format "As X I want Y so that Z" was initially suggested by Connextra at XP 2001 Conference[24]. It has since then evolved and been adapted by others. In particular, the format "I as a (role) want (function) so that (business value)" has been suggested [Cohn, 2004], the format "As a (role) I want (something) so that (benefit)" has been suggested[25], and the format "As a <role>, I can <activity> so that <business value>" has been suggested and called "user voice" [Leffingwell, 2011]. The format "As a

[24] URL: http://agilecoach.typepad.com/photos/connextra_user_story_2001/connextrastorycard.html .

[stakeholder], I want [feature] so that [benefit]" has been suggested [Chelimsky, Astels, Dennis, Hellesoy, Helmkamp, North, 2010], although the term 'stakeholder' is not defined.

Literature Review.

USIU	Frequency	Resource(s)
Statement	4	[Chelimsky, Astels, Dennis, Hellesoy, Helmkamp, North, 2010] [Cohn, 2004] [Leffingwell, 2011] [Monochristou, Vlachopoulou, 2007]

Remark 5.15. The inclusion of user role, goal, and value to structure a user story statement is not exclusive to user stories. Indeed, an expression such as "The <stakeholder type> shall be able to <capability>" to state a software requirement has been suggested [Hull, Jackson, Dick, 2011, Section 4.8], and an expression such as "Subject Verb Object" to structure a use case description has been suggested [Phalp, Vincent, Cox, 2007].

Remark 5.16. A negative user story is a user story that suggests a negative use of the software system. An "abuser story" is a kind of negative user story that is of interest in security requirements engineering [Peeters, 2005]. A negative user story has a negative user story statement. For example, the following is a negative user story statement: A spammer wants to post a message so as to disseminate unsolicited advertisement. The

[25] URL: http://www.agilemodeling.com/artifacts/userStory.htm .

meaning of acceptance criteria for negative user story is different from that of a positive user story as all tests of a negative user story must fail.

USIU: Status

A user story description is related to time.

A user story can have different statuses (or states) depending on whether it is being authored, reviewed, or developed. For example, the authoring status of a user story may be ‘Incomplete’ or ‘Complete’.

Rationale. A user story, following a review, may be accepted or rejected [Lewitz, 2004]. If accepted, a user story goes through a number of developmental stages. There are different schemes for reflecting the development status of a user story [Fancott, Kamthan, Shahmir, 2011; Hiranabe, 2007].

USIU: Theme

The user stories in a user story book can be related in a number of ways. For example, a subset of user stories could be related by a single topic at a higher level of abstraction. This motivates the following definition:

Definition 5.10 [Theme]. A collection of user stories related by a specific topic of interest.

In other words, a theme is a means for topical and textual organization of user stories.

It is evident that a single user story book can have multiple themes. For example, there can be multiple user stories related to 'navigation', 'search', or 'payment'.

It can be expected that all user stories related to a single theme will be completed in a single iteration.

Rationale. A non-trivial software system can have a large number of user stories, and a means of organizing them [Derby, Larsen, 2006, Section 6.8] can be useful for understanding and for allocation of work. The activity of "classify[ing]" user stories is included in the SMM Level 4, Key Process Area 4.5 [Patel, Ramachandran, 2009b].

Literature Review.

USIU	**Frequency**	**Resource(s)**
Theme	2	[Leffingwell, 2011] [Sy, 2007]

USIU: Value

A value provides the rationale or reason for the existence of a user story.

It could be noted that the value system of software engineers, customer, and users, may not be identical, or even overlapping.

In general, the value must be explicit. A user story statement can make the underlying value explicit. For example, in the following user story statement, the value is explicit:

A <u>student</u> can <u>search the system</u> for <u>employment opportunities</u>.

However, at times, the value can be implicit in the goal (that is, subsumed by the way the goal is expressed). For example, in the following user story statement, the role and goal are explicit; however, the value is implicit[26]:

A <u>bank customer</u> can <u>withdraw money from a bank account</u>.

Rationale. The need for being Valuable is among the characteristics of a user story [Wake, 2002]. A user story exists because it is intended to provide value to some user (role). A user story description must make that value explicit.

[26] An obvious, and therefore implicit, value in withdrawing money from an account is the increase in the amount of money in possession of the customer.

Literature Review.

USIU	Frequency	Resource(s)
Value	5	[Chelimsky, Astels, Dennis, Hellesoy, Helmkamp, North, 2010] [Cohn, 2004] [Leffingwell, 2011] [Monochristou, Vlachopoulou, 2007] [Wake, 2002]

Remark 5.17. In a positive user story, value to a user is also value to the customer, that is, the values are aligned in the same direction. Let US be a negative user story that leads to an issue I. Then the value of US can be assessed in terms of the cost to the customer of not addressing I.

5.6. Limitations of USIM

USIM provides a collection, not the collection, of user story information units. This collection of user story information units is based on individual decisions made within the confines of a thesis.

The relevant relationships between user story information units, and properties of those relationships, are not identified or described by USIM. (For example, several pairs of user story information units are mutually-related by a composition relationship. It is possible to express such relationships in UML or OWL.)

5.7. Summary

A user story description, if structured, typically consists of a number of user story information units. A user story information unit may be at the level of a user story book, or at the level of a user story.

Chapter 6 A User Story Representation Model

[The] most important lesson to learn from SGML is not the syntax but the concept of generic markup [for] describing things in terms of their semantics rather than their appearance [...].
— James Clark

This chapter presents a conceptual model for representing a user story description, namely USRM. The purpose of USRM is to help create an understanding of a user story representation from the perspective of markup, in general, and technologies for markup, in particular.

6.1. Goals and Strategy

The goals of USRM are to lend support to USDM and to be sufficiently detailed so as to be useful as an input to USML. USRM aims to be independent of any specific device or tool. The discussion of any technology in USRM is avoided until necessary, and is done so to make the arguments concrete.

USRM is based on the following strategy:

- **Select Markup Type.** There are different types of markup. They need to be identified and compared. This provides necessary rationale for selecting the one that is most appropriate for USML and, by reference, for representing a user story.

- **Make Technological Commitment.** There are different technologies that support the notion of descriptive markup. They need to be identified and compared. This provides necessary rationale for selecting the one that is most appropriate for USML and, by reference, for representing a user story.

6.2. User Story Representation

For the sake of this thesis, a resource[27] is “anything that is identifiable by a naming and addressing scheme”, such as a Uniform Resource Identifier (URI), and a representation is “data that encodes information about resource” [Jacobs, Walsh, 2004].

Definition 6.1 [User Story Representation]. A representation of a user story description in some artificial (usually, computer) language.

6.2. From Scriptio Continua to Descriptive Markup

There are a number of definitions of a document, including the following:

Definition 6.2 [Document] [ISO, 1986]. A collection of information that is processed as a unit.

[27] It could be noted that a resource could be animate or inanimate. For example, a person could be a resource, and could be represented in RDF.

The term markup was coined in the late 1960s, and has its origins in scholarly text processing [Coombs, Renear, DeRose, 1987].

Definition 6.3 [Markup] [ISO, 1986]. Text that is added to the data of a document in order to convey information about it.

In this thesis, the term 'markup' (a noun) instead of 'mark up' (a verb) is used.

There are different kinds of markup, each serving a certain purpose. A classification of markup is presented in [Goldfarb, 1981] and has been extended in [Coombs, Renear, DeRose, 1987]. The main categories are: punctuational markup, presentational markup, procedural markup, and descriptive markup. Figure 5.1 illustrates a taxonomy of markup that includes other categories.

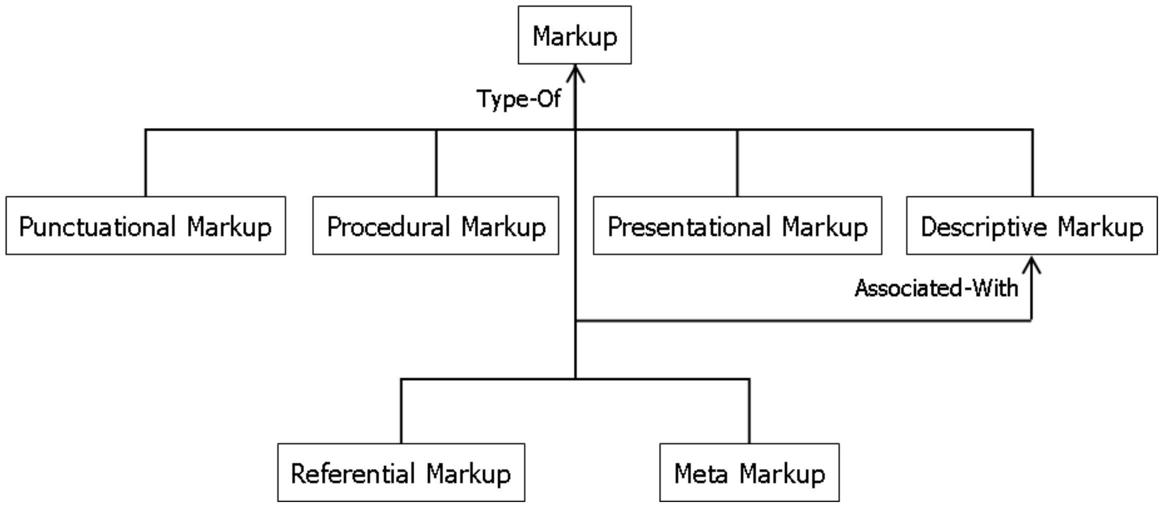


Figure 5.1. A taxonomy of markup.

- **Punctuational Markup.** The purpose of punctuational markup is to create delimiters (boundaries). For example, spaces between words indicate word boundaries, commas indicate phrase boundaries, or periods indicate sentence boundaries are all manifestations of punctuational markup. The issues with punctuational markup include the fact that there is ambiguity stemming from variations in usage and appearance.

- **Presentational Markup.** The purpose of presentational markup is to improve communicability in presentation in some modality such as visual or aural. For example, bolding a keyword, indenting a quote, centering the title of a book chapter, numbering a page, or emphasizing a word in spoken text by increasing its pitch relative to the others in a sentence are all manifestations of presentational markup. The issues with presentational markup include the fact that the presentation decisions are often ad-hoc, and the markup process is repetitious.

- **Procedural Markup.** The purpose of procedural markup is to instruct or provide commands to a document processing system for formatting. For example, `{\rtf1\ansi\{\b Hello World!}}` in Rich Text Format (RTF) 1.0 is a directive to the processor to render "Hello World!" in bold typeface, and `$e^x$` in TEX is a directive to the processor to render `x` slightly elevated next to the right of `e`. The issues with procedural markup include the fact that it is usually device-dependent.

- **Descriptive Markup.** The purpose of descriptive markup is to indicate what a text element is by declaring that a certain portion of text stream is a member of a particular class. For example, `<math><exp/><ci>x</ci></math>` denotes the exponential function of the variable `x` in the Mathematical Markup Language (MathML). It is the intention of descriptive markup to be general in its target applications (that may include formatting, say, by mapping onto procedural markup). For example, a tag in the procedural markup states "do `x` here", while a tag in the descriptive markup describes "this is an `x`".

6.2.1. Significance of Descriptive Markup

The descriptive markup is based on the following postulate [Goldfarb, 1981]: "Markup should describe a document's structure and other attributes, rather than specify processing to be performed on it, as descriptive markup need be done only once and will suffice for all future processing".

The descriptive markup is based on a model of text known as the Ordered Hierarchy of Content Objects (OHCO) [DeRose, Durand, Mylonas, Renear, 1997]. OHCO lends a hierarchical structure to documents, and makes descriptive markup the most general in scope compared to other types of markup.

6.2.2. User Stories and Descriptive Markup

The premise of this thesis is that the normative source of a user story description must be based on descriptive markup. Furthermore, to preserve the original spirit of descriptive markup, the source should not be intermixed with other kinds of markup.

Therefore, this thesis adopts the use of descriptive markup for representation of user stories and the use of presentation markup for presentation of user stories. The use of these markup types also has implications towards publishing user stories in distributed electronic mediums such as the Web.

The difference between a user story statement in punctuational markup and descriptive markup is illustrated in Figure 5.2. A '|' indicates the presence of a start-tag or end-tag, as appropriate. The 'red portion' (or first element node) corresponds to (user) role, the 'green portion' (or second element node) corresponds to goal, and the 'blue portion' (or third element node) corresponds to value.

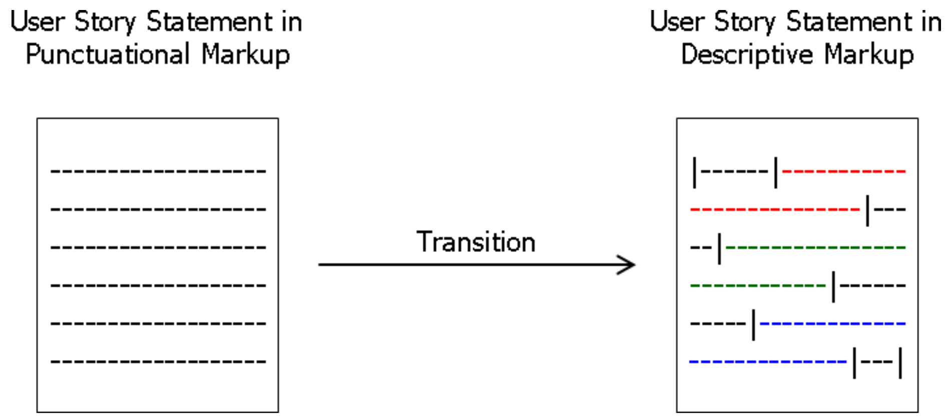

User Story Statement in Punctuational Markup
User Story Statement in Descriptive Markup
Transition

Figure 6.2. The transition of a user story statement from punctuational markup to descriptive markup.

6.3. SGML and XML

The term markup language was coined in the early 1970s and has its origins in the Generalized Markup Language (GML) [Goldfarb, 1981].

Definition 6.4 [Markup Language]. A means for using markup to describe information, belonging to some domain of interest, in a document. To do that, a markup language consists of a collection of elements. These elements may have a set of attributes that describe the properties of those elements.

If used appropriately, the Standard Generalized Markup Language (SGML) [ISO, 1986], and its simplified successor, the Extensible Markup Language (XML) [Bray, Paoli, Sperberg-McQueen, Maler, Yergeau, 2008], are exemplars of descriptive markup. SGML and XML are meta-markup languages, that is, meta-languages for creating markup languages [Maler, El Andaloussi, 1996; Murata, Lee, Mani, Kawaguchi, 2005].

In SGML and XML, there is notion of an instance document.

Definition 6.5 [Instance Document]. A document that satisfies the constraints of a meta-markup language.

For example, an SGML instance document or an XML instance document satisfies the constraints of SGML and XML, respectively.

The interest in this thesis is in XML. A commitment to XML comes with its advantages and disadvantages.

6.3.1. The Advantages and Limitations of the Use of XML

The following are primary advantages of the use of XML:

- The use of XML has been advocated in a number of contexts, especially those related to the Web [Jacobs, Walsh, 2004].
- Using XML, it is possible to constrain the structure of a user story description and, to a certain extent, constrain the type of information in a user story description.
- XML is based on Unicode [The Unicode Consortium, 2007], which is necessary for the support of a variety of characters, including those from different natural languages and mathematical symbols.
- The specification of XML is open.
- XML has a broad support in the industry and in other contexts such as Open Source Software (OSS) development community. This has led to the development of a range

of commercial and non-commercial tools[28] for a variety of tasks including authoring, processing, and archiving XML instance documents.

- XML is designed for the Internet. For example, XML is not dependent on any specific network, device, operating system, or programming language. Furthermore, unlike SGML, XML instance documents can be disseminated on the Internet and processed by tools even if these instance documents do not refer to the schema that they correspond to.

The following are primary limitations of the use of XML:

- There is a cost to a commitment to any technology, and XML is no exception.

- The specification of XML provides little on its own[29], which is both its strength and its weakness. The responsibility for the aspects that are deemed necessary for the definition of (descriptive) markup languages, but are not provided by XML, has been relegated to other information technologies. These aspects include character encoding, language support, disambiguation of elements and attributes from multiple different markup languages in a single instance document, and so on. Therefore, a commitment to XML means, by necessity, reliance on other specifications, which increases the learning time and the slope of the learning curve. These ancillary

[28] URL: http://www.garshol.priv.no/download/xmltools/.
[29] This is a direct consequence of the transition from SGML to XML, a discussion of which is beyond the scope of this thesis.

technologies must also work collectively which, in turn, increases the processing load.

- The specification of XML does not provide directions for use of markup. Therefore, a user of XML may need to rely on other sources for an effective use of markup [Harold, 2003; Megginson, 2005]. This increases the learning time and the slope of the learning curve.

- For some, authoring XML instance documents manually may be tedious and error-prone. In such cases, tool support may be necessary.

- The introduction of descriptive markup in text increases the number of characters, and therefore the file size. In other words, the same text expressed in XML will always be larger than the text without the markup in XML.

6.3.2. Descriptive Markup, XML, and USML

This thesis advocates the use of descriptive markup as the basis for USML due to the number of benefits that it offers. The focus on descriptive markup is critical as neither SGML nor XML impose any stylistic constraints on markup. Therefore, a basis of a markup language on SGML or XML alone is not sufficient for its instance documents to be descriptive. For example, HyperText Markup Language (HTML) is based on SGML but is not descriptive, and Wireless Markup Language (WML) is based on XML but is also not descriptive. Indeed, both HTML and WML mix different kinds of markup.

Furthermore, this thesis adopts XML over SGML as the meta-markup language for USML. (This decision is a manifestation of the use of the USE XML[30] pattern.)

Figure 6.3 illustrates a user story representation model. The XML ‘family’ consists of a body of specifications, each addressing a particular aspect, from a number of different standards organizations. In this thesis, these specifications have been used on a per-need basis. A detailed coverage of these specifications is beyond the scope of this thesis.

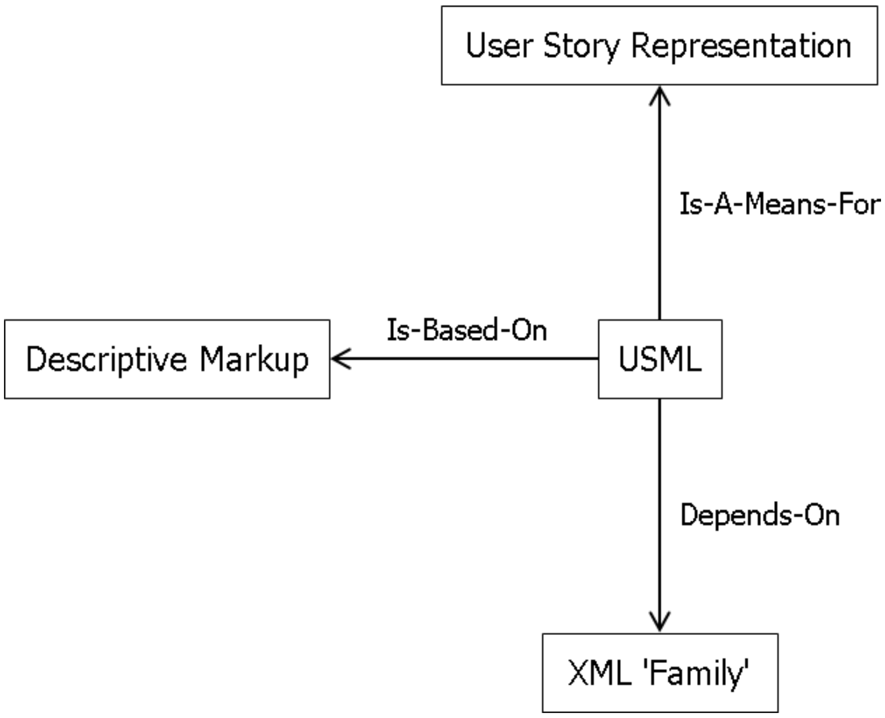


Figure 6.3. The relationships between USML, descriptive markup, XML ‘family’, and user story representation.

A USML instance document is an XML instance document, and conforms to USML. A user story representation and USML instance document are related, as illustrated in Figure 6.4. However, the two are not equivalent, as it is possible to have an instance document of a markup language other than USML to represent a user story description.

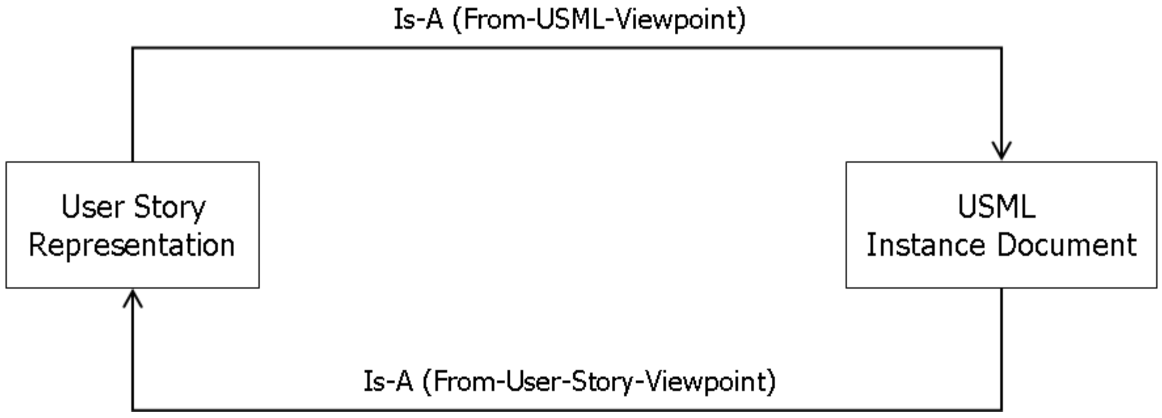

Is-A (From-USML-Viewpoint)
User Story Representation
USML Instance Document
Is-A (From-User-Story-Viewpoint)

Figure 5.4. The relationships between user story representation and USML instance document.

Definition 6.6 [Schema Document]. A document that provides constrains on the physical and logical structure of a class of instance documents.

An XML schema language is a language for authoring an XML schema document. There are a number of XML schema languages including XML Document Type Definition (XML DTD), XML Schema Definition Language (XSDL), REgular LAnguage for XML Next Generation (RELAX NG), and Schematron.

[30] URL: http://www.xmlpatterns.com/ .

An XML schema document is expressed in some XML schema language. The XML instance documents that an XML schema document constrains may correspond to some markup language.

The purpose of a USML schema document, as illustrated in Figure 6.5, is to provide constraints on the physical and logical structure of a USML instance document, beyond those that are provided by XML.

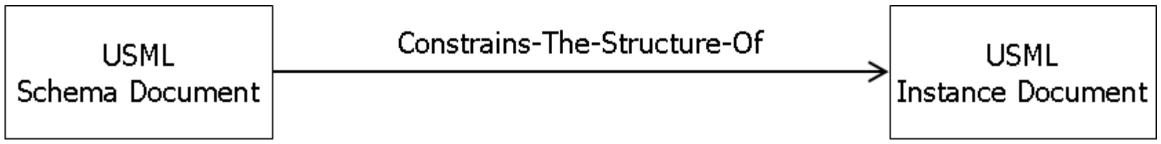


Figure 6.5. The relationship between USML schema document and USML instance document.

Definition 6.7 [Validation]. An activity that involves checking an XML instance document against an XML schema document.

An XML instance document that succeeds validation is called valid.

In SGML [Maler, El Andaloussi, 1996, Chapter 11], as well as in XML [Megginson, 2005, Section 4.1], validation is considered a ‘good’ practice.

6.4. Limitations of USRM

An anticipated consequence of isolating and separating concerns is the overhead of managing each concern that has been separated. The overhead could be perceived as significant as the number of concerns increase.

6.5. Summary

There are a number of means for representing a user story description. USML, as discussed in detail in Chapter 6, is one such means. USML uses descriptive markup, in general, and XML, in particular, due to the number of benefits that they offer for both human and machine consumption. A USML schema document places constraints on the physical and logical structure of a USML instance document, in addition to those that are provided by XML.

Chapter 7 A Characterization of User Story Markup Language

Language is the source of misunderstandings.
— Antoine de Saint-Exupéry

This chapter discusses the nature of USML, and specifics pertaining to the development of USML. In doing so, it lists the goals of USML, and highlights the major high- and low-level decisions underlying schema engineering, especially those related to the identification of elements and attributes of USML. These decisions are, as appropriate, informed by conventions, guidelines, patterns, principles, recommended practices, and standards related to markup language engineering.

7.1. USML as a Language

USML is a descriptive markup language that derives its syntax from a number of specifications.

The specifications layered on top of XML have been categorized into the following [Megginson, 2005, Section 1.4]: Core Specifications, Embeddable Specifications, Utility Specifications, and Application Specifications. Therefore, from an XML viewpoint, the specification of USML is an Application Specification. USML, by virtue of its focus on user stories, is a Domain-Specific Language (DSL) in the sense described in [Mernik, Heering, Sloane, 2005].

7.2. Goals of USML

USML is based on a number of process- and product-specific goals related to accessibility, authorability, efficiency, expressivity, extensibility, findability, flexibility, portability, readability, and reusability. These goals and their realization are discussed in the following sections.

It could be noted that the aforementioned goals are not mutually independent. In fact, from the perspective of impact, there are three kinds of relationships between goals: negative, positive, and neutral. For example, as seen later, steps taken towards readability can have a negative impact on (space) efficiency.

Remark 7.1. This thesis does not associate a numerical 'weight' with a goal. However, in case of a conflict, that is, a positive impact on goal one leads to a negative impact on another, the goals related to consumption of USML-related artifacts, in general, take precedence over goals related to production of those artifacts. In particular, the order of precedence is the following: accessibility, readability, portability, expressivity, flexibility, authorability, findability, efficiency, reusability, and extensibility.

7.2.1. Accessibility

USML aims to be accessible (in general, inclusive) so as to support broadest range of user story consumers. The following steps are taken in the direction of supporting accessibility:

- In the relevant schema documents of USML, there is a separation of representation of information from presentation of that information, there is an element for the title of a USML instance document, there is an attribute for language, and there is an attribute for expansion of an abbreviation.
- In the relevant schema documents of USML, the elements that have similar content have similar content models. This decision is inspired by [Bonneau, Kohl, Tennison, Duckett, Williams, 2003, Section 3.2.5; Megginson, 1998, Section 3.2.2], and is a manifestation of the use of the PARALLEL DESIGN[31] pattern.
- In the relevant style sheets of USML, there is use of relative units, the hyperlinks have associated purpose, the default visual properties of hyperlinks are preserved, the colors are specified by their respective hexadecimal values, the color contrast ratio is at least 7:1, and color is not used exclusively as visual means of conveying information.

It is known [Caldwell, Chisholm, Reid, Vanderheiden, 2008; ISO, 1998b; ISO, 2008] that the aforementioned steps contribute to accessibility. For example, a separation of representation and presentation allows the association of user style sheets.

[31] URL: http://www.xmlpatterns.com/ .

7.2.2. Authorability

USML aims to be authorable so as to support user story authors. The following steps are taken in the direction of supporting authorability:

- USML does not require any special computing environment, such as any particular operating system or editor, for authoring USML instance documents.
- The mandatory elements and attributes of USML are kept to a necessary minimum, and a number of elements and attributes of USML are optional.
- There is overloading of names of elements and attributes of USML. This keeps the names of syntactical constructs from growing. However, as pointed out in [Bonneau, Kohl, Tennison, Duckett, Williams, 2003, Section 3.2.4], overloading element names can lead to degraded performance, especially when using a stream-based XML processor for parsing large documents.
- The names of elements and attributes of USML are kept relatively short wherever possible.

7.2.3. Efficiency

USML aims to be efficient, that is, aims to conserve space and time. The following steps are taken in the direction of supporting efficiency:

- There is extensive use of named patterns[32]. This helps minimize redundancy, however, it also increases the file size.
- It is known [Jelliffe, 1998, Page 1-126] that an XML processor, especially if it is stream-based, has to do more work if it comes across reference to an element that it has not processed earlier. It is suggested by the DECLARE BEFORE FIRST USE[33] pattern that the elements be ordered in such a manner that they are encountered before they are referred to. The listing of RELAX NG schema for USML (in Appendix A) is based on this pattern.

7.2.4. Expressivity

USML aims to be expressive so as to support different needs of user story authors and user story consumers. The following steps are taken in the direction of supporting expressivity:

- The expressive power of a XML schema language is determined by the set of element content models that can be expressed in that language. For its schema documents, USML uses RELAX NG and Schematron directly, and XSDL indirectly. These XML schema languages are known to be expressive [Murata, Lee, Mani, Kawaguchi, 2005].
- The degree to which an element's content model is organized into child elements is known as granularity [Bradley, 2002, Page 92]. The RELAX NG schema for USML is

[32] A named pattern is a notion specific to REgular LAnguage for XML Next Generation (RELAX NG), an XML schema language. A named pattern is a section of a schema document. For example, among other things, it is possible to define named patterns for an element or an attribute, or the information it contains. A named pattern can be referred to, by its name, by other named patterns.

granular to a level deemed necessary. The granularity of markup plays a crucial role in making a user story representation expressive. Indeed, granularity is directly proportional to the number of user story information units. However, increase in granularity, inevitably, also leads to increase in file size, and can be an impediment to authoring.

7.2.5. Extensibility

USML aims to be extensible so as to support different needs of user story authors. The following steps are taken in the direction of supporting extensibility:

- There is an attribute for versioning USML instance documents.
- There is a mechanism for extending USML instance documents using fragments of other markup languages (covered in Sections 7.7.1 and 9.3).
- There are mechanisms for extending the RELAX NG schema for USML (covered in Section 10.3).

7.2.6. Findability

USML aims to be findable so as to support user story consumers. The following steps are taken in the direction of supporting findability:

[33] URL: http://www.xmlpatterns.com/ .

- The information in elements and attributes of USML is in text.
- There is support for meta-information in USML. It is known that meta-information can be used for indexing.
- The names of elements and attributes of USML are based on natural naming [Keller, 1990].
- The USML instance documents can be extended to include metadata from known metadata schemes.

The findability of schema documents and instance documents of USML, and the information therein, is especially relevant in distributed computing environments such as the Web [Morville, 2005].

7.2.7. Flexibility

USML aims to be flexible so as to support user story authors and user story consumers. The following steps are taken in the direction of supporting flexibility:

- A number of elements and attributes of USML are optional. For example, all child elements (except one, by necessity) of the `usml` element are optional.
- For a number of elements and attributes of USML, there is choice of the kind of information that can be included in each of them. For example, there are choices for the type of estimation approach, priority approach, and test style.

This flexibility enables the possibility of multiple user story forms. Indeed, as discussed in Chapter 9, it possible to create user story forms that range from rudimentary to elaborate.

7.2.8. Portability

USML aims to be portable so as to support appropriate user story stakeholders. The following steps are taken in the direction of supporting portability:

- The schema documents and instance documents of USML use technologies that are either standards or are considered as standards.
- The schema documents and instance documents of USML do not depend on any specific computing environment, authoring tool, or entity libraries.

7.2.9. Readability

USML aims to be readable so as to support appropriate user story stakeholders. The following steps are taken in the direction of supporting readability:

- The schema documents and instance documents of USML have white space, specifically, indentation and blank lines, at places that are deemed appropriate.
- The schema documents and instance documents of USML have comments at places that are deemed appropriate. (This is a manifestation of the use of the CODE-COMMENT PROXIMITY pattern [Rüping, 2003]. The inclusion of comments is

considered a ‘good’ practice in the development of SGML DTD [Maler, El Andaloussi, 1996, Section 9.1.1] and XML DTD [Harold, 2003, Item 5].) There are a number of ways of annotating a schema document [Vlist, 2004, Chapter 13], each with their own advantages and disadvantages. The schema documents and instance documents of USML use XML comments. However, an elaborate means for annotating the RELAX NG Schema for USML is illustrated in Appendix A.

- The names of elements and attributes of the RELAX NG schema for USML are not qualified, that is, do not have namespace prefixes.

It could be noted that, in the RELAX NG schema for USML, an extensive use of named patterns, along with references to them, can interrupt the reading order.

7.2.10. Reusability

USML aims to be reusable so as to support user story authors and other, non-user story, stakeholders to whom USML may be of interest. The following steps are taken in the direction of supporting internal and external reusability:

- The RELAX NG schema for USML is modular. It makes extensive use of named patterns, and internal and external references to those named patterns. For example, the definition of each element and attribute of USML is a named pattern.

- The RELAX NG schema for USML is divided into multiple files. (This is a manifestation of the use of the MULTIPART FILES[34] pattern.) The external files standalone as 'child grammars'.

7.3. Scope of USML

It is not the purpose of USML to provide all desirable semantic constraints on a user story description. The following examples illustrate the ensuing impact:

- USML requires that a priority be associated with each user story. However, a verification of whether that priority is correct is beyond the scope of USML.
- USML requires that an estimate be associated with each user story. However, a verification of whether that estimate is acceptable is beyond the scope of USML.

In other words, USML can not always substitute for the desirable skills of a user story author.

7.4. USML and XML Schema Languages

An XML schema language can be either grammar-based or rule-based. There are a number of grammar-based and rule-based XML schema languages, both standard and otherwise.

[34] URL: http://www.xmlpatterns.com/ .

For example, XML DTD, XSDL, and RELAX NG are grammar-based XML schema languages, and Schematron is a rule-based XML schema language.

There are advantages and disadvantages of each XML schema language. For the sake of argument, consider the following. Unlike RELAX NG, it is not possible to have mixed content[35] and, at the same time, enforce order and number of child elements in an XML DTD, although these features are important for adequately representing a user story description. Unlike XML DTD, there is no native support for entities in RELAX NG.

7.4.1. A Multiple XML Schema Language Approach

This thesis is based on the view that the XML schema languages can complement each other. For example, for support of data types, RELAX NG relies on XSDL. Therefore, USML uses multiple schema languages, each for a different purpose, taking advantage of the strengths of each.

The grammar-based USML schema document is expressed in RELAX NG. The rule-based USML schema document is expressed in Schematron. Figure 6.1 illustrates the role played in USML by different XML schema languages.

[35] An element has mixed content when elements of that type may contain character data, optionally interspersed with child elements.

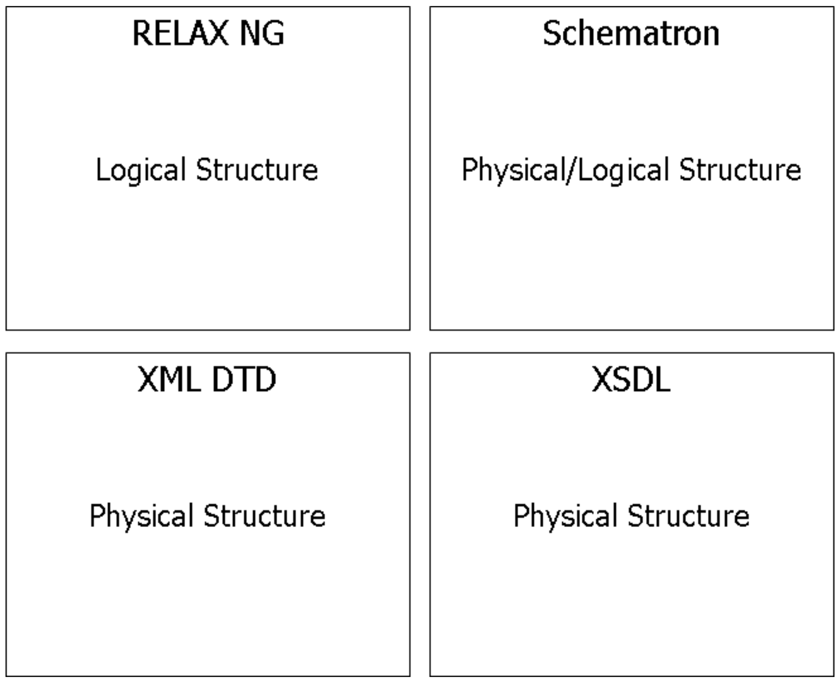


Figure 7.1. The role of XML schema languages in USML.

Remark 7.2. The schema languages selected and used in USML are part of the ISO/IEC 19757 Standard initiative entitled Document Schema Definition Languages (DSDL)[36].

Remark 7.3. There are comparisons of XML schema languages available [Harold, 2003, Item 24; Murata, Lee, Mani, Kawaguchi, 2005]. A comparison of XML schema languages is beyond the scope of this thesis.

7.4.2. Scope of XML Schema Languages

There are certain constraints that are not checked by any current XML schema language. For example, none of the aforementioned XML schema languages can check if the

[36] URL: http://dsdl.org/ .

resource pointed to at the end of a URI exists, or has requisite permissions for being available.

7.4.3. The RELAX NG Schema for USML

RELAX NG is an XML schema language defined by the ISO/IEC 19757-2 Standard [ISO/IEC, 2008]. The RELAX NG schema for USML is given in Appendix A.

For the sake of this thesis, macro-architecture of a schema is concerned with high-level aspects such as those related to organization, while micro-architecture of a schema is concerned with low-level aspects such as those related to individual elements and attributes.

7.4.3.1. The Macro-Architecture of the RELAX NG Schema for USML

The macro-architecture of the RELAX NG schema for USML is a manifestation of the use of the VENETIAN BLIND pattern [Bonneau, Kohl, Tennison, Duckett, Williams, 2003, Section 3.6.4]. The VENETIAN BLIND pattern is selected among the possible patterns as it most optimally contributes to the goal of extensibility of USML. The RELAX NG schema for USML, as a result, has a relatively flat hierarchy.

The macro-architecture of the RELAX NG schema for USML is based on the notion of modularization. In particular, the RELAX NG schema for USML consists of a set of modules, as illustrated in Figure 7.2.

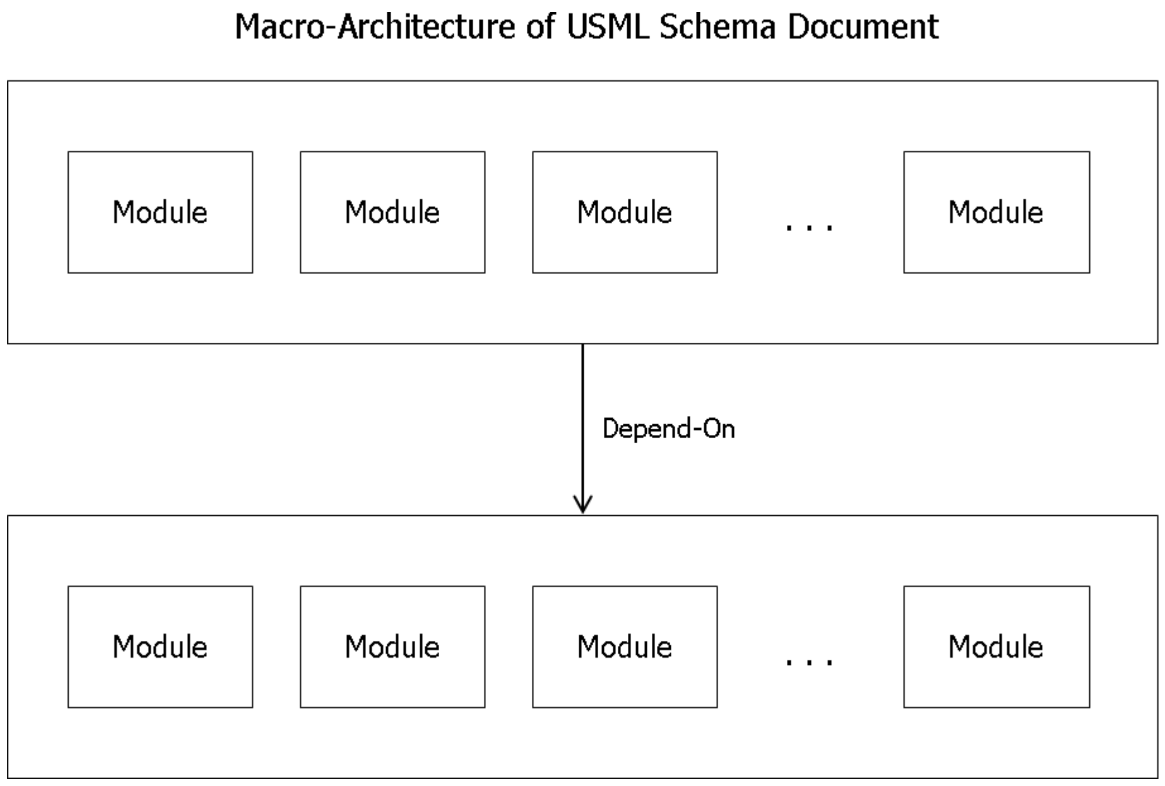


Figure 7.2. The macro-architecture of the RELAX NG schema for USML as a collection of modules.

A module can either be independent or dependent. A dependent module uses (and therefore depends upon) an independent, or another dependent, module.

The modules are individually and uniquely named. In particular, a module in RELAX NG is realized using a named pattern.

A module resides in a file on some file system. A file can have one or more modules. A module can be referenced by its name and, if necessary, its file path.

7.4.3.2. The Micro-Architecture of the RELAX NG Schema for USML

A module is related to an element or an attribute (or both), as illustrated in Figure 7.3.

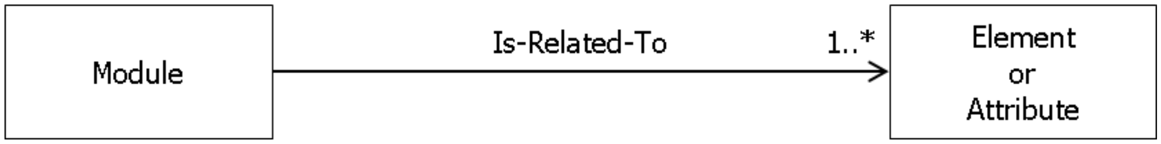


Figure 7.3. The relationship between a module of the grammar-based USML schema document, and USML element or USML attribute.

The arrangement of elements to appear in a USML instance document is a manifestation of the use of the SEPARATE METADATA AND DATA[37] and METADATA FIRST[38] patterns.

The elements and attributes of USML are treated further in Section 7.6.

7.4.4. The Schematron Schema for USML

Schematron is an XML schema language defined by the ISO/IEC 19757-3 Standard [ISO/IEC, 2006a]. The Schematron schema for USML is given in Appendix B.

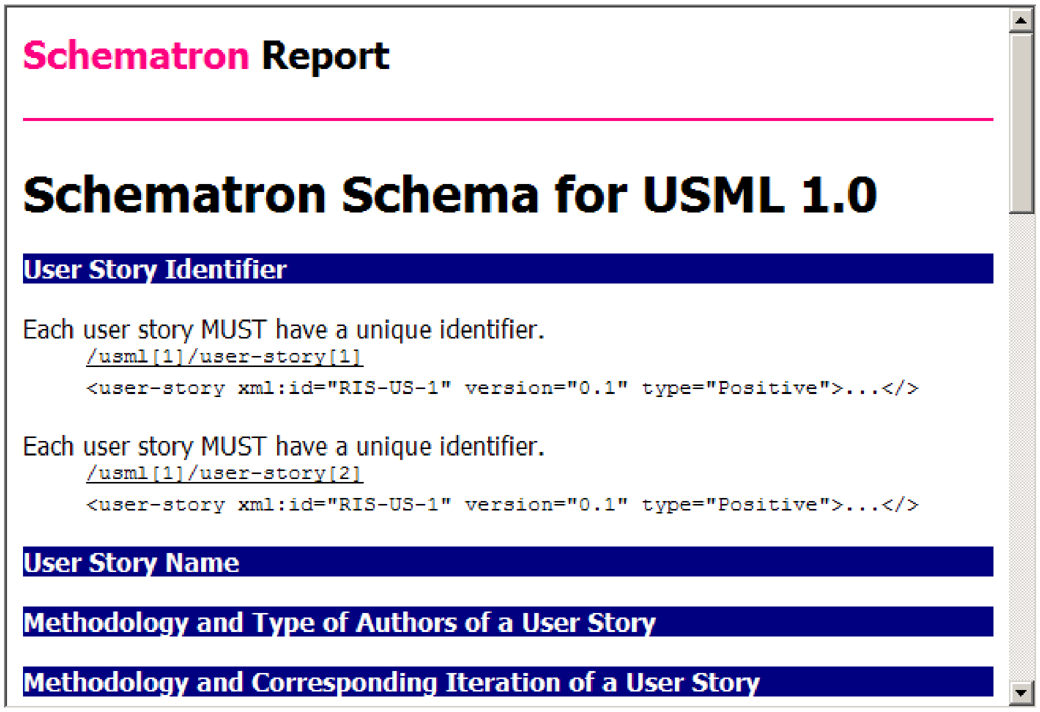
The Schematron schema for USML is complementary to the RELAX NG schema for USML. The purpose of the Schematron schema for USML is to express constraints that, for certain reasons, have not been, or can not be, expressed by the RELAX NG schema

[37] URL: http://www.xmlpatterns.com/ .
[38] URL: http://www.xmlpatterns.com/ .

for USML. These constraints include certain co-occurrence constraints between elements and attributes.

The constraints in the Schematron schema for USML are expressed as rules and tests using the XML Path Language (XPath) [Berglund, Boag, Chamberlin, Fernández, Kay, Robie, Siméon, 2010].

Figure 7.4 illustrates a snapshot of a report of a (candidate) USML instance document that has passed all but failed one of the tests of the Schematron schema for USML.



Schematron Report

Schematron Schema for USML 1.0

User Story Identifier

Each user story MUST have a unique identifier.
/usml[1]/user-story[1]
```
<user-story xml:id="RIS-US-1" version="0.1" type="Positive">...</>
```

Each user story MUST have a unique identifier.
/usml[1]/user-story[2]
```
<user-story xml:id="RIS-US-1" version="0.1" type="Positive">...</>
```

User Story Name

Methodology and Type of Authors of a User Story

Methodology and Corresponding Iteration of a User Story

Figure 7.4. The result of running the Topologi Schematron Validator using, as input, a (candidate) USML instance document and the Schematron schema for USML.

Remark 7.4. It could be noted that a violation by USML instance document of a constraint in the Schematron schema for USML can sometimes be a symptom of a problem, rather than a problem. (This is illustrated by Example 10.11.) Therefore, the results of a diagnostic require human intervention and judgment prior to any subsequent action.

Remark 7.5. The constraints in the Schematron schema for USML are relevant, and therefore necessary; however, they may not be sufficient. In particular, this thesis does not make any claim that all possible co-occurrence constraints have been identified and expressed in the Schematron schema for USML.

7.4.5. USML and NVDL

The Namespace-based Validation Dispatching Language (NVDL) [ISO/IEC, 2006b] is an XML schema language for validating XML instance documents containing multiple namespaces. NVDL enables concurrent validation.

The following is the NVDL schema for USML. It can concurrently validate a USML instance document with respect to the RELAX NG schema for USML and the Schematron schema for USML.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!--

Filename: usml.nvdl
Author: Pankaj Kamthan
Date: July 15, 2011
Version: 1.0
Description: This is the NVDL schema for the User Story Markup Language
             (USML) 1.0.
License: Creative Commons Attribution-Noncommercial-No Derivative Works
         3.0 Unported

-->

<rules xmlns="http://purl.oclc.org/dsdl/nvdl/ns/structure/1.0">

  <namespace ns="http://users.encs.concordia.ca/~kamthan/usml/1.0">
    <allow/>
  </namespace>

  <namespace ns="">
    <validate schema="usml.rng"/>
    <validate schema="usml.sch"/>
  </namespace>


  <anyNamespace>
    <allow/>
  </anyNamespace>

</rules>
```

7.5. Nature of USML Instance Documents

There are two common kinds of SGML or XML instance documents: data-oriented and narration-oriented [Bonneau, Kohl, Tennison, Duckett, Williams, 2003, Section 2.1; Megginson, 2005]. (For example, a library book record is essentially data-oriented.) A USML instance document is essentially a hybrid, although leaning towards a narration-oriented document. The narration-oriented nature of a user story description leads to mixed content [Harold, 2003, Item 13].

A USML instance document derives knowledge from multiple domains, as illustrated in Figure 7.5.

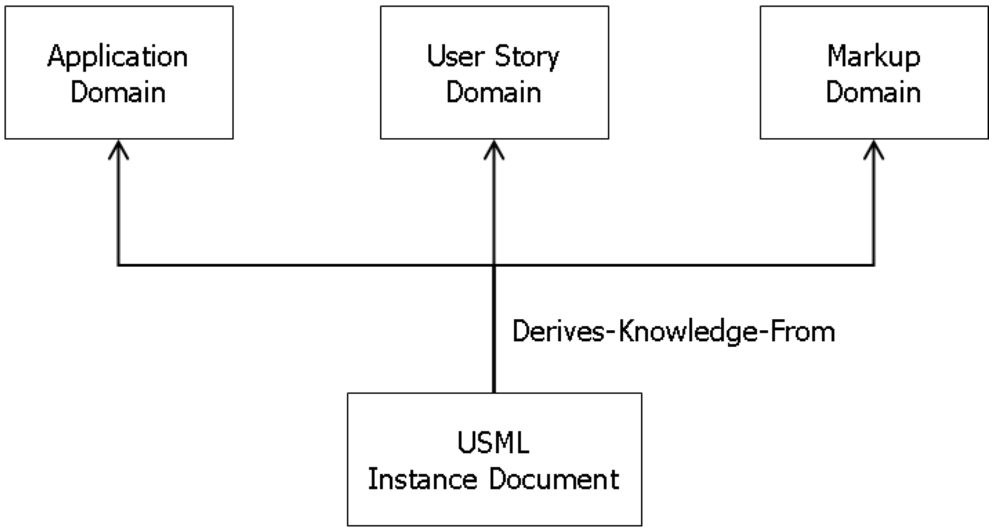


Figure 7.5. The sources of domain knowledge in a USML instance document.

For example, in a USML instance document, a character string such as ‘transfer money from one account to another’ belongs to the banking domain, a character string such as `role` belongs to the user story domain, and a character string such as ‘`/>`’ belongs to the markup domain.

7.6. Elements and Attributes of USML

Table 7.1 illustrates tag clouds, generated using Wordle[39], that provide a graphical organization of the names of elements and attributes of USML. (The assignment of color is random, and is only meant to distinguish names.) The details of elements and attributes of USML are given in Chapter 8.

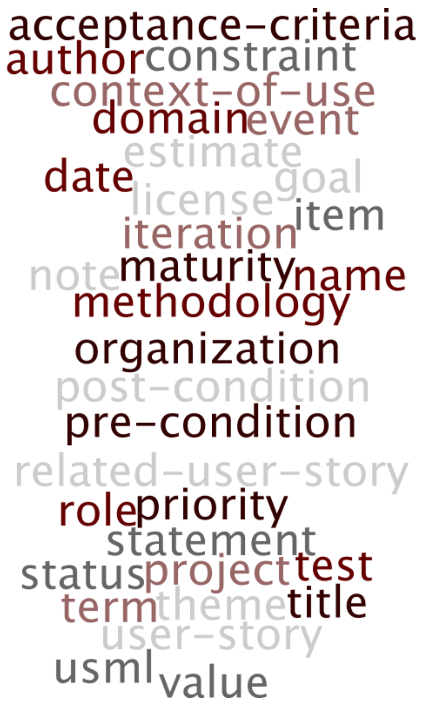
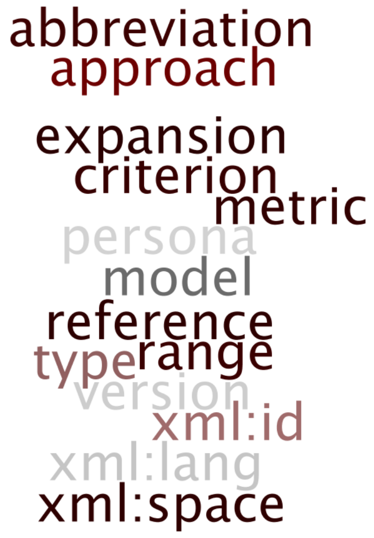
USML Elements	USML Attributes
	

Table 7.1. The tag clouds of elements and attributes of USML.

7.6.1. Characteristics of Elements and Attributes in USML

There are different kinds of elements and attributes in USML:

- There are elements and attributes in USML so that it can be recognized as a markup language based on XML.
- There are elements and attributes in USML due to its dependency on the conceptual models introduced in previous chapters.

[39] URL: http://wordle.net/ .

- There are elements and attributes in USML for the sake of granularity of a user story description.

7.6.2. Elements versus Attributes in USML

The decision whether certain information should be expressed as an element or an attribute has a long heritage in SGML [Jelliffe, 1998, Page 1-80] and, by reference, in XML [Bonneau, Kohl, Tennison, Duckett, Williams, 2003, Section 2.2; Bradley, 2002, Page 95; Harold, 2003, Item 12]. The debate is especially relevant to an SGML DTD or an XML DTD in which an attribute declaration is not 'first-class'.

There are known advantages and disadvantages of using an element or an attribute for expressing information. However, there are no 'universal' rules for deciding if certain information should be expressed in an element or in an attribute.

In deciding whether certain information should be expressed as an element or an attribute, the following applies to USML:

- **Criteria for Element in USML.** In USML, certain information is expressed as an element for one or more of the following reasons: the information is more useful to humans than to machines, order and occurrence are significant, it must have an attribute, and there is a substructure or there is potential for extension by a substructure (such as addition of a child element or an attribute).

- **Criteria for Attribute in USML.** In USML, certain information is expressed as an attribute for one or more of the following reasons: the information is more useful to machines than to humans, the information is a property of some element, and relative order and occurrence of the information are not significant.

7.6.3. Elements and Attributes of USML, User Story Information Units, and USML Instance Documents

A user story information unit is represented in USML as one or more elements or one or more attributes, as illustrated in Figure 7.6.

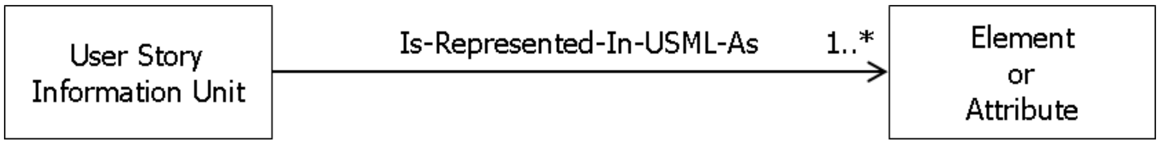


Figure 7.6. The relationship between a user story information unit and USML element or USML attribute.

For example, estimate, as discussed in Chapter 5, is a user story information unit. The placeholder of the numerical value of an estimate could be represented as an element, and the approach and the metric used for estimation could be represented as an attribute.

A USML instance document, as in other markup languages, consists of a number of elements, and these elements may have zero or more attributes. This is illustrated in Figure 7.7. (The precise constraints on an USML instance document are given in Section 7.8, and on USML elements and attributes are given in Chapter 8.)

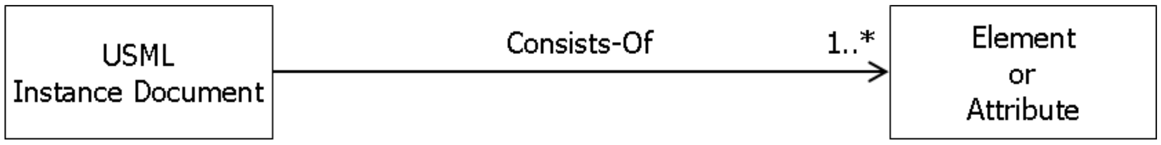


Figure 7.7. The relationship between a USML instance document, and USML element or USML attribute.

7.6.4. Names of USML Elements and Attributes

There are lexical, syntactic, semantic, and pragmatic (stylistic) considerations in naming. The names of elements and attributes in a markup language based on XML must conform to lexical and syntactical constraints imposed by XML. However, beyond that, the names of elements and attributes in a markup language are, to a large extent, dependent on the orthographical and typographical choices of the author of the markup language specification.

There are a number of naming conventions [Bonneau, Kohl, Tennison, Duckett, Williams, 2003, Section 3.2.4; Bradley, 2002, Page 93; Harold, 2003, Item 6; Megginson, 1998, Section 3.3.1], and each convention has its own advantages and disadvantages. These labeling conventions are intended to be relevant to humans, not to machines. In particular, the choice of labels is not significant to a computer program processing a USML instance document.

- **Lexical Considerations.** USML is based on XML, and XML in turn relies on the Unicode character set. The names of elements and attributes in USML follow the lexical guidelines set in [Bray, Paoli, Sperberg-McQueen, Maler, Yergeau, 2008, Appendix J Suggestions for XML Names].

- **Syntactic Considerations.** The names of elements and attributes in USML follow the syntactical guidelines set in [Bray, Paoli, Sperberg-McQueen, Maler, Yergeau, 2008, Appendix J Suggestions for XML Names]. The names of elements and attributes in USML are singular, unless otherwise expressed.

- **Semantic Considerations.** The USIM, as discussed in Chapter 5, is an input to the names of elements and attributes in USML. This decision reflects an extended use of the DOMAIN ELEMENT[40] pattern. However, there are certain elements and attributes in USML that are not in USIM. The labeling convention used for the names of elements and attributes in USML is based on the notion of natural naming. Thus, the names are complete words and, in general, abbreviations are not used for naming. However, this can lead to relatively longer names. This decision also reflects a use, however, only in part, of the SHORT UNDERSTANDABLE NAMES[41] pattern.

- **Pragmatic Considerations.** The names of elements and attributes in USML are expressed in lowercase. (XML is case-sensitive.) There are names of certain elements and attributes in USML that are composed of multiple words. In such cases, the words are separated by a hyphen (-).

[40] URL: http://www.xmlpatterns.com/ .
[41] URL: http://www.xmlpatterns.com/ .

7.7. Identifying USML

It is the intention of USML to be used in a number of contexts by humans and machines. For that, USML requires an ‘interface’.

7.7.1. Uniqueness of USML Names

A ‘compound’ instance document, by definition, has elements and/or attributes from different markup languages. It is possible for these elements and/or attributes from different markup languages to have the same name. If used in the same ‘compound’ instance document, this can potentially lead to collision of names of elements and/or attributes.

7.7.1.1. USML Namespace

A solution to the problem of the potential name collision is the assignment of a USML namespace as specified by Namespaces in XML [Bray, Hollander, Layman, Tobin, Thompson, 2009].

The recommended USML namespace name, identified by a URI, is:

```
http://users.encs.concordia.ca/~kamthan/usml/1.0
```

It could be noted that there is no file at the end of the above URI. The purpose of this URI is identification, not location.

The recommended USML namespace prefix is `us`.

7.7.2. USML Media Type

A computer program, such as a user agent, may be exposed to a USML instance document. In such an event, the program should be able to recognize and subsequently process the document in an appropriate fashion. To do that, the program needs to know the media type of the document.

There are guidelines for formulating a media type for markup languages based on XML [Murata, St. Laurent, Kohn, 2001; Harold, 2003, Item 45].

The media type for USML, as per [Murata, St. Laurent, Kohn, 2001, Section 3.1], is `text/xml`.

7.7.3. USML Filename Extension

It should be possible for a machine to associate, and subsequently recognize, that a certain file on a file system is a USML instance document.

The filename extension for a USML instance document, as per [Murata, St. Laurent, Kohn, 2001, Section 3.1], is `xml`.

However, if there are other files with the same extension then, for the purpose of disambiguation, an extension, such as `usml`, could be used.

7.8. Conformance of a User Story Description to USML

A user story description that aims to conform to USML must satisfy the following criteria:

- It must conform to XML. In particular, it must be well-formed, as defined in [Bray, Paoli, Sperberg-McQueen, Maler, Yergeau, 2008, Section 2.1].
- It must specify an appropriate character encoding[42] in the XML declaration.
- It must declare `usml` as its root element.
- It must validate against the RELAX NG schema for USML.
- It must validate against the Schematron schema for USML.
- If any namespaces other than of USML are used, it must conform to [Bray, Hollander, Layman, 1999]. In particular, if it includes any fragments of non-USML markup, it must specify the USML namespace URI (as specified in Section 7.7.1.1) in its root element and prefix all USML elements with '`us:`'.
- If it makes any references to external style sheets, it must conform to [Clark, 1999].

Remark 7.6. The inclusion of a "conformance model" is one of the "Good Practices" of a specification [Dubost, Rosenthal, Hazaël-Massieux, Henderson, 2005].

7.8.1. A Minimum Instance Document Conforming to USML

The following is a minimal instance document that conforms to USML:

```
<?xml version="1.0" encoding="UTF-8"?>
<usml version="1.0" xml:lang="en">

<user-story xml:id="..." version="...">
  <name>...</name>
  <date>...</date>
  <author type="...">...</author>
  <author type="...">...</author>

  <statement>
    ...<role>...</role>...<goal>...</goal>...
  </statement>

  <estimate approach="..."
            metric="...">
    ...
  </estimate>

  <priority approach="..."
            criterion="...">
    ...
  </priority>

  <acceptance-criteria>
    <test xml:id="...">...</test>
  </acceptance-criteria>
</user-story>

</usml>
```

It is evident that this USML instance document represents a single user story description.

[42] A character encoding is a set of mappings between the bytes representing numbers in the computer and characters in the coded character set such as Unicode.

Remark 7.7. The minimal conforming USML instance document listed previously could be used by a user story author as a starting point for an authoring template.

Remark 7.8. The inclusion of XML declaration in an XML instance document is not required. However, its inclusion is a recommended practice [Bray, Paoli, Sperberg-McQueen, Maler, Yergeau, 2008; Harold, 2003, Item 1].

Remark 7.9. The inclusion of character encoding in XML declaration is not required. However, its inclusion is a recommended practice [Bray, Paoli, Sperberg-McQueen, Maler, Yergeau, 2008; Harold, 2003, Item 1]. USML does not endorse any specific character encoding. A detailed discussion of character encodings is beyond the scope of this thesis.

Remark 7.10. It could be noted that not all character encodings include all Unicode characters. Furthermore, it may not be possible to directly enter certain characters in the character encoding using an input device, such as, a keyboard. In such a case, a character can be entered as an entity. An entity consists of an ampersand, followed by a name (or, equivalently, a decimal or a hexadecimal), followed by a semicolon. There are also certain characters that can not be entered as such, and need to be escaped. For such characters, XML has pre-defined entities. For example, the character '<' indicates start of markup, and can be escaped using the entity '`&lt;`'. It is possible, as shown in Example 9.2, to use entities in USML by referencing entity declarations in the internal DTD subset, that is, in the `DOCTYPE` declaration of each USML instance document that needs them.

7.9. Summary

USML aims to strike a balance among a number of competing desirables expressed as goals. In doing so, it is informed by established, experiential knowledge on markup language engineering. USML instance documents can be read by humans and processed by machines. The schema documents for USML provide means for validating (candidate) USML instance documents. There are a number of limitations of USML, understanding of which is necessary for adoption and use of USML.

Chapter 8 Elements and Attributes of USML

Numquam ponenda est pluralitas sine necessitate.
— William of Ockham

This chapter identifies, rationalizes, if necessary, and defines, in part, the elements and attributes of USML. The relationship, if any, of an element or an attribute to USIM is also mentioned.

The definitions of elements and attributes of USML expressed in RELAX NG are given in Appendix A. It could be noted that the order of definitions of elements and attributes in this chapter is different from that of Appendix A.

There are certain constraints that can not be expressed in RELAX NG schema for USML. These have been expressed in the Schematron schema for USML, which is given in Appendix B.

8.1. USML Elements

This section outlines salient characteristics common to USML elements, and subsequently provides the definitions of each of those elements.

8.1.1. Nature of USML Elements

An element can be either a parent or a child. An element can be either empty or non-empty. An empty element does not have any content. The elements of USML are non-empty, unless otherwise stated. A non-empty element of USML can be homogeneous or heterogeneous in the sense of the composition of its contents. This is made specific in the next section.

8.1.2. Data Types of USML Elements

The data types of USML elements determine the type of data allowed in element content. There are three possibilities:

1. **Singleton.** A Singleton data type indicates that the element content consists of only character data and does not have any child elements. (This is a manifestation of the use of the ATOM pattern [Vitali, Iorio, Gubellini, 2005].)
2. **Aggregate.** An Aggregate data type indicates that the element content consists of only child elements.
3. **Mixed.** A Mixed data type indicates that the element content is a mixture of character data and child elements, that is, character data and other elements can be interspersed in an element declared to be of a Mixed type.

For the types of character data in element content, a library of data types such as the one provided by XSDL can be used. The exact type of character data allowed is specified in the following sections that provide definitions of individual elements.

A None indicates that the element does not have any child elements.

8.1.3. Enumeration of USML Elements

The enumeration of an element is related to the number of times an element can occur in a USML instance document, that is, its multiplicity.

The occurrence of elements is classified as Required, Conditional, or Optional.

1. **Required.** An element that is labeled as Required must be present in every USML instance document.
2. **Conditional.** An element that is labeled as Conditional must, under the given conditions, be present in a USML instance document.
3. **Optional.** An element that is labeled as Optional may not be present in every USML instance document.

8.1.3.1. Cardinality Constraints on USML Elements

The concept of cardinality is related to occurrence, and indicates the number of times an element can occur, if at all. If an element is a child of another element, the cardinality indicates the number of times the child element can occur in its parent.

- A cardinality of 'N' indicates that the element occurs N times, where N = 1, 2, ...
- A cardinality of 'M..N' indicates that the element can occur M to N times, where M = 0, 1, 2, ..., N = 1, 2, ..., and N ≥ M.
- A cardinality of 'N..Unbounded' indicates that the element can occur N or more times, where N = 0, 1, 2, ...

For example, an element whose occurrence is Required has a cardinality of at least 1, and an element whose occurrence is either Optional or Conditional has a cardinality of at least 0. The root element is always Required with cardinality equal to 1.

8.1.4. Enumeration of USML Attributes

An attribute can occur only once in an element.

The enumeration of an attribute is related to the conditions under which an attribute can occur in an element.

The occurrence of attributes is classified as Required, Conditional, or Optional, and is indicated in the parenthesis next to their names in an element definition.

1. **Required.** An attribute that is labeled as Required must always be present in the element it is associated with.
2. **Conditional.** An attribute that is labeled as Conditional must, under the given conditions, be present in the element it is associated with.
3. **Optional.** An attribute that is labeled as Optional may not always be present.

A None indicates that the element does not have any attribute.

8.1.5. Definitions of USML Elements

The elements of USML, ordered alphabetically, are:

- `acceptance-criteria`
- `author`
- `constraint`
- `context-of-use`
- `date`
- `domain`
- `estimate`
- `event`

- `goal`
- `item`
- `iteration`
- `license`
- `maturity`
- `methodology`
- `name`
- `note`
- `organization`
- `pre-condition`
- `post-condition`
- `priority`
- `project`
- `related-user-story`
- `role`
- `statement`
- `status`
- `term`
- `test`
- `theme`
- `title`
- `user-story`

- `usml`
- `value`

In the following, the definitions of all elements of USML are given. The definitions are ordered alphabetically.

The `acceptance-criteria` Element

Description: The purpose of the `acceptance-criteria` element is to outline the conditions under which the user story will be accepted. These conditions are a set of one or more tests.

Associated USIU: Acceptance Criteria.

Data Type: Aggregate.

Occurrence: Required.

Cardinality Constraints: 1.

Attribute(s): None.

Parent Element(s): `user-story`.

Child Element(s): `test` (Required).

Example:

```
<acceptance-criteria>
  <test xml:id="T1">...</test>
  <!-- ... -->
</acceptance-criteria>
```

Remark 8.1. The `acceptance-criteria` element is a manifestation of the use of the CONTAINER ELEMENT[43] pattern.

The `author` Element

Description: The purpose of the `author` element is to include the full name of a user story author (as defined in Section 4.4.1). There must be at least two authors for a user story, one of which must be a customer or a user, and the other must be a software engineer, as per Section 4.4.3.

Associated USIU: None.

Data Type: Singleton.

Occurrence: Required.

Cardinality Constraints: 2..Unbounded.

Attribute(s): `reference` (Optional), `type` (Required).

Parent Element(s): `user-story`.

Child Element(s): None.

Example:

```
<author type="Software Engineer">
  Jane Doyle
</author>
```

Remark 8.2. The presence of exactly two user story authors allows the possibility of Promiscuous Pair Story Authoring [Šmite, Moe, Ågerfalk, 2010, Section 4.2.4.3].

[43] URL: http://www.xmlpatterns.com/ .

Remark 8.3. RELAX NG can not specify arbitrary numerical limits for cardinality constraints such as M ≥ 2, where M is a positive integer, or specific numerical limits for cardinality constraints such as M..N, where M and N are positive integers. In particular, it is not possible to express in the RELAX NG schema for USML that there must be at least two user story authors. Therefore, this constraint has been expressed in the Schematron schema for USML.

Remark 8.4. The `author` element provides minimal information on a user story author, and could be extended to include other details such as means of contact. The style of expressing names varies across cultures. The `author` element could also be extended to express a person's name in Firstname Lastname and in Lastname Firstname formats.

The `constraint` Element

Description: The purpose of the `constraint` element is to express the possible constraints on a user story statement. There can be multiple constraints of the same type or different types on a user story statement. For example, there can be a numerical constraint, or there can be multiple usability-related constraints. An accessibility- or usability-related constraint must be associated with a `context-of-use` element.

Associated USIU: Constraint.

Data Type: Mixed.

Occurrence: Optional.

Cardinality Constraints: 0..Unbounded.

Attribute(s): `type` (Optional), `xml:id` (Required).

Parent Element(s): `user-story`.

Child Element(s): `term` (Optional).

Example:

```
<constraint xml:id="U1" type="Usability">
  The <term>basic search<term> results containing more than 10 items
  must be split over multiple screen views.
</constraint>
```

Remark 8.5. There are standards for accessibility [ISO, 2008] and usability [ISO, 1998a; ISO/IEC, 2001]. A detailed treatment of accessibility or usability is beyond the scope of this thesis.

Remark 8.6. It is not possible to express in the RELAX NG schema for USML that an accessibility- or usability-related constraint must be associated with a `context-of-use` element. Therefore, this constraint has been expressed in the Schematron schema for USML.

The `context-of-use` Element

Description: The purpose of the `context-of-use` element is to provide the context in which the software system, that will implement the user story, will be used. The significance of context of use can be expressed as a list of items.

Associated USIU: Context of Use.

Data Type: Mixed.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): `reference` (Optional).

Parent Element(s): `user-story`.

Child Element(s): `item` (Optional).

Example:

```
<context-of-use>
  <item>Internet Access: Dial-Up Connection</item>
  <item>Operating System: Linux</item>
  <!-- ... -->
</context-of-use>
```

The `date` Element

Description: The purpose of the `date` element is to provide information on the date of creation of a user story. For the sake of consistency and uniguity (that is, single interpretation), an international standard, such as the ISO 8601 Standard [ISO, 2000] for dates, should be followed. However, there are certain restrictions. For example, `xsd:date` does not support any calendar system other than Gregorian, and does not support the date format `CCYYMMDD`. Therefore, the date element provides choices.

Associated USIU: None.

Data Type: Singleton.

Occurrence: Required.

Cardinality Constraints: 1.

Attribute(s): None.

Parent Element(s): `user-story`.

Child Element(s): None.

Example: The date January 10, 2010 can be represented as one of the following:

```
<date>January 10, 2010</date>
```

or

```
<date>2010-01-10</date>
```

However, the following is not valid:

```
<date>2010-10-01</date>
```

The `domain` Element

Description: The purpose of the `domain` element is to state the application (or problem) domain corresponding to the user story.

Associated USIU: Domain.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): None.

Parent Element(s): `usml`.

Child Element(s): None.

Example:

```
<domain>Banking</domain>
```

The `estimate` Element

Description: The purpose of the `estimate` element is to provide the result of estimating the user story.

Associated USIU: Estimate.

Data Type: Singleton.

Occurrence: Required.

Cardinality Constraints: 1.

Attribute(s): `approach` (Required), `metric` (Required), `range` (Conditional).

Parent Element(s): `user-story`.

Child Element(s): None.

Example:

```
<estimate approach="Planning Poker"
          metric="Story Points"
          range="Fibonacci Sequence">
  3
</estimate>
```

The `event` Element

Description: The purpose of the `event` element is to, in the context of testing, state the event that occurs upon the interaction of the user with the software system. The `event` element is meant to be used with the `test` element.

Associated USIU: None.

Data Type: Singleton.

Occurrence: Conditional.

Cardinality Constraints: 0..1.

Attribute(s): None.

Parent Element(s): `test`.

Child Element(s): None.

Example:

```
<event>
  The bank customer is reminded to respond within 5 seconds.
</event>
```

The `goal` Element

Description: The purpose of the `goal` element is to state the user's goal that will be realized upon completion of the user story.

Associated USIU: Goal.

Data Type: Singleton.

Occurrence: Required.

Cardinality Constraints: 1.

Attribute(s): None.

Parent Element(s): `statement`.

Child Element(s): `term` (Optional).

Example:

```
A <role>bank customer</role> can <goal>withdraw money from a bank
account</goal>.
```

The `item` Element

Description: The purpose of the `item` element is to organize some information in an itemized manner so that, if necessary, the information can subsequently be processed in different ways. In particular, itemized information could be presented in different ways.

Associated USIU: None.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..Unbounded.

Attribute(s): `reference` (Optional).

Parent Element(s): `context-of-use`, `note`.

Child Element(s): None.

Example:

```
<item>Internet Access: Dial-Up Connection</item>
```

The `iteration` Element

Description: The purpose of the `iteration` element is to indicate the iteration corresponding to a given user story. The notion of iteration is specific to evolutionary software processes. For example, iteration is inherent to agile methodologies [Williams, 2010]. However, it is possible to deploy user stories in non-evolutionary software processes. To accommodate such cases, the iteration element is optional.

Associated USIU: Iteration.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): None.

Parent Element(s): `user-story`.

Child Element(s): None.

Example:

```
<iteration>1<iteration>
```

The `license` Element

Description: The purpose of the `license` element is to act as the placeholder for declaring the license name for a user story book. It could be noted that the `license` element itself does not contain the license. The license corresponding to the license name could be pointed to using a URI.

Associated USIU: License.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): `reference` (Optional), `type` (Optional).

Parent Element(s): `usml`.

Child Element(s): None.

Example:

```
<license reference="http://www.gnu.org/licenses/gpl-3.0.html"
         type="Free">
  GNU General Public License 3.0
</license>
```

The `maturity` Element

Description: The purpose of the `maturity` element is to indicate the maturity level corresponding to the user story book, and therefore for a user story therein. It can be, based on Section 3.8, expected that a higher maturity level will have higher demands on the development and management of a user story book.

Associated USIU: Maturity.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): `maturity-model` (Required).

Parent Element(s): `usml`.

Child Element(s): None.

Example:

```
<maturity model="SMM">Level 4</maturity>
```

The `methodology` Element

Description: The purpose of the `methodology` element is to state the software development methodology, corresponding to the user story book in question, being pursued.

Associated USIU: Methodology.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): `reference` (Optional), `type` (Optional).

Parent Element(s): `usml`.

Child Element(s): None.

Example:

```
<methodology type="Agile">Scrum</methodology>
```

The `name` Element

Description: The purpose of the `name` element is to state a mnemonic corresponding to (1) the user story statement of a given user story, or (2) a test.

Associated USIU: Name.

Data Type: Singleton.

Occurrence: Required (in case of `related-user-story` and `user-story`), Conditional (in case of `test`).

Cardinality Constraints: 0..1.

Attribute(s): `reference` (Optional).

Parent Element(s): `related-user-story`, `test`, `user-story`.

Child Element(s): None.

Example: In case of a `related-user-story` or `user-story`:

```
<name>Display Course Schedule</name>
```

In case of a `test`:

```
<name>Check User Credentials</name>
```

The `note` Element

Description: The purpose of the `note` element is to include open, unresolved issues during negotiation, or otherwise record miscellaneous information.

Associated USIU: Note.

Data Type: Mixed.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): None.

Parent Element(s): `user-story`.

Child Element(s): `item` (Optional).

Example:

```
<note>The support for iPad is to be decided.</note>
```

The `organization` Element

Description: The purpose of the `organization` element is to act as the placeholder for naming the organization under the auspices of which the software project, in general, is being pursued, and the user story book, in particular, is being created.

Associated USIU: Organization.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): None.

Parent Element(s): `usml`.

Child Element(s): None.

Example:

```
<organization>Xanadu Communications</organization>
```

The `pre-condition` Element

Description: The purpose of the `pre-condition` element is to, in the context of testing, state the pre-condition (or the situation) under which the corresponding event, as given by the `event` element, occurs. The `pre-condition` element is meant to be used with the `test` element.

Associated USIU: None.

Data Type: Singleton.

Occurrence: Conditional.

Cardinality Constraints: 0..1.

Attribute(s): None.

Parent Element(s): `test`.

Child Element(s): None.

Example:

```
<pre-condition>
  The bank customer is presented with a number of choices for the
  types of transactions to select one from.
</pre-condition>
```

The `post-condition` Element

Description: The purpose of the `post-condition` element is to, in the context of testing, state the post-condition (or the consequence) of the corresponding event, as given by the `event` element. The `post-condition` element is meant to be used with the `test` element.

Associated USIU: None.

Data Type: Singleton.

Occurrence: Conditional.

Cardinality Constraints: 0..1.

Attribute(s): None.

Parent Element(s): `test`.

Child Element(s): None.

Example:

```
<post-condition>
  The bank customer's card is retained.
</post-condition>
```

The `priority` Element

Description: The purpose of the `priority` element is to include the priority of the user story.

Associated USIU: Priority.

Data Type: Singleton.

Occurrence: Required.

Cardinality Constraints: 1.

Attribute(s): `approach` (Required), `type` (Optional), `criterion` (Required).

Parent Element(s): `user-story`.

Child Element(s): None.

Example:

```
<priority approach="MoSCoW"
          type="Absolute"
          criterion="Risk Value">
  MUST
</priority>
```

The `project` Element

Description: The purpose of the `project` element is to act as the placeholder for the name of the software project of which the user story book is a result of.

Associated USIU: Project.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): None.

Parent Element(s): `usml`.

Child Element(s): None.

Example:

```
<project>Registrar Information System</project>
```

The `related-user-story` Element

Description: The purpose of the `related-user-story` element is to act as a placeholder for expressing any binary relationships between two user stories, namely the given user story and other user stories.

Associated USIU: None.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..Unbounded.

Attribute(s): `type` (Required).

Parent Element(s): `user-story`.

Child Element(s): `name` (Required).

Example:

```
<user-story ...>
  <!-- ... -->
  <name>Display Course Schedule on User Agent</name>
  <!-- ... -->
  <related-user-story type="Occurs-After">
    <name>Print Course Schedule</name>
  </related-user-story>
</user-story>
```

Remark 8.7. The issue of relationship(s) between software requirements has been studied under the topic of requirements interaction management [Robinson, Pawlowski, Volkov, 2003; Woit, 2005], which is defined as "the set of activities directed toward the discovery, management, and disposition of critical relationships among sets of requirements".

Remark 8.8. The `related-user-story` element is a manifestation of the use of the CONTAINER ELEMENT[44] pattern.

The `role` Element

Description: The purpose of the `role` element is to state the (user) role towards which (the realization of) the user story is being targeted.

Associated USIU: Role.

Data Type: Singleton.

[44] URL: http://www.xmlpatterns.com/ .

Occurrence: Required.

Cardinality Constraints: 1.

Attribute(s): `persona` (Optional), `type` (Optional).

Parent Element(s): `statement`.

Child Element(s): `term` (Optional).

Example:

```
A <role type="Positive">bank customer</role> can <goal>withdraw money
from a bank account</goal>.
```

The `statement` Element

Description: The purpose of the `statement` element is to state the user story in the form of a requirement.

Associated USIU: Statement.

Data Type: Singleton.

Occurrence: Required.

Cardinality Constraints: 1.

Attribute(s): `xml:id` (Optional).

Parent Element(s): `user-story`.

Child Element(s): `goal` (Required), `role` (Required), `term` (Optional), `value` (Optional).

Example:

```
<statement>
  A <role>bank customer</role> can <goal>withdraw money from a bank
  account</goal>.
</statement>
```

The `status` Element

Description: The purpose of the `status` element is to indicate the authoring status of a user story.

Associated USIU: Status.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): None.

Parent Element(s): `user-story`.

Child Element(s): None.

Example:

```
<status>Incomplete</status>
```

The `term` Element

Description: The purpose of the `term` element is to acknowledge the presence of terms specific to some application (or problem) domain in a user story description.

Associated USIU: None.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..Unbounded.

Attribute(s): `abbreviation` (Conditional), `expansion` (Conditional), `reference` (Optional).

Parent Element(s): `constraint`, `goal`, `role`, `value`.

Child Element(s): None.

Example:

```
<term expansion="Automated Teller Machine">ATM</term>
```

Remark 8.9. It is evident that a user story description, by necessity, will have terms specific to some application (problem) domain. For example, these terms may include domain-specific abbreviations. A markup of such terms enables subsequent processing if needed.

Remark 8.10. The information foraging theory [Pirolli, 2007] is based on the notion that people forage for information in much the same way that human ancestors in the past and animals forage for food. The most important concept in information foraging theory is the metaphor 'information scent', that is, a cue in an information environment. The presence of the `term` element contributes towards enabling information scent for user story consumers.

The `test` Element

Description: The purpose of the `test` element is to state a user story acceptance test.

Associated USIU: Acceptance Criteria.

Data Type: Singleton (in case of basic test style), Aggregate (in case of BDD test style).

Occurrence: Required.

Cardinality Constraints: 1..Unbounded.

Attribute(s): `xml:id` (Required).

Parent Element(s): `acceptance-criteria`.

Child Element(s): `event` (Conditional), `name` (Conditional), `pre-condition` (Conditional), `post-condition` (Conditional).

Example:

```
<test xml:id="T10">

  <name>Check Timely Customer Response to Type of Transaction</name>

  <pre-condition>
    The bank customer has inserted the bank card.
  </pre-condition>
  <pre-condition>
    The bank customer has entered the PIN. (The PIN is valid.)
  </pre-condition>
  <pre-condition>
    The bank customer is presented with a number of choices for the
    types of transactions to select one from.
  </pre-condition>

  <event>
    There is no response from the bank customer for 15 seconds.
  </event>
  <event>
    The bank customer is reminded to respond within 5 seconds.
  </event>
  <event>
    There is no response from the bank customer for another 5 seconds.
  </event>

  <post-condition>
    The bank customer's card is retained.
  </post-condition>
  <post-condition>
    The bank customer is prompted to contact the bank for further
    deliberation.
  </post-condition>

</test>
```

Remark 8.11. The order in which the tests are listed is not significant. However, a logical order of reading needs to be preserved. It is beyond the scope of USML to suggest or constrain such an order.

The `theme` Element

Description: The purpose of the `theme` element is to state the theme to which a user story belongs to.

Associated USIU: Theme.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): None.

Parent Element(s): `user-story`.

Child Element(s): None.

Example:

```
<theme>Course Schedule</theme>
```

The `title` Element

Description: The purpose of the `title` element is to view the user story book as a 'document', and provide a title for it.

Associated USIU: None.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): None.

Parent Element(s): `usml`.

Child Element(s): None.

Example:

```
<title>User Stories for Registrar Information System</title>
```

Remark 8.12. The need for a title in a document aimed for consumption on the Web has been emphasized in [Caldwell, Chisholm, Reid, Vanderheiden, 2008].

The `user-story` Element

Description: The purpose of the `user-story` element is to contain all information pertaining to a single (individual) user story.

Associated USIU: None.

Data Type: Aggregate.

Occurrence: Required.

Cardinality Constraints: 1..Unbounded.

Attribute(s): `type` (Optional), `version` (Required), `xml:id` (Required).

Parent Element(s): `usml`.

Child Element(s): `acceptance-criteria` (Required), `author` (Required), `constraint` (Optional), `context-of-use` (Optional), `date` (Required), `estimate` (Required), `iteration` (Optional), `name` (Required), `note` (Optional), `priority` (Required), `related-user-story` (Optional), `statement` (Required), `status` (Optional), `theme` (Optional).

Example:

```
<user-story xml:id="CMS-US-1"
            version="0.1"
            type="Positive">
  <!-- ... -->
</user-story>
```

Remark 8.13. The `user-story` element is a manifestation of the use of the CONTAINER ELEMENT[45] pattern.

The `usml` Element

Description: The purpose of the `usml` element is to be the root element of USML. (Every XML instance document must have a unique root element.)

Associated USIU: None.

Data Type: Aggregate.

Occurrence: Required.

Cardinality Constraints: 1.

Attribute(s): `version` (Required), `xml:lang` (Required), `xml:space` (Optional).

Parent Element(s): None.

Child Element(s): `domain` (Optional), `methodology` (Optional), `organization` (Optional), `project` (Optional), `license` (Optional), `title` (Optional), `user-story` (Required).

[45] URL: http://www.xmlpatterns.com/ .

Example:

```
<usml version="1.0" xml:lang="en">
  <user-story>
    <!-- ... -->
  </user-story>
</usml>
```

The `value` Element

Description: The purpose of the `value` element is to underscore the value provided by the user story to the (user) role.

Associated USIU: Value.

Data Type: Singleton.

Occurrence: Optional.

Cardinality Constraints: 0..1.

Attribute(s): None.

Parent Element(s): `statement`.

Child Element(s): `term` (Optional).

Example:

```
A <role>student</role> can <goal>search the system</goal> for
<value>employment opportunities</value>.
```

8.2. USML Attributes

This section outlines salient characteristics common to USML attributes, and subsequently provides the definitions of each of those attributes.

8.2.1. Data Types of USML Attributes

The data types of USML attributes determine the type of data allowed in attribute values. For the types of character data in attribute values, a library of data types, such as the one provided by XSDL, has been used. The exact data types are specified in the following sections that provide definitions of individual attributes.

8.2.2. Definitions of USML Attributes

The attributes of USML, ordered alphabetically, are:

- `abbreviation`
- `approach` (of estimate)
- `approach` (of priority)
- `criterion`
- `expansion`
- `metric`
- `model`
- `persona`
- `range`
- `reference`
- `type` (of author)
- `type` (of constraint)

- `type` (of license)
- `type` (of methodology)
- `type` (of priority)
- `type` (of role)
- `type` (of user story)
- `type` (of user story relationship)
- `version`
- `xml:id`
- `xml:lang`
- `xml:space`

In the following, the definitions of all attributes of USML are given. The definitions are ordered alphabetically.

The `abbreviation` Attribute

Description: The purpose of the `abbreviation` attribute is to provide an abbreviation, if one exists or is otherwise deemed necessary, for a term.

Associated USIU: None.

Data Type: xsd:token.

Related Element(s): `term`.

Example:

```
<term abbreviation="ATM">Automated Teller Machine</term>
```

The `approach` (of estimate) Attribute

Description: The purpose of the `approach` attribute is to name the approach used for estimation.

Associated USIU: Estimate.

Data Type: xsd:token.

Related Element(s): `estimate`.

Example:

```
<estimate approach="Planning Poker" ...>
  <!-- ... -->
</estimate>
```

The `approach` (of priority) Attribute

Description: The purpose of the `approach` attribute is to name the approach used for prioritization.

Associated USIU: Priority.

Data Type: xsd:token.

Related Element(s): `priority`.

Example:

```
<priority approach="MoSCoW" ...>
  <!-- ... -->
</priority>
```

The `criterion` Attribute

Description: The purpose of the `criterion` attribute is to list (one or more) criteria used for prioritization.

Associated USIU: Priority.

Data Type: xsd:token.

Related Element(s): `priority`.

Example:

```
<priority criterion="Risk Value" ...>
  <!-- ... -->
</priority>
```

The `expansion` Attribute

Description: The purpose of the `expansion` attribute is to provide the expansion of a term that has been expressed as an abbreviation.

Associated USIU: None.

Data Type: xsd:token.

Related Element(s): `term`.

Example:

```
<term expansion="Video-On-Demand">VOD</term>
```

The `metric` Attribute

Description: The purpose of the `metric` attribute is to name the metric used for estimation.

Associated USIU: Estimate.

Data Type: xsd:token.

Related Element(s): `estimate`.

Example:

```
<estimate metric="Story Points" ...>
  <!-- ... -->
</estimate>
```

The `model` Attribute

Description: The purpose of the `model` attribute is to name the maturity model corresponding to the maturity level. The maturity model could be related to software process, requirements engineering process, or user story process.

Associated USIU: Maturity.

Data Type: xsd:token.

Related Element(s): `maturity`.

Example:

```
<maturity model="SMM">Level 1</maturity>
```

The `persona` Attribute

Description: The purpose of the `persona` attribute is to name a persona corresponding to the (user) role. The `persona` attribute is optional as it may be difficult to elicit a persona for a negative (user) role.

Associated USIU: Persona.

Data Type: xsd:token.

Related Element(s): `role`.

Example:

```
<role persona="John Smith">student</role>
```

The `range` Attribute

Description: The purpose of the `range` attribute is to name the sequence of numbers used for estimation.

Associated USIU: Estimate.

Data Type: xsd:token.

Related Element(s): `estimate`.

Example:

```
<estimate range="Fibonacci Sequence" ...>
  <!-- ... -->
</estimate>
```

The `reference` Attribute

Description: The purpose of the `reference` attribute is to act as a placeholder for a relationship between a source (that is, the corresponding element) and some target. (The target may be an animate or an inanimate entity.) This relationship can be expressed as an absolute or relative URI.

Associated USIU: None.

Data Type: xsd:anyURI.

Related Element(s): `author`, `context-of-use`, `goal`, `license`, `methodology`, `name`, `related-user-story`, `role`, `term`, `test`, `value`.

Example: The following has an absolute URI:

```
<license reference="http://www.gnu.org/licenses/gpl-3.0.html">
  <!-- ... -->
</license>
```

The following has a relative URI:

```
<role reference="student.html">Student</role>
```

Remark 8.14. A link is an implementation of a relationship between two resources, where one resource refers to the other resource by means of a URI [Jacobs, Walsh, 2004]. The origin of the notion of a link in the aforementioned sense dates back to the 1940s [Bush, 1945].

Remark 8.15. The XML Linking Language (XLink) [DeRose, Maler, Orchard, Walsh, 2010] and its companion XML Base [Marsh, Tobin, 2009] enable hyperlinks in XML instance documents. For example, XLink could be used to express relationships between user stories, or between user stories and other software artifacts, as hyperlinks that are 'richer' and more sophisticated than those possible in HTML or XHTML. However, for a number of reasons, including a sustained lack of support of XLink in user agents, USML does not natively use XLink. If needed, it is possible to extend USML to include XLink elements and/or attributes. This is shown in Example 10.5.

The **`type`** (of author) Attribute

Description: The user story authors can be classified in different ways, as discussed in Section 4.4.3. The purpose of the `type` attribute is to act as a placeholder for the type of user story author.

Associated USIU: None.

Data Type: (Internal | External) | (Software Engineer | Customer | User).

Related Element(s): `author`.

Example:

```
<author type="Software Engineer">Jane Marshall</author>
```

The **`type`** (of constraint) Attribute

Description: There are different kinds of constraints that a user story statement may be subjected to. The purpose of the `type` attribute is to act as a placeholder for the type of constraint. The `type` attribute is optional as the type of a given constraint may not always have a known label.

Associated USIU: Constraint.

Data Type: xsd:token.

Related Element(s): `constraint`.

Example:

```
<constraint type="Usability">...</constraint>
```

The `type` (of license) Attribute

Description: There are different kinds of licenses under which a user story book can be released. The purpose of the `type` attribute is to act as a placeholder for the type of license.

Associated USIU: License.

Data Type: xsd:token.

Related Element(s): `usml`.

Example:

```
<license reference="http://www.gnu.org/licenses/gpl-3.0.html"
         type="Free">
  <!-- ... -->
</license>
```

The `type` (of methodology) Attribute

Description: There are different kinds of methodologies under which a user story can be developed, as discussed in Section 5.4. The purpose of the `type` attribute is to be a placeholder for the type of methodology.

Associated USIU: Methodology.

Data Type: xsd:token.

Related Element(s): `methodology`.

Example:

```
<methodology type="Agile">XP</methodology>
```

The `type` (of priority) Attribute

Description: There are different kinds of prioritization schemes, as discussed in Section 5.5. The purpose of the `type` attribute is to be a placeholder for the type of priority.

Associated USIU: Priority.

Data Type: Absolute | Relative.

Related Element(s): `priority`.

Example:

```
<priority type="Absolute" ...>
  <!-- ... -->
</priority>
```

The `type` (of role) Attribute

Description: There are different kinds of (user) roles, as discussed in Section 4.5. The purpose of the `type` attribute is to be a placeholder for the type of (user) role.

Associated USIU: Role.

Data Type: Positive | Negative.

Related Element(s): `role`.

Example:

```
<role type="Positive">traveler</role>
```

The `type` (of user story) Attribute

Description: There are different kinds of user stories, as discussed in Section 5.5. The purpose of the `type` attribute is to be a placeholder for the type of user story.

Associated USIU: None.

Data Type: Positive | Negative.

Related Element(s): `user-story`.

Example:

```
<user-story xml:id="CMS-US-1"
            version="0.1"
            type="Positive">
  <!-- ... -->
</user-story>
```

The `type` (of user story relationship) Attribute

Description: The purpose of the `type` attribute is to act as a placeholder for one or more types of user story relationships. For example, using the classification of [Robinson, Pawlowski, Volkov, 2003; Woit, 2005], the different kinds of possible relationships between user stories can be: Similar-To, Resource-Contender-Of, Consequence-Of, Required-By, Occurs-Before, Occurs-Concurrently-With, and Occurs-After. The direction of relationship, in cases where it is relevant, is from the related user story to the given user story.

Associated USIU: None.

Data Type: xsd:token.

Related Element(s): `related-user-story`.

Example:

```
<related-user-story type="Similar-To Occurs-After">
  <!-- ... -->
</related-user-story>
```

The `version` Attribute

Description: The purpose of the `version` attribute is to provide a numerical value of the release date of an artifact. For example, such an artifact could be the USML specification or a specific user story. USML or a user story can evolve for a number of reasons, and this is acknowledged explicitly by a presence of the `version` attribute.

Associated USIU: None.

Data Type: xsd:decimal.

Related Element(s): `usml`, `user-story`.

Example:

```
<usml version="1.0" xml:lang="en">
  <user-story version="1.2">
    <!-- ... -->
  </user-story>
</usml>
```

However, the following is not valid:

```
<user-story version="1.">
  <!-- ... -->
</user-story>
```

Remark 8.16. For the `version` attribute in the `usml` element, for example, the value 1.0 is allowed; however, 0.1 is not. This constraint is not expressed in the RELAX NG schema for USML but, along with other constraints related to versioning, is expressed by the Schematron schema for USML.

Remark 8.17. The significance of versioning has been realized in different but related contexts. In general, versioning is relevant for managing the variability of information on the Web [Berners-Lee, Connolly, 1998]. In particular, providing a mechanism for version information is considered to be one of the 'Good Practices' of a specification aiming towards increasing the value of the Web [Jacobs, Walsh, 2004]. The association of a version with a requirement has been recommended by the IEEE Standard 830-1998 [IEEE, 1998]. It has been suggested [Harold, 2003, Item 26] that version information be associated with those XML instance documents, schema documents, and style sheets that are prone to evolve.

Remark 8.18. It is expected that a USMS has support for a version management system. It may appear that the use of a version management system for tracking a user story will render the `version` attribute redundant. However, USML aims to be independent of any USMS. Furthermore, the PRIVATE VERSIONS pattern [Berczuk, Appleton, 2003] suggests a versioning mechanism independent of a version management system.

The `xml:id` Attribute

Description: There is a need to uniquely identify certain USML elements. The `xml:id` specification [Marsh, Veillard, Walsh, 2005], an ancillary to XML, provides a native attribute, namely `xml:id`, for such a purpose. The identifier must be unique, at least across the user story book for the same software project. The `xml:id` attribute is mandatory for elements that can occur multiple times in a user story or in a USML instance document. For example, it is possible to have multiple constraints in a user story, or multiple user stories in a USML instance document. There are certain restrictions placed by XML on the type of values that can be used for `xml:id`. For example, the identifier could be an alphanumeric string.

Associated USIU: None.

Data Type: xsd:ID.

Related Element(s): `constraint`, `role`, `statement`, `test`, `user-story`.

Example:

```
<user-story xml:id="CMS-US-1"
            version="0.1"
            type="Positive">
  <!-- ... -->
</user-story>
```

However, the following is not valid:

```
<test xml:id="1">
  <!-- ... -->
</test>
```

Remark 8.19. The activity of "uniquely identify[ing]" a user story is included in the SMM Level 4, Key Process Area 4.3 [Patel, Ramachandran, 2009b].

The `xml:lang` Attribute

Description: There is a need to identify the natural language of the contents of certain USML elements. For example, such a natural language could be English or French. The XML specification provides a native attribute, namely `xml:lang`, for such a purpose. The value of the `xml:lang` attribute is a language tag defined by the IETF RFC 5646 [Phillips, Davis, 2009]. The Internet Assigned Numbers Authority (IANA) Language

Subtag Registry[46] maintains a list of language tags. Let E be a USML element with the `xml:lang` attribute. The value of the `xml:lang` attribute, unless overridden by another instance of the `xml:lang` attribute, applies to (1) any child elements of E, and (2) attribute values associated with E. Therefore, it is not necessary to include the `xml:lang` attribute in every element's start-tag.

Associated USIU: None.

Data Type: xsd:language.

Related Element(s): `usml`.

Example:

```
<usml version="1.0" xml:lang="en">
  <!-- ... -->
</usml>
```

The `xml:space` Attribute

Description: In XML, white space (such as space, tab, blank line) is used to enhance readability of an XML instance document [Bray, Paoli, Sperberg-McQueen, Maler, Yergeau, 2008, Section 2.10; Harold, 2003, Item 10]. Furthermore, in XML, and by reference USML, white space can be significant in elements, unless otherwise specified. The `xml:space` attribute is used to indicate to an XML processor to preserve white space while processing a USML instance document.

Associated USIU: None.

Data Type: preserve.

Related Element(s): `usml`.

[46] URL: http://www.iana.org/assignments/language-subtag-registry .

Example:

```
<usml version="1.0"
      xml:lang="en"
      xml:space="preserve">
  <!-- ... -->
</usml>
```

8.3. Summary

The elements and attributes of USML determine, to a large extent, the capability of USML. There are elements and attributes in USML for both a user story book and a user story. The elements and attributes of a user story book apply to the entire collection of user stories in a USML instance document. USML reuses certain attributes that are defined elsewhere in XML and its ancillary specifications.

Chapter 9 Using USML

There's more than one way to do it.
— Larry Wall

The potential of a language can, to a certain extent, be demonstrated and assessed by the depth and breadth of its use, and USML is no exception.

This chapter explores a number of different, but related, ways in which USML can be used by different user story stakeholders. The focus of the coverage is on variety rather than pedagogy.

9.1. User Story Process and User Story Description in USML

The user story authors need to develop a user story description at some stage of a user story process that, in turn, can be part of a software development methodology. In [Kamthan, Shahmir, 2010], a user story development process model is proposed. USML can serve as input to such a process, as illustrated in Figure 9.1.

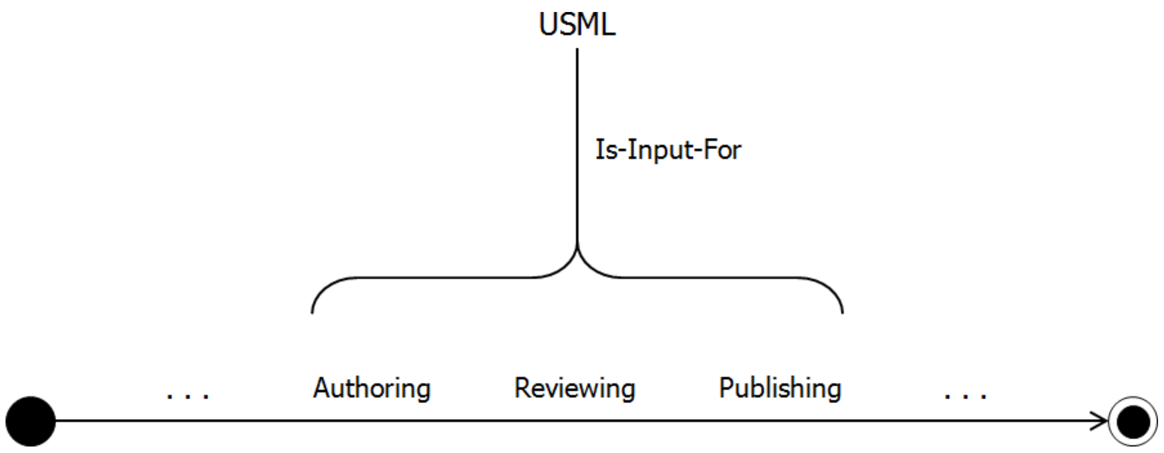


Figure 9.1. The relationship between certain stages of a user story development process and USML.

For example, USML instance documents could, in certain cases, be a complement, if not an alternate, to paper-based index cards.

9.2. User Story Forms in USML

USML provides support for user story forms, as illustrated in Figure 9.2. In this thesis, a user story form based on USML is called a USML user story form.

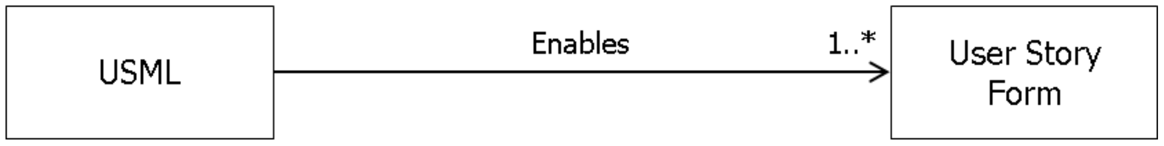


Figure 9.2. The relationship between USML and user story form.

A USML instance document is based on some USML user story form, as illustrated in Figure 9.3.

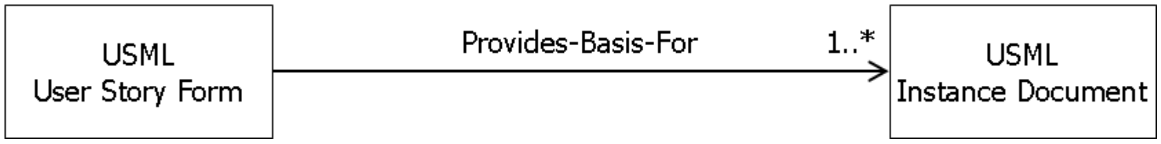


Figure 9.3. The relationship between USML user story form and USML instance document.

9.2.1. Understanding User Story Form in USML

There are multiple possible views of a USML user story form, two of which are illustrated in Figures 9.4 and 9.5.

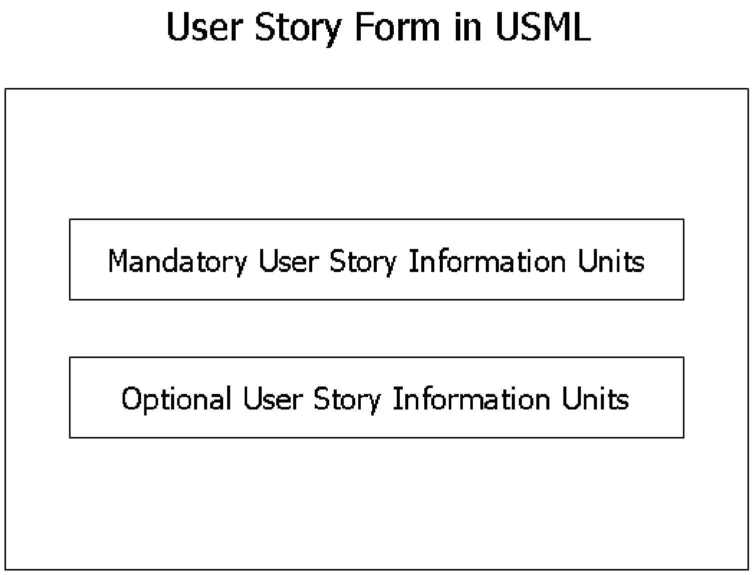

User Story Form in USML
Mandatory User Story Information Units
Optional User Story Information Units

Figure 9.4. A USML user story form from the perspective of occurrence of user story information units.

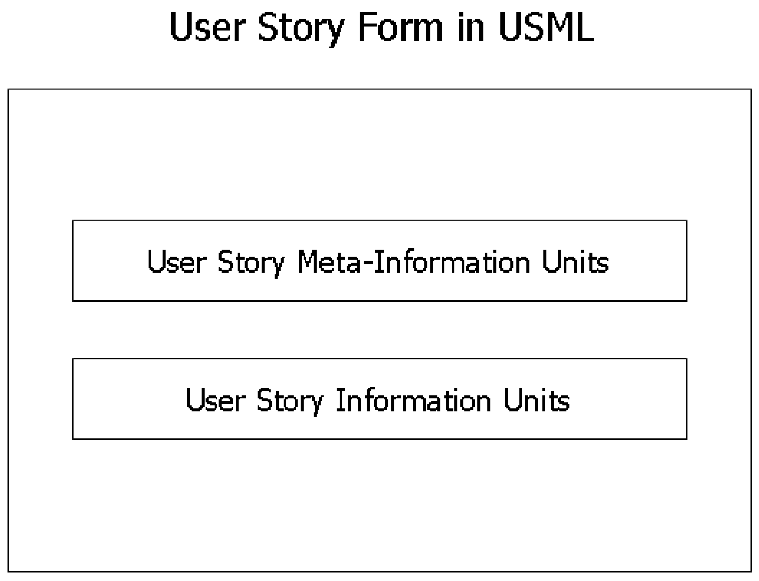
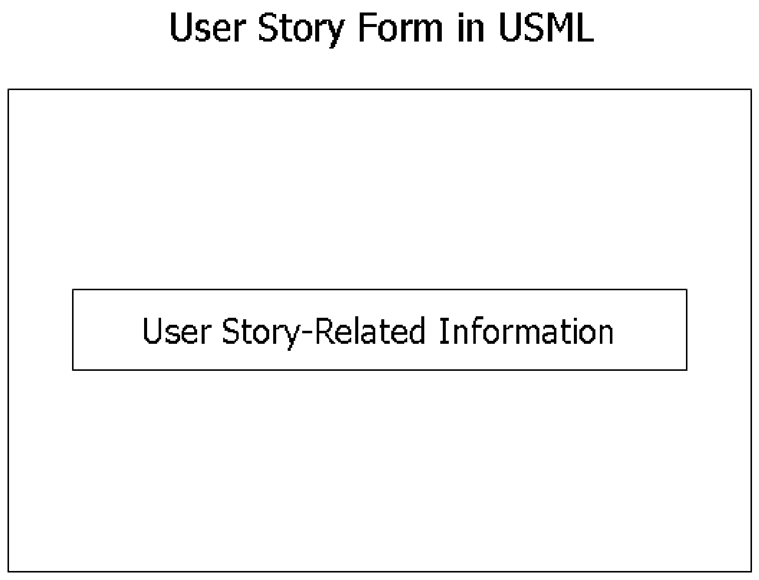
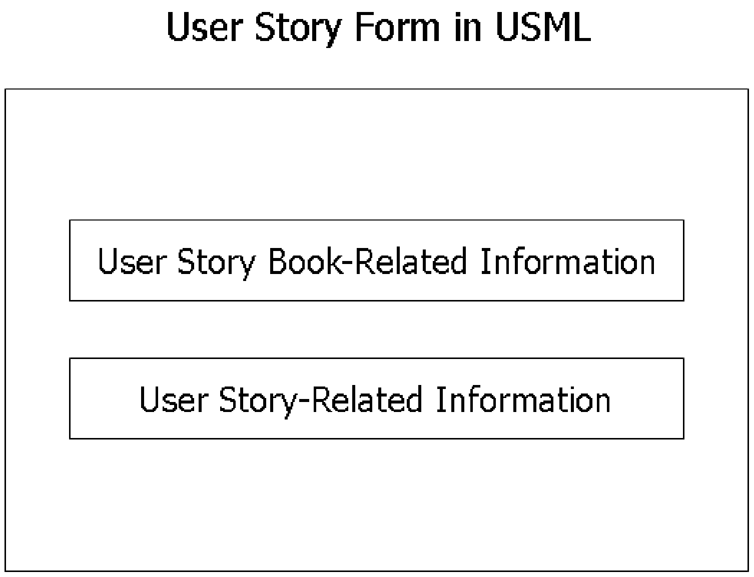

User Story Form in USML
User Story Meta-Information Units
User Story Information Units

Figure 9.5. A USML user story form from the perspective of nature (type) of user story information units.

In USML, it is possible to devise user story forms that are increasingly elaborate in the sense of the user story information units each of them includes. A typical collection of USML user story forms is illustrated in Figure 9.6.



(a)


User Story Form in USML
User Story Book-Related Information
User Story-Related Information

(b)

Figure 9.6. A collection of USML user story forms in increasing order of number and/or type of user story information units.

A USML instance document corresponding to Figure 9.6(a) has the following form:

```
<?xml version="1.0" encoding="UTF-8"?>
<usml version="1.0" xml:lang="en">

<user-story ...>
  <!-- Elements and Attributes for User Story -->
</user-story>

</usml>
```

Remark 9.1. The minimal conforming USML instance document is based on the simplest of the USML user story forms, namely that illustrated in Figure 9.6(a).

A USML instance document corresponding to Figure 9.6(b) has the following form:

```
<?xml version="1.0" encoding="UTF-8"?>
<usml version="1.0" xml:lang="en">

<!-- Elements and Attributes for User Story Book -->

<user-story ...>
  <!-- Elements and Attributes for User Story -->
</user-story>

</usml>
```

9.2.2. An ‘Extension’ of User Story Form in USML

A non-trivial software system can have a large number of user stories. The user story book-related information applies to each of these user stories.

If there is one USML instance document for each user story description, such as that based on Figure 9.6(b), then it leads to two management issues: (1) the user story book-related information will repeat across documents, and (2) there will be multiple files, one for each user story.

To avoid this, USML allows multiple user stories in the same USML instance document, as illustrated in Figure 9.7. This arrangement could be seen as an ‘extension’ of the notion of a user story form as given in Chapter 3.

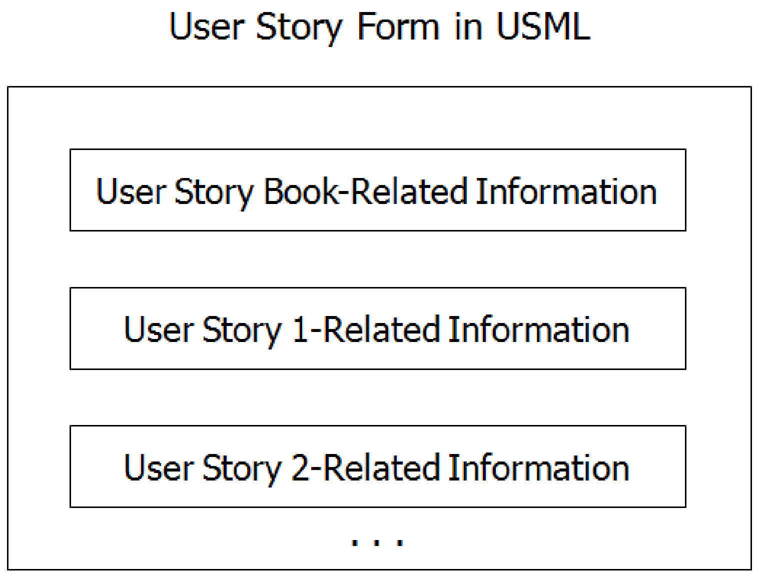

User Story Form in USML
User Story Book-Related Information
User Story 1-Related Information
User Story 2-Related Information
. . .

Figure 9.7. An ‘extended’ USML user story form for multiple user story descriptions.

A USML instance document corresponding to Figure 9.7 has the following form:

```
<?xml version="1.0" encoding="UTF-8"?>
<usml version="1.0" xml:lang="en">

<!-- Elements and Attributes for User Story Book -->

<user-story ...>
  <!-- Elements and Attributes for User Story -->
</user-story>

<user-story ...>
  <!-- Elements and Attributes for User Story -->
</user-story>

<!-- ... -->

</usml>
```

Remark 9.2. The activity of “[defining] a standard structure” for a user story is included in the SMM Level 2, Key Process Area 2.2 [Patel, Ramachandran, 2009b]. USML is not a ‘standard’; however, a USML user story form does impose a uniform structure on user stories represented in USML.

Remark 9.3. The need to communicate software requirements at different levels of abstraction, to target different stakeholders, has been pointed out previously [Ambler, 2002; Firesmith, 2003; Park, Maurer, 2008; Zhang, Arvela, Berki, Muhonen, Nummenmaa, Poranen, 2010]. USML user story forms are a realization of the need to communicate user stories at different levels of abstraction.

9.3. USML and Other Markup Languages

A markup fragment (or fragment, for short) is markup in some markup language. For a number of reasons, USML and other markup languages may need to interface. In particular, (1) a fragment in another markup language may need to interface with an instance document of USML and, conversely, (2) a fragment of USML may need to interface with an instance document of another markup language.

This leads to heterogeneous or 'compound' instance documents. Figure 9.8 provides an abstract illustration of these two scenarios.

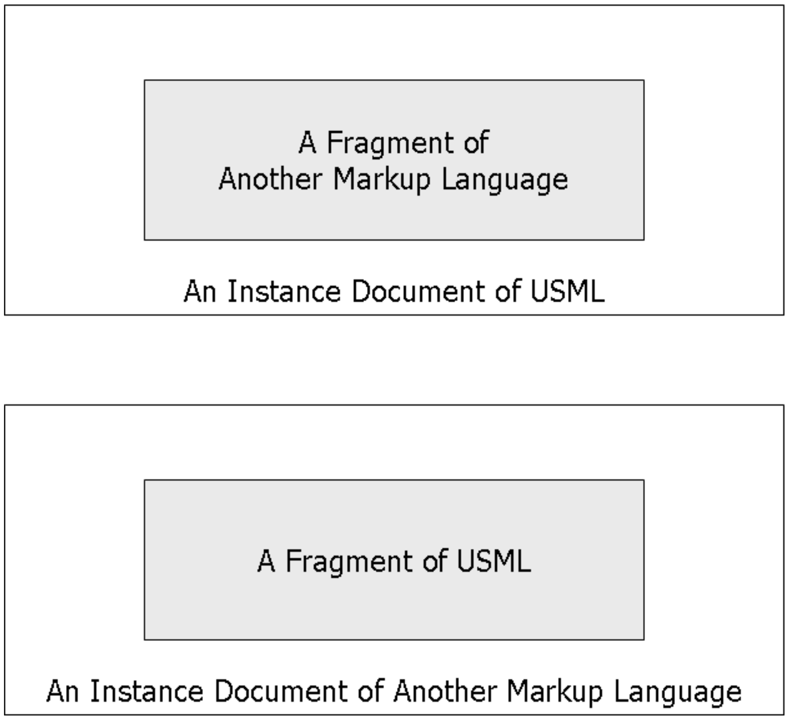

A Fragment of
Another Markup Language
An Instance Document of USML
A Fragment of USML
An Instance Document of Another Markup Language

Figure 9.8. The two different ways in which 'compound' instance documents can be formed.

An example of case (1) is illustrated in Example 10.5. In the following, an example of case (2) is given.

Example 9.1. DocBook XML [Walsh, Hamilton, 2010] is a markup language suitable for representing certain types of documentation, including certain software process artifacts. The following is a 'compound' instance document that includes USML fragment in a DocBook XML article. It uses the USML namespace name and prefix to identify USML elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<article
  xmlns="http://docbook.org/ns/docbook"
  version="5.0"
  xml:lang="en"
  xml:space="preserve"
  xmlns:us="http://users.encs.concordia.ca/~kamthan/usml/1.0">

  <title>USML Tutorial</title>
  <para>
    The following is an example of a user story statement in USML:

    <us:statement>
      A <us:role>student</us:role> can <us:goal>register for
      courses</us:goal> so as to <us:value>complete program
      requirements</us:value>.
    </us:statement>
  <!-- ... -->
  </para>

</article>
```

9.4. Presenting USML Instance Documents

There are a number of reasons that motivate the issue of presentation of USML instance documents:

- **Absence of Presentational Elements.** In this thesis, representation and presentation of a user story description are orthogonal concerns. USML is based on the principles of descriptive markup and, as such, does not include any presentational elements. Therefore, the responsibility of presenting a USML instance document needs to be relegated elsewhere.

- **Variations among User Story Stakeholders.** It can not always be expected that all user story stakeholders, specifically external user story authors, will browse or read the source of an USML instance document. For example, a customer or user may not have the background in XML or its ancillary technologies.

This section proposes a conceptual model for presenting a user story description, namely USPM. In this thesis, it is assumed that the presentation is on a two-dimensional electronic surface, and the sensory modality is visual.

9.4.1. Goals and Strategy

The goals of USPM are to lend support to USDM and to demonstrate some of the capabilities of USML. USPM aims to be independent of any specific device or tool.

USPM is general and suggests high-level decisions, both by necessity and by choice. It is not the intention of USPM to mandate low-level orthographical or typographical decisions. For example, it is not the purpose of USPM to suggest a specific emphasis or typeface.

USPM is based on the following strategy:

- **Identify Concerns.** In presenting a user story, there are a number of concerns. USPM identifies the major components in a presentation environment that are relevant to user story stakeholders.

- **Relate Representation and Presentation.** The purpose of USPM is to make a USML instance document readable to non-technical user story stakeholders. Therefore, the relationship between representation and presentation needs to be shown.

- **Make Technological Commitment.** There are different technologies that support presentation. They need to be identified and compared. This provides necessary rationale for selecting the one that is most appropriate for presenting a user story description.

Remark 9.4. USPM is inspired by previous work on presentation of pattern descriptions [Kamthan, 2011b].

9.4.2. Representation to Presentation via Transformation

It is possible to produce a presentation from a representation of a user story description via transformation, as illustrated in Figure 9.9.

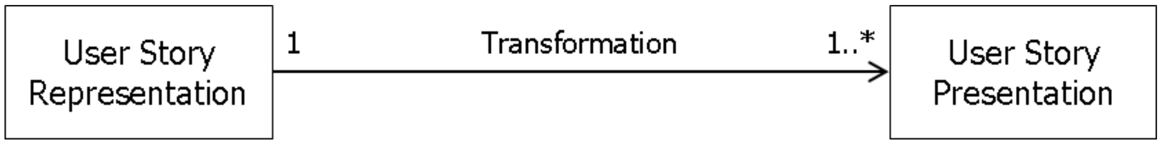


Figure 9.9. A user story representation can be transformed to one or more user story presentations.

This 'one-representation-to-many-presentations' is a direct consequence of one of the principles of descriptive markup [Goldfarb, 1981]. It is also a manifestation of the single source approach as suggested by the SINGLE SOURCE pattern [Correia, Ferreira, Flores, Aguiar, 2009] and the SINGLE SOURCE AND MULTIPLE TARGETS pattern [Rüping, 2003].

The ability to produce multiple presentations from a single representation has a number of advantages. It reduces a nonlinear (specifically, quadratic) problem to a linear problem[47]. If carried out automatically, partially or completely, it reduces effort on part of the user story producers towards modifying and publishing a user story description. Finally, it helps address different presentation needs of different user story consumers, a characteristic desirable also of a 'modern' software requirements specification [Firesmith, 2003].

Definition 9.1 [Transformer]. The thing that transforms a given representation to a corresponding presentation.

For example, a transformer could be a style sheet or a program.

There can be different types of transformations. A transformation from descriptive markup to presentational markup is a down transformation. A transformation T in which the structure of the target is different from that of the source, such that the change is permanent, is a permanent transformation; otherwise T is a temporary transformation.

[47] A collection of M sources and N targets will require M·N transformations. A single source approach requires N transformations.

There are, as outlined in Table 9.1, a number of considerations in presenting a user story via transformation.

Item	**Context**	**Example(s)**
Representation	Type of Representation Device	Notebook Computer
	Type of Representation Language	USML
	Type of Representation	USML Instance Document
Presentation	Type of Presentation Device	Notebook Computer, Tablet Computer, Mobile Device, Text Terminal, Printer
	Type of Presentation Processor	Shell, User Agent, User Agent Emulator
	Type of Presentation Language	HTML, XHTML Basic
	Type of Presentation	Block, 'Index Card', Thumbnail, Table
Transformation	Type of Transformation	Down; Temporary, Permanent
	Type of Transformer	Style Sheet, Program
	Type of Transformer Language	CSS, XSLT
	Type of Transformer Processor	Command Line Program, User Agent, User Agent Emulator, Web Application
	Type of Transformer Processor Device	Notebook Computer, Tablet Computer

Table 9.1. A summary of concerns related to the USPM.

Remark 9.5. It could be noted that not all considerations in Table 8.1 are necessary for every transformation. (This becomes evident from examples that follow.)

Remark 9.6. Table 9.1 is applicable beyond the realms of USPM. In particular, it applies to representation markup languages other than USML.

9.4.3. Representation to Presentation via Style Sheet

It is possible to conduct temporary transformations of USML instance documents using Cascading Style Sheets (CSS) [Bos, Çelik, Hickson, Lie, 2011].

It is possible to conduct permanent transformations of USML instance documents using Extensible Stylesheet Language Transformations (XSLT) [Kay, 2007]. In XSLT, the source must be an XML instance document, and the target could be an XML instance document, HTML instance document, or text.

Finally, it is possible to use a combination of CSS and XSLT, each for a different purpose: XSLT for transforming and CSS for styling. Figure 9.10 illustrates these possibilities.

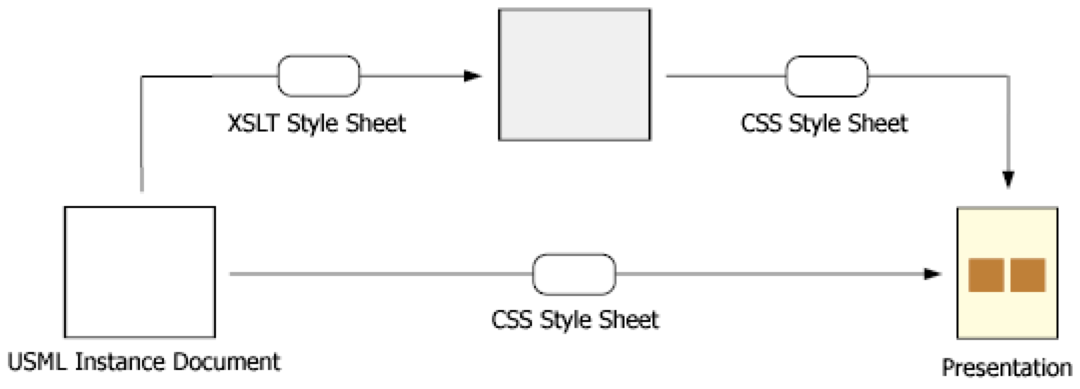


Figure 9.10. The different style sheet language-based approaches for presenting a USML instance document.

9.4.4. Associating the Transformer with the Source

The means for associating a style sheet with an instance document of a (descriptive) markup language based on XML is specified via a processing instruction [Clark, 1999].

For example, consider a USML instance document, a CSS style sheet (say, `example.css`), and an XSLT style sheet (say, `example.xsl`), all residing in the same directory. The USML instance document is aimed for presentation on computer monitor screen. For associating the CSS style sheet with the USML instance document, the following processing instruction can be used:

```
<?xml-stylesheet type="text/css" href="example.css" media="screen"?>
```

A similar argument applies in case of XSLT. For associating the XSLT style sheet with the USML instance document, the following processing instruction can be used:

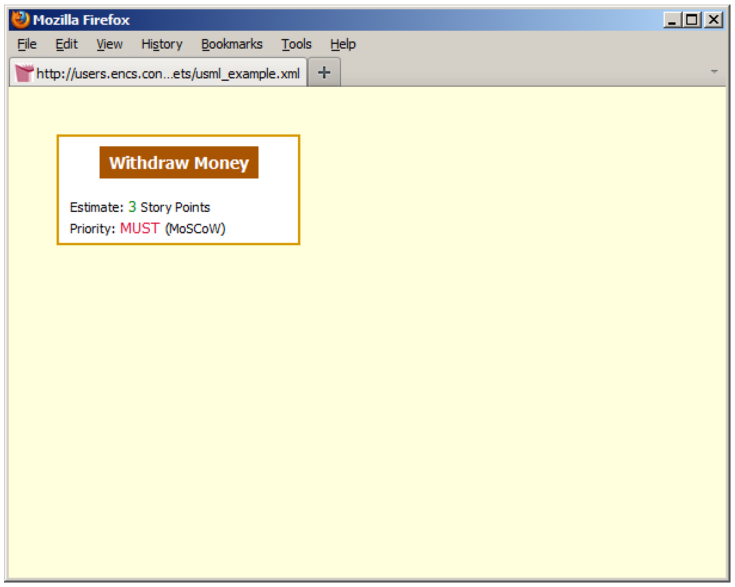
```
<?xml-stylesheet type="application/xml" href="example.xsl"?>
```

It could be noted that recommended media types can vary across different versions of the same language.

9.4.5. Examples

In this section, there are three examples of presenting USML instance documents, each with a different goal and a different approach. These examples make use of a number of tools that are listed in Appendix E.

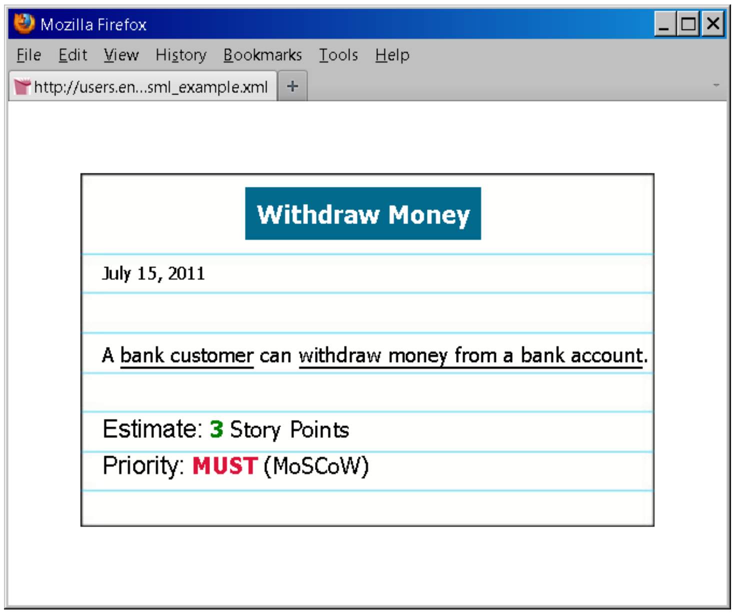
Example 9.2. Figure 9.11 illustrates snapshots of multiple renderings in Mozilla Firefox user agent of the same USML instance document. The USML instance document includes entity declarations and entity references for a special character and a mathematical symbol. The USML instance document is associated with different CSS style sheets, each intended for a specific purpose. The user story name has been emphasized to support a user story browser. The order of presentation of snapshots reflects progressive disclosure. The transformations occur on the client-side. In each case, there is a complete separation of representation and presentation. The sample USML instance document is given in Appendix C, and the associated CSS style sheets for USML are given in Appendix D.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://users.encs.con...ets/usml_example.xml

Withdraw Money

Estimate: 3 Story Points
Priority: MUST (MoSCoW)

(a) A USML instance document presented as a thumbnail.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://users.en...sml_example.xml

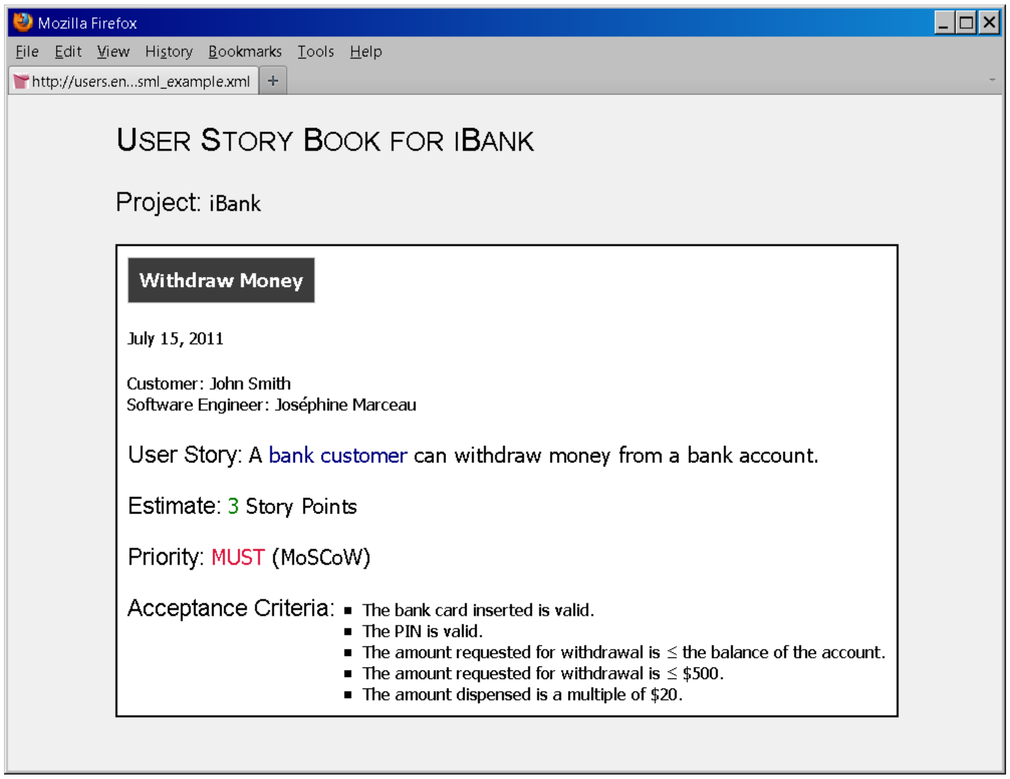
Withdraw Money

July 15, 2011

A bank customer can withdraw money from a bank account.

Estimate: 3 Story Points
Priority: MUST (MoSCoW)

(b) A USML instance document presented as an ‘index card’.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://users.en...sml_example.xml

USER STORY BOOK FOR IBANK

Project: iBank

Withdraw Money

July 15, 2011

Customer: John Smith
Software Engineer: Joséphine Marceau

User Story: A bank customer can withdraw money from a bank account.

Estimate: 3 Story Points

Priority: MUST (MoSCoW)

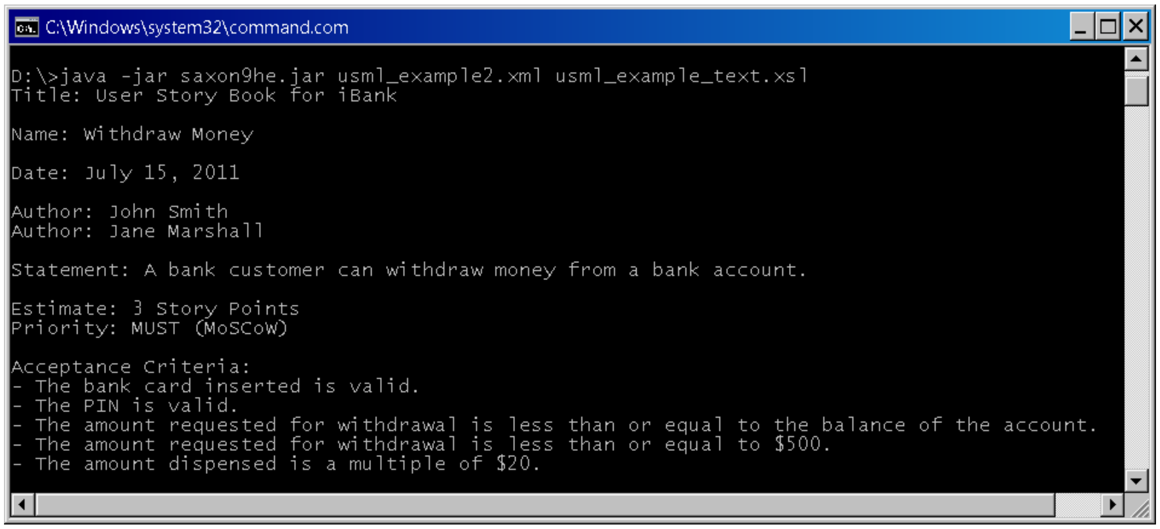
Acceptance Criteria:
- The bank card inserted is valid.
- The PIN is valid.
- The amount requested for withdrawal is ≤ the balance of the account.
- The amount requested for withdrawal is ≤ $500.
- The amount dispensed is a multiple of $20.

(c) A USML instance document presented as a block.

Figure 9.11. A demonstration of the single source approach: 'one user story representation, multiple user story presentations'.

Remark 9.7. CSS is a styling language. In CSS, it is not possible to carry out a number of tasks such as restructuring a USML instance document, adding elements or attributes, or carrying out conditional processing. For such tasks, a general-purpose or a special-purpose programming language can be used. For example, XSLT can serve as a special-purpose programming language for carrying out aforementioned tasks, although there are alternatives.

Example 9.3. Figure 9.12 illustrates a snapshot of rendering in an arbitrary 'plain' text environment[48] of a USML instance document. The USML instance document is associated with an XSLT style sheet. This XSLT style sheet transforms the USML instance document into 'structured' punctuational markup. In particular, the style sheet 'strips' all start-tags and end-tags and, for the sake of formatting, adds text, white space, and newline characters. The XSLT processor used is Saxon. The transformation occurs on the command-line. The sample USML instance document is given in Appendix C, and the associated XSLT style sheet for USML is given in Appendix D.



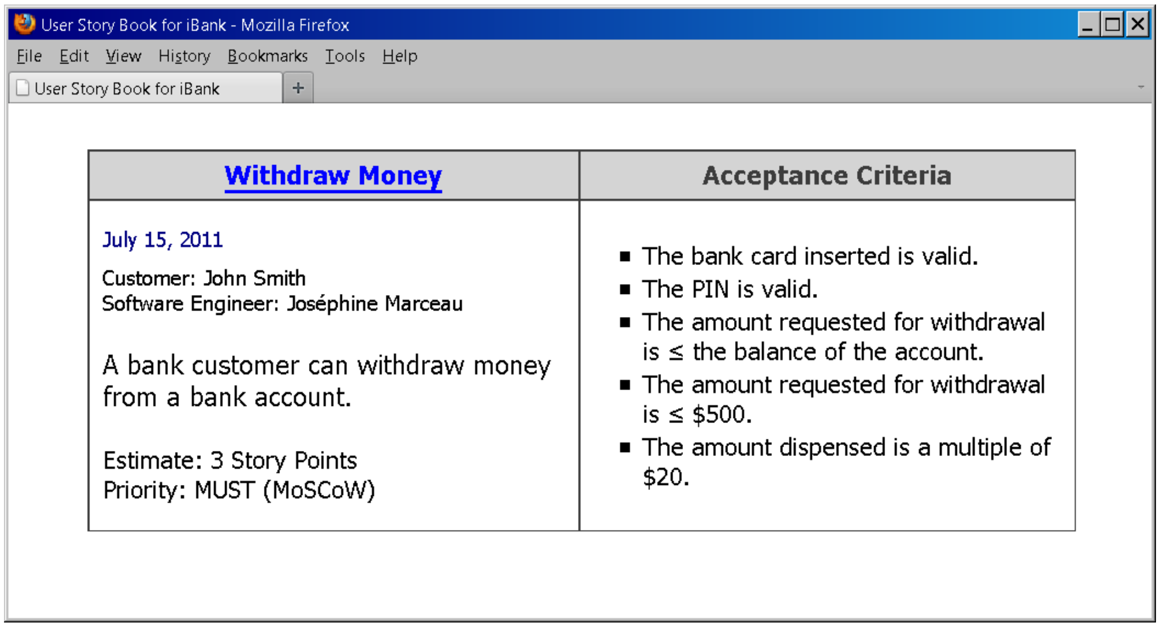
```
D:\>java -jar saxon9he.jar usml_example2.xml usml_example_text.xsl
Title: User Story Book for iBank

Name: Withdraw Money

Date: July 15, 2011

Author: John Smith
Author: Jane Marshall

Statement: A bank customer can withdraw money from a bank account.

Estimate: 3 Story Points
Priority: MUST (MoSCoW)

Acceptance Criteria:
- The bank card inserted is valid.
- The PIN is valid.
- The amount requested for withdrawal is less than or equal to the balance of the account.
- The amount requested for withdrawal is less than or equal to $500.
- The amount dispensed is a multiple of $20.
```

Figure 9.12. A demonstration of a transformation of a user story representation to 'plain' text.

Example 9.4. Figure 9.13 illustrates snapshots of renderings in Mozilla Firefox user agent and in Opera Mobile Emulator[49] of the USML instance document of Example 9.2. The USML instance document is associated with an XSLT style sheet. This XSLT style sheet transforms the USML instance document into an 'intermediary', in-memory

[48] For example, such an environment could be standard output in DOS or UNIX shell.

XHTML Basic[50] instance document. The XHTML Basic instance document is associated with a CSS style sheet. The transformations occur on the client-side. The transformation process resembles the PIPE-AND-FILTER pattern [Buschmann, Meunier, Rohnert, Sommerlad, Stal, 1996]. The XSLT style sheet is a manifestation of the use of the FILL-IN-THE-BLANKS pattern [Kay, 2008]. In each case, there is a complete separation of representation and presentation. The sample USML instance document is given in Appendix C, and the associated XSLT and CSS style sheets for USML are given in Appendix D.

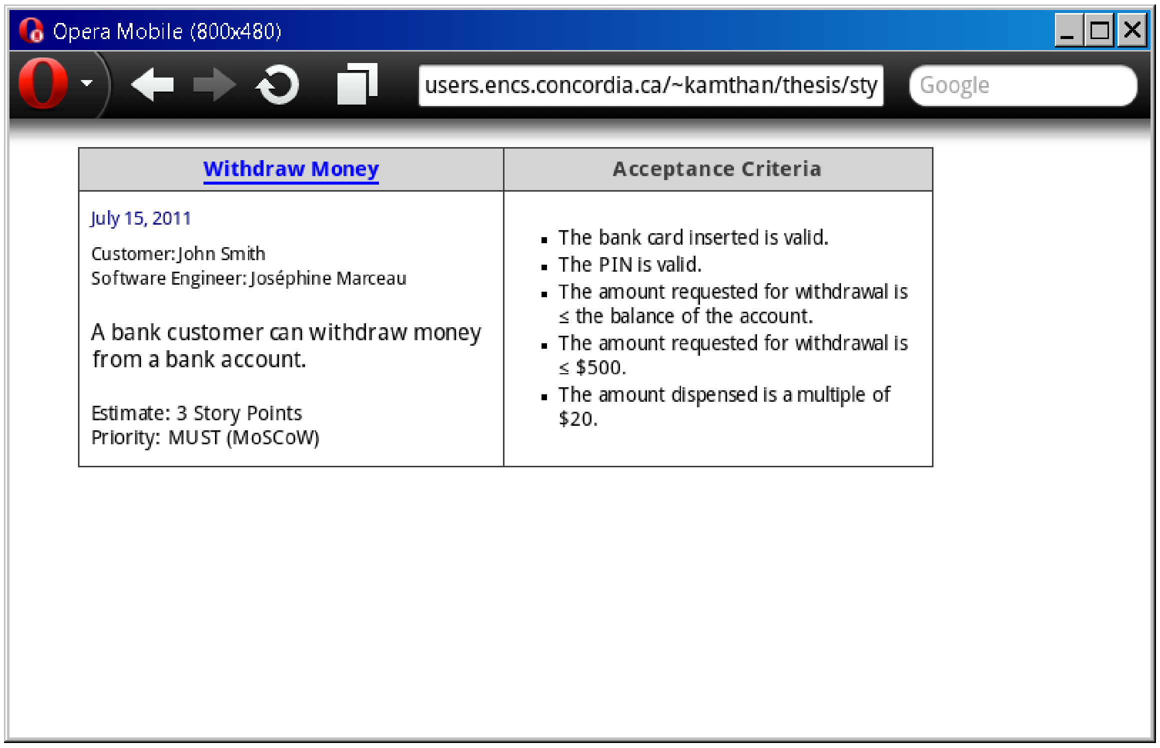

User Story Book for iBank - Mozilla Firefox
File Edit View History Bookmarks Tools Help
User Story Book for iBank

Withdraw Money	Acceptance Criteria
July 15, 2011 Customer: John Smith Software Engineer: Joséphine Marceau A bank customer can withdraw money from a bank account. Estimate: 3 Story Points Priority: MUST (MoSCoW)	▪ The bank card inserted is valid. ▪ The PIN is valid. ▪ The amount requested for withdrawal is ≤ the balance of the account. ▪ The amount requested for withdrawal is ≤ $500. ▪ The amount dispensed is a multiple of $20.

(a) A USML instance document presented as a table, mimicking the two sides of an index card, on a notebook computer.

[49] The Opera Mobile Emulator has built-in profiles for different mobile devices. These profiles can emulate the behavior of Opera user agent on a variety of mobile devices.

[50] XHTML Basic, a member of the XHTML 'Family', is aimed for devices with relatively low capabilities [McCarron, 2010].



Withdraw Money	Acceptance Criteria
July 15, 2011 Customer: John Smith Software Engineer: Joséphine Marceau A bank customer can withdraw money from a bank account. Estimate: 3 Story Points Priority: MUST (MoSCoW)	▪ The bank card inserted is valid. ▪ The PIN is valid. ▪ The amount requested for withdrawal is ≤ the balance of the account. ▪ The amount requested for withdrawal is ≤ $500. ▪ The amount dispensed is a multiple of $20.

(b) A USML instance document presented as a table, mimicking the two sides of an index card, on an emulation of a tablet computer (specifically, Viewsonic ViewPad 7).

Figure 9.13. A demonstration of a transformation of a user story representation for the sake of presentation on different devices.

Remark 9.8. There are a number of challenges inherent to delivering and presenting information on mobile devices that have been discussed elsewhere [Firtman, 2010; Kamthan, 2008].

Remark 9.9. It is possible to improve Example 9.4 in a number of ways. For example, besides the user story name, information in other elements that have a `reference` attribute, could be presented as hypertext. In certain cases, such as on certain mobile devices, the horizontal screen space can be restricted to the point that it may be unsuitable to simultaneously present both ‘front’ and ‘back’ of a user story. In such cases, the

MODULE TABS pattern [Tidwell, 2011, Page 155] could be used for presenting a USML instance document. This arrangement is illustrated in Figure 9.14.

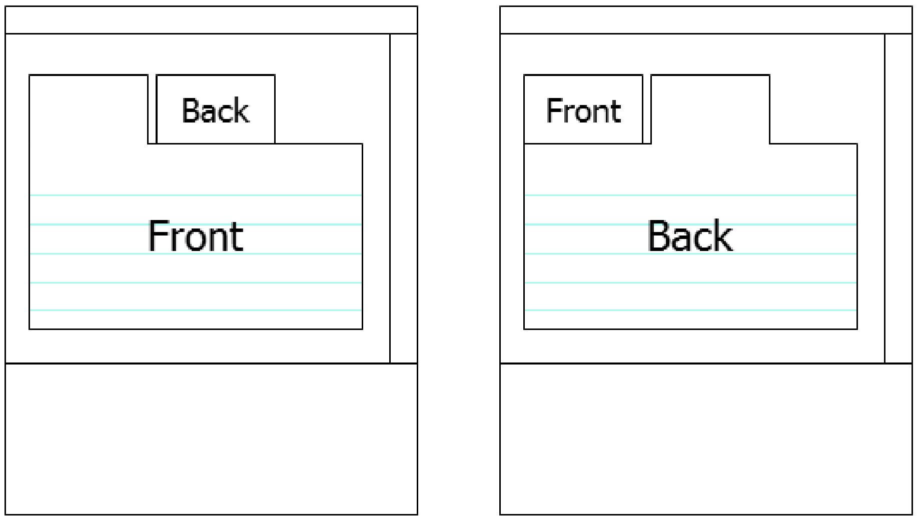


Figure 9.14. An abstract presentation of a USML instance document that uses tabs to conserve space.

Remark 9.10. Examples 9.2–9.4 make the disadvantage of separation of representation and presentation, namely the necessity to manage multiple files, apparent.

9.4.6. Limitations of USPM

The contextual information supplied by Table 9.1 does not include human or social environment factors critical for a successful presentation of a user story. In particular, it does not take into account personal characteristics of user story consumers, as they relate to successfully perceiving a user story presentation. It also does not consider personal preferences of user story consumers regarding a user story presentation.

9.5. Organizing User Stories

The organization of information is considered important for understanding. Indeed, a number of approaches for organizing information, especially pertaining to the Web, have been considered previously [Morville, Rosenfeld, 2006].

There has been previous work on organizing software requirements. For example, the idea of visualization of software requirements has been encapsulated as the VISUALIZE INFORMATION OF REQUIREMENTS TO MAKE PRIORITIZATION pattern [Välimäki, Kääriäinen, 2007]. The user stories could also be organized graphically using a Kanban Board, as suggested previously [Hiranabe, 2007].

However, there is currently no 'standard' way of organizing user stories, and there is currently no user story organization model.

There are a number of ways in which USML can assist towards organizing user stories.

9.5.1. Graphical Organization of User Stories

A user story stakeholder can benefit from a high-level, graphical view of a collection of user stories. For example, a project manager may be interested in an overview of the kind of user stories that being attended to in a specific iteration. This motivates the following definition:

Definition 9.2 [User Story Board]. An organization of a collection of user story thumbnails on a two-dimensional surface.

For example, the two-dimensional surface in the previous definition could be the primary window of a user agent that is displayed on the screen of a computing device.

Let there be a collection of user stories for a single iteration. These user stories could be represented in a single USML instance document, as per Section 9.2.2, and presented visually as a user story board, as shown in Figure 9.11(a). However, this would lead to a collection of thumbnails, one thumbnail for each user story in the USML instance document, all aligned vertically in a single column. Using XSLT along with CSS, as done in Example 8.4, it is possible for the thumbnails to make better use of the screen space.

9.5.2. Topical Organization of User Stories

A user story stakeholder may be interested in a specific aspect of a user story across a collection of user stories. For example, a project manager may be interested in all user stories that are created by a specific author, all user stories that are incomplete, or all user stories that belong to a specific iteration.

The following is a compendium of elements and attributes in USML that can serve as facets for classifying or indexing user stories:

- `author`
- `date`
- `estimate`
- `iteration`
- `metric`
- `name`
- `priority`
- `role`
- `status`
- `theme`

These facets could be used by a computer program, such as XMLStarlet[51], to query an USML instance document, and subsequently output desirable user story-related information.

9.6. Summary

USML can serve as an input to a user story development process. In USML, it is possible to have a variety of user story forms, ranging from rudimentary to elaborate. This lends USML its flexibility, and gives user story authors the ability to choose. It is also possible to present a USML instance document in different ways. This helps accommodate variations in computing environment context, and preferences of user story consumers for the type of presentation. It is also possible to use USML to organize user stories in different ways, including graphically and topically.

[51] URL: http://xmlstar.sourceforge.net/ .

Chapter 10 Extending USML

It is what it is because it was what it was.
— Grady Booch

USML, in its current state, may not be suitable for some user story stakeholders. USML takes this into account, and is designed for variability. This chapter explores and demonstrates different ways in which USML can be extended.

There are two approaches for extending USML: (1) extension at the schema-level, and (2) extension at the document-level. The extension at the document-level has been considered in Chapter 9. In this chapter, the interest is in extension at the schema-level.

Remark 10.1. The ease of extensibility comes with the challenge of scoping the language.

Remark 10.2. The provision for an extension is one of the 'Good Practices' of a specification [Dubost, Rosenthal, Hazaël-Massieux, Henderson, 2005; Jacobs, Walsh, 2004]. The EXTENSIBLE CONTENT MODEL pattern[52] provides directions for extending an XML DTD.

[52] URL: http://www.xmlpatterns.com/ .

10.1. Motivation for Extending USML

There are a number of reasons to extend the capabilities provided by USML. They can be placed into two categories:

1. **Anticipated Sources of Change.** It is possible that further granularity of a user story description, than that supported by the current version of USML, may be desirable. For example, there can be interest in different kinds of status of a user story, besides its authoring status.

2. **Unanticipated Sources of Change.** It is not possible for the specification of USML to anticipate a priori all possible needs of a user story author. For example, a different estimation approach or prioritization approach, other than those that have been provided, may be desirable by some user story authors. USML must be able to accommodate such cases.

10.2. Considerations in Extending USML

There are a number of considerations before initiating an extension of USML:

- **Legality.** If a schema is to be extended, then both its use and reuse must be legally possible. This inevitably depends on the conditions of the license of the schema. The license under which USML schema documents have been released enables extensions.

- **Backward-Compatibility.** It is important that any extension of a USML schema document aim for backward-compatibility. Let S be a USML schema document. Let D be a USML instance document that is valid with respect to S. If S' is an extension of S, then D must also be valid with respect to S'. It is evident that a restriction of S may not lead to backward-compatibility. The extensions of USML that are backward-compatible also provide an indirect rationale for the current scope of USML.

10.3. Extending the RELAX NG Schema for USML

It is possible to extend the RELAX NG schema for USML by adding the definitions of new named patterns, if necessary, followed by a redefinition of one or more existing named patterns. The following provides a generic format for such an extension:

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<include href="usml.rng"/>

<!-- The definitions of new named patterns, if any. -->
<!-- ... -->

<!-- The redefinition of one or more existing named patterns. -->
<!-- ... -->

</grammar>
```

There are six possibilities:

1. Adding an Element.
2. Deleting an Element.

3. Modifying the Content Model of an Element.
4. Adding an Attribute.
5. Deleting an Attribute.
6. Modifying the Permitted Value of an Attribute.

10.3.1. Adding an Element

The process of adding an element is illustrated by the following example.

Example 10.1. The notion of value points has been given in [Highsmith, 2009, Chapter 8]. The following (say, `usml2.rng`) shows how the `value-points` element can be added accordingly (in the vicinity of the `priority` element), while remaining backward-compatible.

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<include href="usml.rng"/>

<!-- The redefinition of usml.priority. -->
<define name="usml.priority" combine="interleave">
  <externalRef href="usml_value_points.rng"/>
</define>

</grammar>
```

The details of `usml_value_points.rng` are given in Appendix A.

The above, for instance, allows any of the following in a USML instance document that is valid with respect to `usml2.rng`:

```
<!-- The details of the priority element here. -->
<value-points range="Fibonacci Sequence">
  3
</value-points>
```

or

```
<value-points range="Fibonacci Sequence">
  5
</value-points>
<!-- The details of the priority element here. -->
```

10.3.2. Deleting an Element

The process of deleting an element is illustrated by the following example.

Example 10.2. The following shows how the `status` element can be deleted.

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<include href="usml.rng"/>

<!-- The redefinition of usml.status. -->
<define name="usml.status" combine="interleave">
  <notAllowed/>
</define>

</grammar>
```

There are certain backward-compatibility considerations in deleting an element. Let E_1 be an element that is required in the content model of an element E_2. (For example, the `estimate` element is a required child element of the `user-story` element.) Then, it is not possible to delete E_1 and assure validity of a USML instance document containing E_2.

10.3.3. Modifying the Content Model of an Element

The process of modifying the content model of an element is illustrated by the following examples.

Example 10.3. USML considers only the authoring status of a user story. It is possible to also consider review status or development status. The following (say, `usml2.rng`) shows how the `status` element can be modified, while remaining backward-compatible.

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<include href="usml.rng"/>

<!-- The redefinition of usml.status. -->
<define name="usml.status" combine="choice">
  <externalRef href="usml_status.rng"/>
</define>

</grammar>
```

The details of `usml_status.rng` are given in Appendix A.

The above, for instance, allows the following in a USML instance document that is valid with respect to `usml2.rng`:

```
<status>In Queue</status>
```

Example 10.4. The approaches for estimation currently supported by USML are subjective. USML could benefit by the addition of an approach that is objective. It has been pointed out [Rule, 2009] that, after a slight restructuring of the user story statement, the COSMIC FSM can be used to provide an estimate of a user story. The following (say, `usml2.rng`) shows how a new choice for an estimate in COSMIC Function Points can be added by modifying the `estimate` element, while remaining backward-compatible.

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<include href="usml.rng"/>

<!-- The redefinition of usml.estimate. -->
<define name="usml.estimate" combine="choice">
  <externalRef href="usml_estimate_cosmic.rng"/>
</define>

</grammar>
```

The details of `usml_estimate_cosmic.rng` are given in Appendix A.

The above, for instance, allows the following in a USML instance document that is valid with respect to `usml2.rng`:

```
<estimate approach="Data Movement" metric="COSMIC Function Points">
  6
</estimate>
```

10.3.4. Adding an Attribute

The process of adding an attribute is illustrated by the following example.

Example 10.5. The following (say, `usml2.rng`) shows how `xml:base` attribute can be added to the `usml` element and how some of the XLink attributes can be added to the `user-story` element. This addition is not backward-compatible.

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         xmlns:xlink="http://www.w3.org/1999/xlink"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<include href="usml.rng"/>

<!-- The definition of the xlink:type attribute. -->
<define name="usml.attribute.xlink-type">
  <attribute name="xlink:type">
    <value type="token">simple</value>
  </attribute>
</define>

<!-- The definition of the xlink:href attribute. -->
<define name="usml.attribute.xlink-href">
  <attribute name="xlink:href">
    <data type="anyURI"/>
  </attribute>
</define>

<!-- The definition of the xlink:title attribute. -->
<define name="usml.attribute.xlink-title">
  <attribute name="xlink:title">
    <data type="token"/>
  </attribute>
</define>

<!-- The definition of the xml:base attribute. -->
<define name="usml.attribute.xml-base">
  <attribute name="xml:base">
    <data type="anyURI"/>
  </attribute>
</define>
```

```
<!-- The redefinition of usml.user-story.attribute-list. -->
<define name="usml.user-story.attribute-list" combine="interleave">
  <ref name="usml.attribute.xlink-type"/>
  <ref name="usml.attribute.xlink-href"/>
  <ref name="usml.attribute.xlink-title"/>
</define>

<!-- The redefinition of usml.attribute-list. -->
<define name="usml.attribute-list" combine="interleave">
  <ref name="usml.attribute.xml-base"/>
</define>

</grammar>
```

The above, for instance, allows the following in a USML instance document that is valid with respect to `usml2.rng`:

```
<usml version="1.0"
      xml:lang="en"
      xml:space="preserve"
      xml:base="http://www.usml.org/"
      xmlns:xlink="http://www.w3.org/1999/xlink">
<!-- ... -->
  <user-story xml:id="RIS-US-1"
              version="0.1"
              type="Positive"
              xlink:type="simple"
              xlink:title="What's New"
              xlink:href="new.html">
  </user-story>
  <!-- ... -->
</usml>
```

10.3.5. Deleting an Attribute

The process of deleting an attribute is illustrated by the following example.

Example 10.6. The following shows how the `type` attribute in the `license` element can be deleted.

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<include href="usml.rng">
  <!-- The redefinition of usml.license.type. -->
  <define name="usml.license.type">
    <empty/>
  </define>
</include>

</grammar>
```

The backward-compatibility considerations in deleting an attribute are similar to those for deleting an element.

10.3.6. Modifying the Permitted Value of an Attribute

The process of modifying the permitted value of an attribute is illustrated by the following example.

Example 10.7. The following (say, `usml2.rng`) shows how the enumeration of the `type` attribute in the `author` element can be modified so as to classify user story authors on a new facet, while remaining backward-compatible.

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<include href="usml.rng"/>
```

```
<!-- The redefinition of usml.author.type.enumeration. -->
<define name="usml.author.type.enumeration" combine="choice">
  <group>
    <choice>
      <value type="token">Requirements Specialist</value>
      <value type="token">Domain Specialist</value>
     </choice>
  </group>
</define>

</grammar>
```

The above, for instance, allows the following in a USML instance document that is valid with respect to `usml2.rng`:

```
<author type="Requirements Specialist">
  John Smith
</author>
<author type="Domain Specialist">
  Jane Marshall
</author>
```

10.3.7. Adding an Element and an Attribute

It is possible to add both an element and an attribute, as illustrated by the following example.

Example 10.8. The `iteration` element in USML is basic and could be refined. The following (say, `usml2.rng`) shows how a `length` element, a `number` element, and a `unit` attribute, can be added, while remaining backward-compatible. The `length` element represents the length of the iteration, which can vary depending on the type of agile methodology, and the `unit` attribute represents the unit of time being used.

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<include href="usml.rng"/>

<!-- The definition of the length element. -->
<define name="usml.iteration.length.attribute-list">
  <attribute name="unit">
    <choice>
      <value type="token">Day</value>
      <value type="token">Week</value>
    </choice>
  </attribute>
</define>


<define name="usml.iteration.length.content-model">
  <data type="positiveInteger"/>
</define>

<define name="usml.iteration.length">
  <element name="length">
    <ref name="usml.iteration.length.attribute-list"/>
    <ref name="usml.iteration.length.content-model"/>
  </element>
</define>

<!-- The definition of the number element. -->
<define name="usml.iteration.number.content-model">
  <data type="positiveInteger"/>
</define>

<define name="usml.iteration.number">
  <element name="number">
    <ref name="usml.iteration.number.content-model"/>
  </element>
</define>

<!-- The redefinition of usml.iteration.content-model. -->
<define name="usml.iteration.content-model" combine="choice">
  <ref name="usml.iteration.length"/>
  <ref name="usml.iteration.number"/>
</define>

</grammar>
```

The above, for instance, allows the following in a USML instance document that is valid with respect to `usml2.rng`:

```
<iteration>
  <length unit="Week">4</length>
  <number>1</number>
</iteration>
```

Remark 10.3. It could be noted that a USML instance document based on an extension of the RELAX NG schema for USML may not be valid with respect to the Schematron Schema for USML. Furthermore, the extensions of the RELAX NG schema for USML may require the addition of new rules to the Schematron Schema for USML.

10.4. Extending the Schematron Schema for USML

The Schematron Schema for USML could be extended by adding more rules. For the sake of backward-compatibility, it is important that a new rule does not contradict any of the existing rules.

10.4.1. Relevant Information in an Element and Attribute of USML Instance Document

There are certain elements in USML that can only have text and do not have any child elements. For example, `item`, `name`, and `title` are such elements. In such cases, it is possible that the content of such an element consists of an empty string or only white space.

The RELAX NG schema for USML will check and report if there is a child element, but will not report if there is an empty string or only white space. For example,

`<title></title>` or `<title> </title>` in a USML instance document are considered valid by the RELAX NG schema for USML.

Example 10.9. The following rule asserts that the content of the `term` element must not have an empty string or only white space. The semantics of the term 'MUST' is given in [Bradner, 1997].

```
<sch:pattern name="Information in item Element">
  <sch:rule context="item">
    <sch:assert test="normalize-space()">
      The item element MUST express relevant information.
      The presence of an empty string or only white space is not
      considered relevant.
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

The same idea applies to attributes as well.

Example 10.10. The following rule asserts that the value of the `abbreviation` attribute must not have an empty string or only white space. The semantics of the term 'MUST' is given in [Bradner, 1997].

```
<sch:pattern name="Information in abbreviation Attribute">
  <sch:rule context="term">
    <sch:assert test="
      not(@abbreviation) or (normalize-space(@abbreviation))
      ">
      The abbreviation attribute, if present, MUST express relevant
      information.
      The presence of an empty string or only white space is not
      considered relevant.
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

It is possible to have similar rules for other elements and attributes.

10.4.2. Domain of User Story

A user story, as illustrated in Figure 6.5, derives knowledge from a number of domains including the application (or problem) domain.

Example 10.11. The following rule asserts that the content of the `statement` element must not have terms related to the solution domain. The semantics of the term 'SHOULD' is given in [Bradner, 1997].

```
<sch:pattern name="Problem Domain of User Story">
  <sch:rule context="statement">
    <sch:assert test="
      count(descendant::node()[contains(.,'interface')]) = 0
      ">
      The statement of a user story SHOULD include terms from the
      application (or problem) domain.
      (The terms such as 'interface' belong to the solution domain.)
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

It is possible that a term such as 'interface' may have been used in a context other than solution domain. Therefore, the report from a rule such as the above serves as a warning, rather than as an outright rejection.

10.4.3. Testability of User Story

A user story must be testable [Wake, 2002]. It is known that vagueness is a form of indeterminacy [Kamsties, Berry, Krieger, 2003], and is prohibitive to testability.

Example 10.12. The following rule asserts that the content of the `statement` element must not have terms, such as ‘easily’, that are vague. The semantics of the term ‘SHOULD’ is given in [Bradner, 1997].

```
<sch:pattern name="Determinacy of User Story">
  <sch:rule context="statement">
    <sch:assert test="
      count(descendant::node()[contains(.,'easily')]) = 0
      ">
      The statement of a user story SHOULD avoid terms that are
      indeterminate.
      (The terms such as 'easily' are vague.)
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

10.5. Summary

There are no sufficient conditions that can be imposed on USML so that it can satisfy the needs of all, present or future, user story stakeholders. Therefore, USML must be extensible by necessity, and considerations of extensibility must be forethought. In this sense, USML is extensibility-centered.

There are number of ways to extend USML, including a modification of its set of elements and attributes. The extensibility of USML empowers user story authors with the ability to personalize USML according to their own needs.

Chapter 11 Evaluating USML

I have learned that an ounce of prevention is more than worth a pound of cure.
— Andreas Zeller

This chapter explores different ways of evaluating USML. These can assist user story producers in making an informed decision whether to adopt USML.

11.1. The Advantages and Limitations of USML

There are a number of social, organizational, and technical advantages of USML:

- **Supporting Present and Future Agile Methodologies.** The current version of USML supports the state-of-the-art in user stories and agile methodologies. If future software development methodologies, including agile methodologies, adopt user stories for expressing software requirements, then USML could be used as-is or via extensions. USML, as demonstrated in Chapter 10, can be extended in a number of different ways.

- **Making Implicit Knowledge Explicit.** USML is not meant to be a substitute for implicit knowledge of software project stakeholders. (In general, writing and speaking are not substitutes for each other.) However, USML has constructs, such as the `note` element, that allow stakeholders to make their thoughts and reasoning explicit. (This is related to [P1] of Section 1.4.)

- **Learning User Stories.** USML constrains the manner in which a user story should be expressed. A violation of a constraint on physical or logical structure leads to an error upon validation. The corresponding error messages can help a person learn about certain desirable quality attributes of user stories.

- **Identifying and Organizing User Stories.** USML has constructs that can help a human or a machine to uniquely identify a user story. The user stories represented in USML can also be organized in different ways, as discussed in Section 9.5. (This is related to [P3] of Section 1.4.)

- **Publishing and Sharing User Stories.** USML enables a variety of user story forms that can benefit different user story stakeholders, as discussed in Sections 4.6 and 9.2. USML is compliant with the current electronic communication infrastructure. For example, a user story represented in USML can be used on a stationary and non-stationary device, and can be accessed on a desktop or distributed computing environment. (This is related to [P2] of Section 1.4.)

- **Navigating between Software Process Artifacts.** USML has constructs for navigating across user stories, and from user stories to other (software process) artifacts, which can be useful for traversing and tracing. (This is related to [P3] of Section 1.4.)

- **Creating 'Compound' Documents.** USML enables creation of 'compound' documents that can be validated, as illustrated by Section 9.3, and specifically by Example 9.1.

There are also a number of social, organizational, and technical limitations of USML, and they occur both by necessity and by choice:

- **Limitations Inherited from User Story Practices.** There are, as pointed out in Section 2.4, inherent limitations of user stories that limit their applicability for certain types of software projects.

- **Limitations towards Adoption.** It can not be expected that all organizations can or will be prepared to pay the price that comes with a commitment to USML. This is because of a number of reasons, including the following: the organization's requirements engineering process maturity and the expectations of USML may not be aligned; there is a learning curve inherent to any new technology, which some may not be willing to accept; there is a need to install and learn processing tools involved that some may not be willing to carry out; and that a commitment to USML may be considered as 'risky' due to the fact that it is an individual and isolated effort. In this sense, USML is not, and can not be, aimed for all.

- **Limitations towards Legacy Support.** There are certain USMS that archive user stories in 'plain' text or in proprietary formats such as Microsoft Excel. This thesis does not consider the conversion of such user stories to USML. A conversion of a user story expressed in 'plain' text to USML calls for an up-transformation that is non-trivial, especially if the original text is not structured in any known manner.

- **Limitations of Authoritativeness.** It could be suggested that, in principle, the specification of USML should follow the SPECIFICATION AS A JOINT EFFORT pattern [Rüping, 2003, Page 36]. However, being in the realm of a thesis, the specification of USML is not a collaborative effort. A standard, apart from being based on consensus, is authoritative. USML, again as being a part of a thesis, is not a standard.

- **Limitations Emanating from Commitment to Experiential Knowledge.** USML relies on experiential knowledge, including patterns. The usage of any pattern implies a commitment. This commitment, by virtue of choice, must accept both the positive and negative consequences [Buschmann, Henney, Schmidt, 2007] of using that pattern. USML is based on a number of patterns. Therefore, it inherits the limitations of applying the solutions of those patterns, unless steps are taken to address those limitations. (For example, Section 6.2.10 suggests the use of the MULTIPART

FILES[53] pattern. However, as a consequence, these multiple files need to be managed.)

- **Limitations Emanating from Commitment to Technologies.** USML, by virtue of commitment, inherits the limitations of the technologies it depends upon. USML directly or indirectly depends on a number of information technologies for its definition. For example, USML depends directly on XML and indirectly on Unicode. It is evident that limitations of these other specifications are inherited by USML. (The limitations of XML are considered in Section 6.3.2.) Furthermore, changes in these ancillary specifications can impact USML. The origins of these changes could be discovery of errors, or deprecation, obsolescence, or replacement of certain functionality in a specification. For example, the specifications of both XML and Unicode have evolved, and continue to evolve, since their inception. As history has shown, information technologies, in general, are not time-invariant.

- **Limitations of Representation.** USML is independent of any application (problem) domain. As a result, a USML instance document does not represent, and therefore constrain, knowledge of any particular application domain. USML depends on schema languages for representation, and schema languages (unlike ontology languages) are limited in their capabilities towards describing relationships.

[53] URL: http://www.xmlpatterns.com/ .

11.2. A Comparison of Previous Work for Representing User Stories and USML

In Section 1.5, previous work for representing user stories was outlined. In this section, previous work and USML are compared, as shown in Table 11.1.

The criteria for comparison are based on [Jacobs, Walsh, 2004], and are considered equally significant. The presence of 'Y' indicates that there is support for the criterion in question; a 'N' indicates otherwise.

Criterion	[PW1]	[PW2]	[PW3]	USML
Extendability	N	N	N	Y (Chapter 10)
Identifiability	Y	Y	Y	Y (Section 8.2.2)
Navigability	N	N	Y	Y (Section 8.2.2)
Validatibility	Y	N	Y	Y (Section 7.4)
Versionability	Y	Y	Y	Y (Section 8.2.2)

Table 11.1. A comparison of previous work for representing user stories and USML.

It could be noted that [PW4] and [PW5] are not included in the comparison as their specifications are publicly unavailable.

11.3. The Design of an Experiment for USML

There are different types of empirical investigations in software engineering [Wohlin, Runeson, Höst, Ohlsson, Regnell, Wesslén, 2000, Section 2.1]: experiment, case study, and survey.

An experiment is "research in the small" [Fenton, Bieman, 2012, Chapter 4], and is defined as a "rigorous, controlled investigation of an activity […] that involves the "testing of hypotheses concerning postulated effects of independent variables on dependent variables in a setting that minimizes other factors that might affect the outcome". Figure 11.1, an adaptation of [Wohlin, Runeson, Höst, Ohlsson, Regnell, Wesslén, 2000, Figure 9], illustrates an abstract view of an experiment.

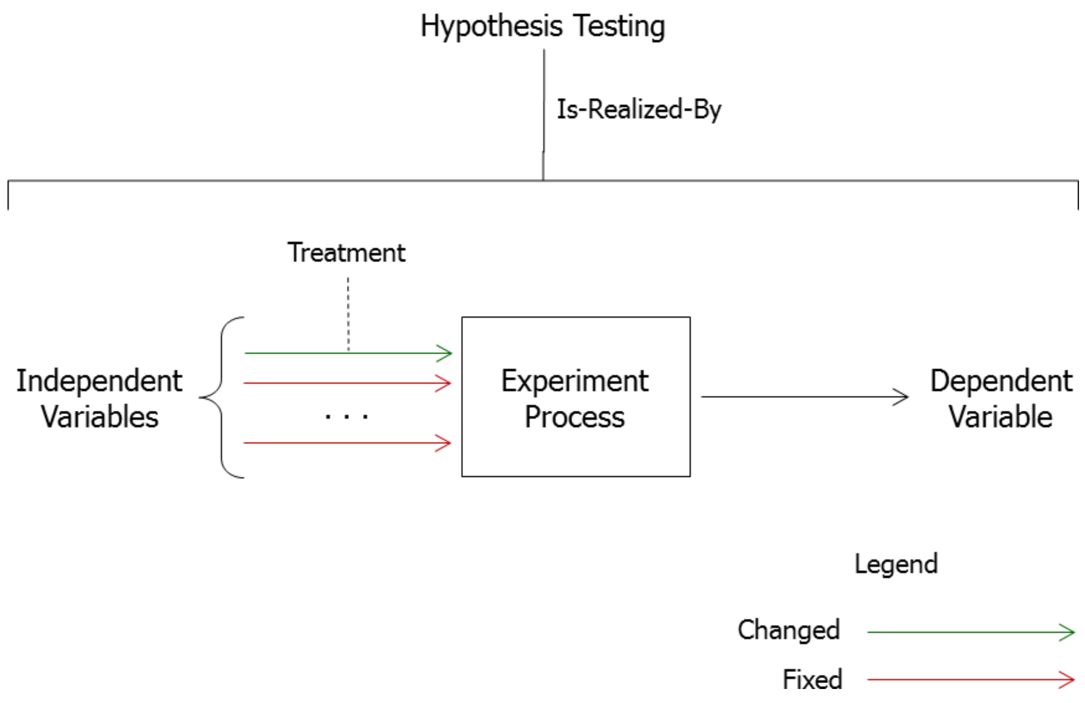


Figure 11.1. A high-level view of an experiment.

From the given types of empirical investigations in software engineering, experiment is adopted in this thesis due to its suitability in an academic environment.

There is increasing, diverse literature on experiments in software engineering [Fenton, Bieman, 2012, Chapter 4; Juristo, Moreno, 2001; Wohlin, Runeson, Höst, Ohlsson, Regnell, Wesslén, 2000]. For the sake of consistency, the terminology adopted here is that of [Fenton, Bieman, 2012, Chapter 4].

The purpose of an experiment process is to ensure that there is support for setting up and conducting the experiment so that it is successful. In [Fenton, Bieman, 2012, Chapter 4], an experiment process is given, and it has the following phases: Conception, Design, Preparation, Execution, Analysis, and Dissemination and Decision-Making. Figure 11.2 illustrates this experiment process. The dashed arrows imply that the process is iterative.

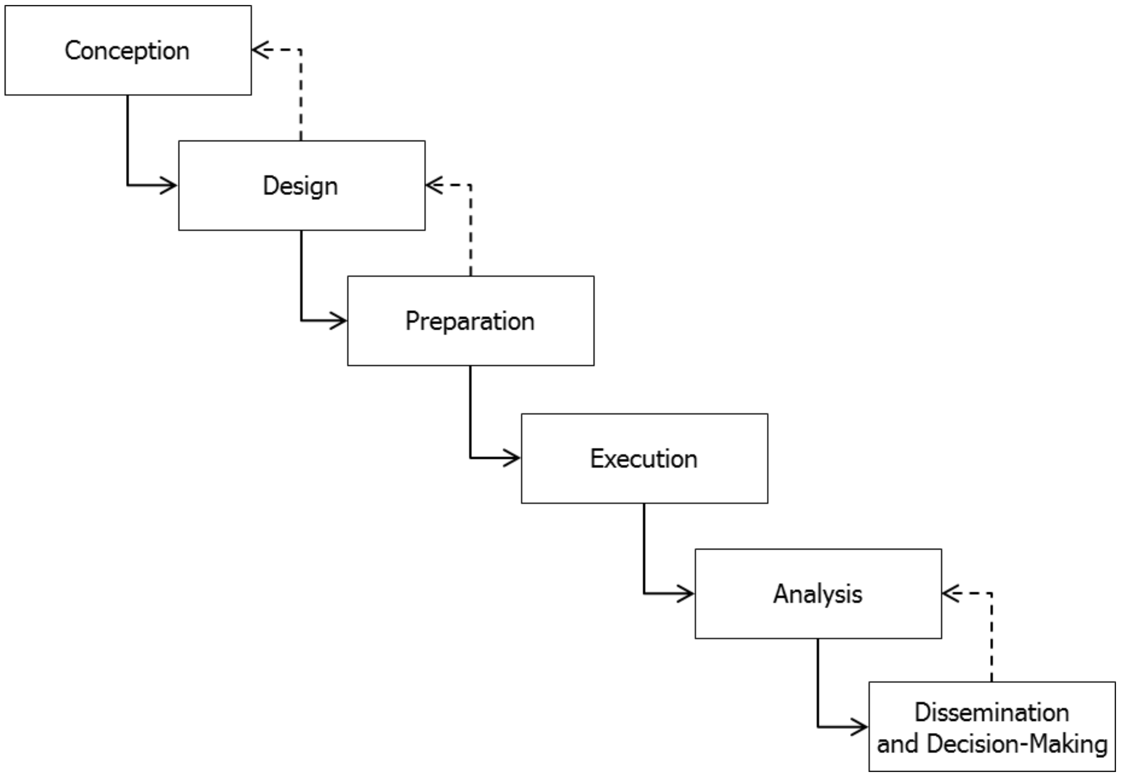


Figure 11.2. A partially iterative experiment process.

There are a number of possible experiments in relation to this thesis. In the following, one experiment is considered, and only the first three phases, namely Conception, Design, and Preparation of its process are described.

- **Conception.** In this phase, the goal of the experiment is stated in a manner that can be evaluated.
- **Design.** In this phase, the goal of the experiment is operationalized. To do that, the hypothesis, variables, objects, subjects, and trials are given, and any threats to the validity of the experiment are pointed out.
- **Preparation.** In this phase, logistical, technical, or social elements required for the experiment to successfully commence and continue are stated.

11.3.1. Phase 1: Conception

There is need for further elaboration on expressivity before the goal of the experiment is stated.

Expressivity of a Representation of a User Story

The expressivity of a representation of a user story is related to making certain information related to a user story explicit, that is, inclusion in the representation of certain elements or attributes of a user story. (The expressivity of a representation based on XML depends on the granularity of markup, as discussed in Section 7.2.4.) This leads to different categories of expressivity on an ordinal scale, as illustrated in Table 11.2.

Level of Expressivity	**Elements/Attributes in a Representation of a User Story**
E_0	$\neg$ (Role $\wedge$ Goal $\wedge$ Value)
E_1	Role $\wedge$ Goal $\wedge$ Value
E_2	Role $\wedge$ Goal $\wedge$ Value $\wedge$ Acceptance Criteria
E_3	Role $\wedge$ Goal $\wedge$ Value $\wedge$ Acceptance Criteria $\wedge$ (Estimate $\vee$ Priority)
E_4	Role $\wedge$ Goal $\wedge$ Value $\wedge$ Acceptance Criteria $\wedge$ Estimate $\wedge$ Priority
E_5	Role $\wedge$ Goal $\wedge$ Value $\wedge$ Acceptance Criteria $\wedge$ Estimate $\wedge$ Priority $\wedge$ Other

Table 11.2. A classification of expressivity of a representation of a user story.

Remark 11.1. The scheme in Table 11.2 can be explained as follows. E_0, …, E_5 reflect increasing order of expressivity, E_0 being 'lowest' and E_5 being 'highest'. In level E_0, at least one of Role, Goal, or Value is absent. A representation of a user story at this level is missing basic user story information units for a user story statement, and therefore can not be considered as 'highly' expressive. In levels E_1 and E_2, basic user story information units are present, so they are more expressive than E_0, but there is no support for planning or meta-information. The arguments for rest of the levels can be made similarly.

The index i of a level of expressivity E_i in Table 11.2 is called expressivity index. It can be noted that expressivity index can take one of the values in {0, …, 5}. For example, expressivity index of E_3 is 3.

Proposition 11.1. Let US be a user story. Let R_1 and R_2 be representations in languages L_1 and L_2, respectively, of US. Let the expressivity of R_1 and R_2 be $E_i(US, R_1)$ and $E_j(US, R_2)$, respectively, given that $E_i(US, R_1)$ and $E_j(US, R_2)$ belong to $\{E_0, \ldots, E_5\}$ as per Table 11.2. If $i > j$, that is, if the expressivity index of $E_i(US, R_1)$ is greater than the expressivity index of $E_j(US, R_2)$, then R_1 is more expressive than R_2.

Proposition 11.2. Let $\{US_1, \ldots, US_n\}$ be a user story book. Let $R_1(US_k)$ and $R_2(US_k)$ be representations in languages L_1 and L_2, respectively, of the user stories US_k, $k = 1, \ldots, n$. Let the expressivity of $R_1(US_k)$ and $R_2(US_k)$ be $E_i(US_k, R_1)$ and $E_j(US_k, R_2)$, respectively, given that $E_i(US_k, R_1)$ and $E_j(US_k, R_2)$ belong to $\{E_0, \ldots, E_5\}$ as per Table 11.2. If median of expressivity indices of $\{E_i(US_1, R_1), \ldots, E_i(US_n, R_1)\}$ > median of expressivity indices of $\{E_j(US_1, R_2), \ldots, E_j(US_n, R_2)\}$, then it can be concluded in general that a representation of a user story in L_1 is more expressive than in L_2.

Let L be any one of the current means for representing user stories, namely [PW1] – [PW3].

Goal: To assess if, as given by Proposition 11.2, a representation of a user story is more expressive in USML than in L, for any L.

11.3.2. Phase 2: Design

The experiment design type is single-factor, multiple-treatment, unrelated between-subjects design.

Hypothesis

The null hypothesis is:

H_0: A representation of a user story based on USML is not more expressive than a representation based on L, for any L.

The alternative hypothesis is:

H_1: A representation of a user story based on USML is more expressive than a representation based on L, for every L.

Independent Variables

The independent variables are Markup Language, Knowledge of Application (or Problem) Domain, Knowledge of User Stories, Skill in XML, Number of Well-Formedness Errors, Number of Validation Errors, and Time Allotted for Completion of a Trial.

Factor

The factor is Markup Language.

Treatments

The treatments of Markup Language are USML and L. In other words, the experiment has four treatments.

Dependent Variable

The dependent variable is Expressivity. It can take six values {E0, …, E5}, as per Table 11.2.

Experimental Object

The experimental object is User Story Representation.

Experimental Subjects

The experimental subjects are either Undergraduate Students or Graduate Students (but not both) from a software engineering program.

In empirical software engineering studies, the privacy of the experimental subjects is a concern. To address the issue of privacy, the experimental subjects must be guaranteed anonymity. To achieve that, the experiment must be conducted in a double-blind manner.

There must be a signed statement of commitment and informed consent from each experimental subject.

The following is the list of characteristics expected of experimental subjects:

- **Software Engineering.** The experimental subjects have basic knowledge of authoring software requirements, preferably in user stories; of software estimation, preferably in agile estimation; and of software testing, specifically, acceptance testing.
- **XML.** The experimental subjects have basic knowledge of authoring XML instance documents.

Trials

The trials of the experiment are based on the following partial description of a software project:

A human resources information system is to be developed. The potential users of the system include job seekers, such as John Smith, looking for employment opportunities. The system is to include employment information. The system is to allow job seekers to upload their résumés. The system must prevent upload of undesirable files, such as malware, by malicious users.

Trial 1

Task A: A positive user story must be authored based on the given software project description. The representation of the user story must include a user story statement, an estimate in story points, the priority using the MoSCoW scheme, and an acceptance test. The representation may include a user story name.

Task B: The representation of the user story must be validated.

Trial 2

Task A: A negative user story must be authored based on the given software project description. The representation of the user story must include a user story statement, an estimate in ideal days, the priority using the MoSCoW scheme, and an acceptance test. The representation may include a user story name.

Task B: The representation of the user story must be validated.

Experimental Error and Validity

The following steps must be taken to minimize experimental error, and to maximize the validity of the experiment:

- **Replication.** It is expected that the experimental subjects are new to USML. However, to avoid confounding of variables and to support internal validity, the experimental subjects must also be unaware of X.

- **Randomization.** To support internal validity, conclusion validity, and external validity, the selection of the experimental subjects and the assignment of experimental subjects to a trial must be random. For example, such randomization can be achieved via simple random sampling. To avoid preconception, the experimental subjects must also not be informed of the goal of the experiment.

- **Local Control.** To support local control, there must be different kinds of balancing. The number of experimental subjects assigned to each trial must be identical. The day and time allotted to each experimental subject for completing a trial must be identical. Furthermore, a trial must be completed within a single day and within the same time slot. There must be consistency in the use of tools across a trial. In particular, the experimental subjects assigned to a specific treatment must use the same set of tools.

11.3.3. Phase 3: Preparation

The experiment requires preparation in a number of directions before its execution.

Room

A computer laboratory with controlled access is the recommended place for executing the experiment.

Equipment

There is need for a computer per experimental subject. The computer must have tools for authoring and validating XML instance documents, in general, and USML instance documents, in particular, installed on it prior to the execution of the experiment. The list of tools in Appendix E is sufficient.

Training

The experimental subjects need a basic introduction to USML, X, and the tools for authoring and validating XML instance documents being deployed in the experiment. It is crucial that the time and depth of coverage of training is uniform across the experimental subjects. To achieve that, all experimental subjects must be trained simultaneously.

Remark 11.2. It is expected that $\mathbf{H_0}$ will be rejected, and $\mathbf{H_1}$ will be accepted, especially for Trial 2.

Remark 11.3. The instance documents authored by the experimental subjects can be checked, manually or automatically, for the elements and attributes they contain, and matched against Table 11.2 to conclude the expressivity of each representation.

Remark 11.4. There can be other experiments, such as to assess the availability of a representation of a user story over the Web via a mobile device, and to assess the readability of a representation of a user story by a visually-abled person with no knowledge of XML.

11.4. Summary

USML needs to be evaluated, both absolutely and relatively. There are both theoretical advantages and practical limitations of USML. In a relative assessment, USML is unique among the possible choices. An experiment can help evaluate USML in a real-world setting.

Chapter 12 Conclusion

The more I learn, the more I realize I don't know.
— Albert Einstein

The attention on both user and usage is critical to many software systems currently being developed, especially those that are interactive. It has also been predicted [Boehm, 2011] that attention towards users and their needs will continue to be important determinant, and will shape all aspects of software development, including business, organizational, and technical aspects.

The user stories provide one possibility for expressing the requirements for interactive software systems. In doing so, user stories could also serve other purposes. For example, user stories could act as a much desirable 'bridge' between software engineering and human-computer interaction, and thereby help reduce the classical 'chasm' between these disciplines [Beyer, 2010; Jokela, Abrahamsson, 2004]. The user stories could also serve as a useful entry point to requirements engineering pedagogy of interactive software systems, as the author's experience in teaching undergraduate and graduate courses related to software engineering over the years has shown.

This thesis is created on the premise that any means for describing user stories must be amenable to both humans and machines. This means must also be minimally constrained in the sense that it must be independent, as far as possible, of any specific software development methodology, hardware, and software, and it must be open. To do that, an understanding of the environment in which user stories are described is a requisite, reliance on experiential knowledge is constructive, and prudent selection of technologies is an imperative. These aspects manifest themselves in many ways throughout the thesis, including reliance on conceptual modeling (for theory) and the notion of descriptive markup (for practice).

There are a number of advantages for user story stakeholders of expressing user stories in the electronic medium. To realize that, an avenue that has certain characteristics including the following, is desirable: it rests on a theoretical foundation; it is informed by practical knowledge in form of conventions, guidelines, patterns, principles, recommended practices, and standards; it is put into perspective by highlighting trade-offs; it offers user story stakeholders multiple options for representation; and it can accommodate variability. USML can serve as one such avenue.

12.1. A Summary of Contributions

The contributions of this thesis can be summarized as follows:

- A set of conceptual models that form a theoretical foundation for creating or supporting markup languages like USML (covered in Chapters 3–6, and in Section 9.4).
- A markup language, namely USML, for representing user stories (covered in Chapters 7, 8, and 10, and in Appendices A and B).

- A set of applications of USML (covered in Chapter 9, and in Appendices C and D).

12.2. Directions for Future Research

There are a number of, not necessarily mutually exclusive, directions for future research that emanate from this thesis. These are briefly outlined in the following sections.

12.2.1. User Story Quality

There is currently no user story quality model, although certain approaches for the quality of user stories have been proposed [Alexander, Maiden, 2004, Page 271; Jeffries, 2001; Patel, Ramachandran, 2009a; Pham, Pham, 2012, Chapter 4; Wake, 2002]. The quality attributes included in these approaches have overlaps. For example, testability of a user story is one of the common quality attributes.

However, these approaches have one or more of the following limitations: theoretical foundation of the quality attributes is not given; lack of sufficiency of the quality attributes as a collective has not been pointed out; previous efforts on software requirements quality modeling are not mentioned; the scope, say, applicability to single user story or multiple user stories, is not considered; and views of user story description, say, information and representation, are not separated.

It would be useful to have multiple quality models [Dromey, 1996], each for a different view [Garvin, 1984] of user story description, say, information, representation, and presentation, as per Figure 3.1. The interest, as it pertains to this thesis, is primarily in a quality model for user story information and a quality model for user story representation. In each case, a quality attribute could be determined by its relevancy to some user story stakeholder, as per USSM. In doing so, the scope of a quality attribute is significant.

For example, pronounceability, uniguity [Kiyavitskaya, Zeni, Mich, Berry, 2008; Kamsties, Berry, Krieger, 2003; Winbladh, Ziv, Richardson, 2008], readability, and (internal) consistency are quality attributes relevant to a user story consumer. The scope of pronounceability is certain user story information unit, such as, name (of user story or that of one of the tests), scope of uniguity is the user story statement, the scope of readability is the user story description, and the scope of (internal) consistency is multiple user story descriptions.

12.2.2. Representation of User Story-Related Artifacts

It is evident from previous work [Kamthan, Shahmir, 2010] and from earlier chapters, in particular Chapter 5, that there are a number of artifacts related to a user story. Indeed, the existence of such artifacts forms, in part, the motivation for the `reference` attribute on a number of USML elements, as specified in Chapter 8.

The focus in this thesis has been on the representation of a user story. It is also of interest to consider the representations of at least some of these other artifacts, in case suitable means of representation do not exist. A realization of this allows the construction of a 'network' of interrelated and co-located/distributed representations.

For example, personal details of a user story author could be represented using vCard[54], and a software project glossary could be represented using DocBook XML[55]. However, for other artifacts such possibilities currently do not exist. In particular, an investigation into a suitable representation of a persona is of interest.

12.2.3. User Story Management and Implications of the Social Web

There is currently no user story management model, although certain approaches for managing user stories have been proposed [Breitman, Leite, 2002; Lewitz, 2004; Paetsch, 2003]. The activity of managing user stories using a "database system" is included in the SMM Level 3, Key Process Area 3.6 [Patel, Ramachandran, 2009b]. However, these approaches do not consider the management of user stories in agile software development with geographically dispersed teams.

The Social Web, or as it is more commonly referred to by the pseudonym Web 2.0 [O'Reilly, 2005], is the perceived evolution of the Web in a direction that is driven by 'collective intelligence', realized by information technology, and characterized by user participation, openness, and network effects. The Social Web has the potential for supporting distributed requirements development and management, as indicated by the use of Wiki systems [Decker, Ras, Rech, Jaubert, Rieth, 2007; Louridas, 2006; Minocha, Petre, Roberts, 2008]. The implications of the Social Web for software engineering

[54] URL: http://www.vcarddav.org/ .
[55] URL: http://www.docbook.org/ .

education have been addressed in [Kamthan, 2009; Kamthan, 2011c; Kamthan, 2011d], and for developing and disseminating user stories have been considered in [Fancott, Kamthan, Shahmir, 2011; Kamthan, 2011d]. For the new generation of user story stakeholders, the prospects of the Social Web are likely to be particularly appealing, as suggested by Figure 12.1.

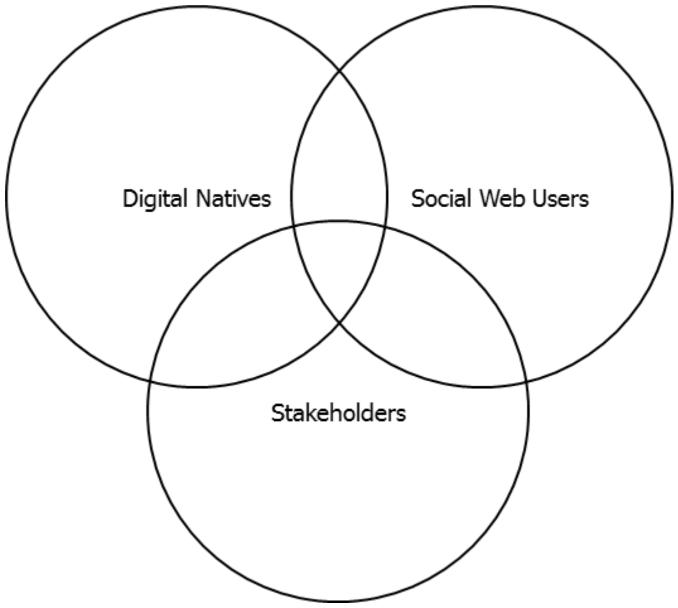


Figure 12.1. The new generation of user story stakeholders is likely to consist of digital natives [Palfrey, Gasser, 2008] who are avid users of the Social Web.

It would be useful to examine the potential of USML as a language for server-side representation of user stories in a USMS characterized as a Social Web Application. For example, such a USMS could be a topic for a team-based project in a course on requirements engineering. This would also lend an opportunity to demonstrate symbiosis in the relationship between user stories and the Social Web [Fancott, Kamthan, Shahmir, 2011].

Appendix A The RELAX NG Schema for USML

This appendix includes the listing of the RELAX NG Schema for USML, which is relevant to Chapters 7 and 8, and the listing of that of related files, which are relevant to Chapter 10. A means for annotating the RELAX NG Schema for USML is also given.

A.1. Conventions for Definitions of Elements and Attributes

The convention used to define a USML attribute is:

```
<define name="usml.*">
  <attribute name="*">
    <!-- ... -->
  </attribute>
</define>
```

where a "*" denotes the name of some USML attribute.

The convention used to define a USML element is:

```
<define name="usml.*.attribute-list">
  <!-- ... -->
</define>

<define name="usml.*.content-model">
  <!-- ... -->
</define>

<define name="usml.*">
  <element name="*">
    <ref name="usml.*.attribute-list"/>
    <ref name="usml.*.content-model"/>
  </element>
</define>
```

where a “*” denotes the name of some USML element.

A.2. Files Related to USML

In the following, the organization of the RELAX NG Schema for USML is explained, and corresponding files are listed (that are relevant to Chapters 7 and 8).

Organization of the RELAX NG Schema for USML

There are seven files related to the RELAX NG Schema for USML, of which six are referred to from the main file, namely `usml.rng`.

The order of definitions of elements and attributes in the RELAX NG Schema for USML is based on the DECLARE BEFORE FIRST USE[56] pattern. If the pattern is not applicable, then the order of definitions of elements and attributes is alphabetical.

The list of definitions of attributes common to certain elements is a manifestation of the COMMON ATTRIBUTES[57] pattern.

The list of definitions of child elements that are common to certain elements is a manifestation of the CONSISTENT ELEMENT SET [58] pattern.

[56] URL: http://www.xmlpatterns.com/ .
[57] URL: http://www.xmlpatterns.com/ .
[58] URL: http://www.xmlpatterns.com/ .

The Main RELAX NG Schema for USML File (`usml.rng`)

```
<?xml version="1.0" encoding="UTF-8"?>

<!--

Filename: usml.rng
Author: Pankaj Kamthan
Date: July 15, 2011
Version: 1.0
Description: This is the RELAX NG schema for the User Story Markup
             Language (USML) 1.0.
License: Creative Commons Attribution-Noncommercial-No Derivative Works
         3.0 Unported

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [IETF] RFC 2119.

-->

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<!-- DEFINITIONS OF ATTRIBUTES -->

<!-- DEFINITIONS OF COMMON ATTRIBUTES -->

<!-- The definition of the (hypertext) reference attribute. -->
<define name="usml.reference">
  <attribute name="reference">
    <data type="anyURI"/>
  </attribute>
</define>

<!-- The definition of the version attribute. -->
<define name="usml.version">
  <attribute name="version">
    <data type="decimal">
      <!-- The regular expression for versioning. -->
      <param name="pattern">\p{Nd}+(\.\p{Nd}+)?</param>
    </data>
  </attribute>
</define>

<!-- The definition of the xml:id attribute. -->
<define name="usml.xml-id">
  <attribute name="xml:id">
    <data type="ID"/>
  </attribute>
</define>
```

```
<!-- DEFINITIONS OF OTHER ATTRIBUTES -->

<!-- The definition of the abbreviation attribute. -->
<define name="usml.term.abbreviation">
  <attribute name="abbreviation">
    <data type="token"/>
  </attribute>
</define>

<!-- The definition of the expansion attribute. -->
<define name="usml.term.expansion">
  <attribute name="expansion">
    <data type="token"/>
  </attribute>
</define>

<!-- The definition of the (maturity) model attribute. -->
<define name="usml.maturity.model">
  <attribute name="model">
    <data type="token"/>
  </attribute>
</define>

<!-- The definition of the persona attribute. -->
<!-- The persona MUST correspond to the role. -->
<define name="usml.role.persona">
  <attribute name="persona">
    <data type="token"/>
  </attribute>
</define>

<!-- The definition of the user story author type attribute. -->

<!-- The choices of user story authors on different facets. -->
<define name="usml.author.type.enumeration">
  <choice>
    <group>
      <choice>
        <value type="token">Internal</value>
        <value type="token">External</value>
      </choice>
    </group>
    <group>
      <choice>
        <value type="token">Software Engineer</value>
        <value type="token">Customer</value>
        <value type="token">User</value>
      </choice>
    </group>
  </choice>
</define>

<define name="usml.author.type">
  <attribute name="type">
    <ref name="usml.author.type.enumeration"/>
  </attribute>
</define>
```

```
<!-- The definition of the constraint type attribute. -->
<define name="usml.constraint.type">
  <attribute name="type">
    <data type="token"/>
  </attribute>
</define>

<!-- The definition of the license type attribute. -->
<define name="usml.license.type">
  <attribute name="type">
    <data type="token"/>
  </attribute>
</define>

<!-- The definition of the methodology type attribute. -->
<define name="usml.methodology.type">
  <attribute name="type">
    <choice>
      <value type="token">Agile</value>
      <value type="token"/>
    </choice>
  </attribute>
</define>

<!-- The definition of the role type attribute. -->
<define name="usml.role.type">
  <attribute name="type">
    <choice>
      <value type="token">Positive</value>
      <value type="token">Negative</value>
    </choice>
  </attribute>
</define>

<!-- The definition of the user story type attribute. -->
<define name="usml.user-story.type">
  <attribute name="type">
    <choice>
      <value type="token">Positive</value>
      <value type="token">Negative</value>
    </choice>
  </attribute>
</define>
```

```
<!-- The definition of the user story relationship type attribute. -->
<define name="usml.user-story.relationship.type">
  <attribute name="type">
    <list>
      <oneOrMore>
        <choice>
          <!-- The given choices for the relationship type. -->
          <value type="token">Similar-To</value>
          <value type="token">Resource-Contender-Of</value>
          <value type="token">Consequence-Of</value>
          <value type="token">Required-By</value>
          <value type="token">Occurs-Before</value>
          <value type="token">Occurs-Concurrently-With</value>
          <value type="token">Occurs-After</value>
          <!-- The relationship-type not among the given choices. -->
          <data type="token"/>
        </choice>
      </oneOrMore>
    </list>
  </attribute>
</define>

<!-- The definition of the xml:lang attribute. -->
<define name="usml.xml-lang">
  <attribute name="xml:lang">
    <data type="language"/>
  </attribute>
</define>

<!-- The definition of the xml:space attribute. -->
<define name="usml.xml-space">
  <attribute name="xml:space">
    <value>preserve</value>
  </attribute>
</define>

<!-- DEFINITIONS OF ELEMENTS -->

<!-- DEFINITIONS OF COMMON ELEMENTS -->

<!-- The definition of the item element. -->
<define name="usml.item.content-model">
  <text/>
</define>

<define name="usml.item">
  <element name="item">
    <ref name="usml.item.content-model"/>
  </element>
</define>

<!-- The definition of the name element. -->
<define name="usml.name.attribute-list">
  <optional>
    <ref name="usml.reference"/>
  </optional>
</define>
```

```
<define name="usml.name.content-model">
  <data type="token"/>
</define>

<define name="usml.name">
  <element name="name">
    <ref name="usml.name.attribute-list"/>
    <ref name="usml.name.content-model"/>
  </element>
</define>


<!-- The definition of the term element. -->
<define name="usml.term.attribute-list">
  <optional>
    <ref name="usml.reference"/>
  </optional>
  <optional>
    <choice>
      <ref name="usml.term.abbreviation"/>
      <ref name="usml.term.expansion"/>
    </choice>
  </optional>
</define>

<define name="usml.term.content-model">
  <data type="token"/>
</define>

<define name="usml.term">
  <element name="term">
    <ref name="usml.term.attribute-list"/>
    <ref name="usml.term.content-model"/>
  </element>
</define>

<!-- DEFINITIONS OF OTHER ELEMENTS -->

<div>

<!-- DEFINITIONS OF ELEMENTS FOR USER STORY BOOK INFORMATION -->

<!-- The definition of the user story book title element. -->
<define name="usml.title.content-model">
  <data type="token"/>
</define>

<define name="usml.title">
  <element name="title">
    <ref name="usml.title.content-model"/>
  </element>
</define>
```

```
<!-- The definition of the organization element. -->
<define name="usml.organization.content-model">
  <data type="token"/>
</define>

<define name="usml.organization">
  <element name="organization">
    <ref name="usml.organization.content-model"/>
  </element>
</define>

<!-- The definition of the (software) project element. -->
<define name="usml.project.content-model">
  <data type="token"/>
</define>

<define name="usml.project">
  <element name="project">
    <ref name="usml.project.content-model"/>
  </element>
</define>

<!-- The definition of the (software) license element. -->
<define name="usml.license.attribute-list">
  <optional>
    <ref name="usml.reference"/>
  </optional>
  <optional>
    <ref name="usml.license.type"/>
  </optional>
</define>

<define name="usml.license.content-model">
  <data type="token"/>
</define>

<define name="usml.license">
  <element name="license">
    <ref name="usml.license.attribute-list"/>
    <ref name="usml.license.content-model"/>
  </element>
</define>

<!--
     The definition of the (software development) methodology element.
-->
<define name="usml.methodology.attribute-list">
  <optional>
    <ref name="usml.reference"/>
  </optional>
  <optional>
    <ref name="usml.methodology.type"/>
  </optional>
</define>
```

```
<define name="usml.methodology.content-model">
  <choice>
    <!-- The choices for the (agile) methodology. -->
    <value type="token">Behaviour-Driven Development</value>
    <value type="token">Extreme Programming</value>
    <value type="token">Scrum</value>
    <value type="token">User-Centered Agile Process</value>
    <!-- The methodology not among the given choices. -->
    <data type="token"/>
  </choice>
</define>

<define name="usml.methodology">
  <element name="methodology">
    <ref name="usml.methodology.attribute-list"/>
    <ref name="usml.methodology.content-model"/>
  </element>
</define>

<!-- The definition of the (problem) domain element. -->
<define name="usml.domain.content-model">
  <data type="token"/>
</define>

<define name="usml.domain">
  <element name="domain">
    <ref name="usml.domain.content-model"/>
  </element>
</define>

<!-- The definition of the user story book maturity element. -->
<define name="usml.maturity.attribute-list">
  <ref name="usml.maturity.model"/>
</define>

<define name="usml.maturity.content-model">
  <data type="token"/>
</define>

<define name="usml.maturity">
  <element name="maturity">
    <ref name="usml.maturity.attribute-list"/>
    <ref name="usml.maturity.content-model"/>
  </element>
</define>

</div>
```

```
<div>

<!-- DEFINITIONS OF ELEMENTS FOR USER STORY INFORMATION -->

<!-- The definition of the date element. -->
<define name="usml.date.content-model">
  <choice>
    <data type="date"/>
    <data type="token"/>
  </choice>
</define>

<define name="usml.date">
  <element name="date">
    <ref name="usml.date.content-model"/>
  </element>
</define>

<!-- The definition of the author element. -->
<define name="usml.author.attribute-list">
  <optional>
    <ref name="usml.reference"/>
  </optional>
  <ref name="usml.author.type"/>
</define>

<define name="usml.author.content-model">
  <data type="token"/>
</define>

<define name="usml.author">
  <element name="author">
    <ref name="usml.author.attribute-list"/>
    <ref name="usml.author.content-model"/>
  </element>
</define>

<!--
     The definition of the (user story) authoring status element
     as per the User Story Process Model.
-->
<define name="usml.status.content-model">
  <choice>
    <value type="token">Incomplete</value>
    <value type="token">Complete</value>
  </choice>
</define>

<define name="usml.status">
  <element name="status">
    <ref name="usml.status.content-model"/>
  </element>
</define>
```

```
<!-- The definition of the iteration element. -->
<define name="usml.iteration.content-model">
  <data type="positiveInteger"/>
</define>

<define name="usml.iteration">
  <element name="iteration">
    <ref name="usml.iteration.content-model"/>
  </element>
</define>

<!-- The definition of the (user story) theme element. -->
<define name="usml.theme.content-model">
  <data type="token"/>
</define>

<define name="usml.theme">
  <element name="theme">
    <ref name="usml.theme.content-model"/>
  </element>
</define>

<!-- The definition of the context-of-use element. -->
<define name="usml.context-of-use.attribute-list">
  <optional>
    <ref name="usml.reference"/>
  </optional>
</define>

<define name="usml.context-of-use.content-model">
  <interleave>
    <text/>
    <zeroOrMore>
      <ref name="usml.item"/>
    </zeroOrMore>
  </interleave>
</define>

<define name="usml.context-of-use">
  <element name="context-of-use">
    <ref name="usml.context-of-use.attribute-list"/>
    <ref name="usml.context-of-use.content-model"/>
  </element>
</define>
```

```
<!-- The definition of the role element. -->
<define name="usml.role.attribute-list">
  <optional>
    <ref name="usml.xml-id"/>
  </optional>
  <optional>
    <ref name="usml.reference"/>
  </optional>
  <optional>
    <ref name="usml.role.type"/>
  </optional>
  <optional>
    <ref name="usml.role.persona"/>
  </optional>
</define>

<define name="usml.role.content-model">
  <interleave>
    <text/>
    <zeroOrMore>
      <ref name="usml.term"/>
    </zeroOrMore>
  </interleave>
</define>

<define name="usml.role">
  <element name="role">
    <ref name="usml.role.attribute-list"/>
    <ref name="usml.role.content-model"/>
  </element>
</define>

<!-- The definition of the goal element. -->
<define name="usml.goal.attribute-list">
  <optional>
    <ref name="usml.reference"/>
  </optional>
</define>

<define name="usml.goal.content-model">
  <interleave>
    <text/>
    <zeroOrMore>
      <ref name="usml.term"/>
    </zeroOrMore>
  </interleave>
</define>

<define name="usml.goal">
  <element name="goal">
    <ref name="usml.goal.attribute-list"/>
    <ref name="usml.goal.content-model"/>
  </element>
</define>
```

```
<!-- The definition of the value element. -->
<define name="usml.value.attribute-list">
  <optional>
    <ref name="usml.reference"/>
  </optional>
</define>

<define name="usml.value.content-model">
  <interleave>
    <text/>
    <zeroOrMore>
      <ref name="usml.term"/>
    </zeroOrMore>
  </interleave>
</define>

<define name="usml.value">
  <element name="value">
    <ref name="usml.value.attribute-list"/>
    <ref name="usml.value.content-model"/>
  </element>
</define>

<!-- The definition of the statement element. -->
<define name="usml.statement.attribute-list">
  <optional>
    <ref name="usml.xml-id"/>
  </optional>
</define>

<define name="usml.statement.content-model">
  <mixed>
    <group>
      <ref name="usml.role"/>
      <ref name="usml.goal"/>
      <optional>
        <ref name="usml.value"/>
      </optional>
    </group>
  </mixed>
</define>

<define name="usml.statement">
  <element name="statement">
    <ref name="usml.statement.attribute-list"/>
    <ref name="usml.statement.content-model"/>
  </element>
</define>

<!-- The definition of the constraint element. -->
<define name="usml.constraint.attribute-list">
  <ref name="usml.xml-id"/>
  <optional>
    <ref name="usml.constraint.type"/>
  </optional>
</define>
```

```
<define name="usml.constraint.content-model">
  <interleave>
    <text/>
    <zeroOrMore>
      <ref name="usml.term"/>
    </zeroOrMore>
  </interleave>
</define>

<define name="usml.constraint">
  <element name="constraint">
    <ref name="usml.constraint.attribute-list"/>
    <ref name="usml.constraint.content-model"/>
  </element>
</define>

<!-- The definition of the estimate element. -->
<define name="usml.estimate">
  <choice>
    <externalRef href="usml_estimate_story_points.rng"/>
    <externalRef href="usml_estimate_ideal_days.rng"/>
  </choice>
</define>

<!-- The definition of the priority element. -->
<define name="usml.priority">
  <choice>
    <externalRef href="usml_priority_basic.rng"/>
    <externalRef href="usml_priority_moscow.rng"/>
  </choice>
</define>

<!-- The definition of the note element. -->
<define name="usml.note.content-model">
  <interleave>
    <text/>
    <zeroOrMore>
      <ref name="usml.item"/>
    </zeroOrMore>
  </interleave>
</define>

<define name="usml.note">
  <element name="note">
    <ref name="usml.note.content-model"/>
  </element>
</define>

<!-- The definition of the test element. -->
<define name="usml.test">
  <choice>
    <externalRef href="usml_test_basic.rng"/>
    <externalRef href="usml_test_bdd.rng"/>
  </choice>
</define>
```

```
<!-- The definition of the acceptance-criteria element. -->
<define name="usml.acceptance-criteria.content-model">
  <oneOrMore>
    <ref name="usml.test"/>
  </oneOrMore>
</define>

<define name="usml.acceptance-criteria">
  <element name="acceptance-criteria">
    <ref name="usml.acceptance-criteria.content-model"/>
  </element>
</define>

<!-- The definition of the related-user-story element. -->
<define name="usml.related-user-story.attribute-list">
  <optional>
    <ref name="usml.reference"/>
  </optional>
  <ref name="usml.user-story.relationship.type"/>
</define>

<define name="usml.related-user-story.content-model">
  <ref name="usml.name"/>
</define>

<define name="usml.related-user-story">
  <element name="related-user-story">
    <ref name="usml.related-user-story.attribute-list"/>
    <ref name="usml.related-user-story.content-model"/>
  </element>
</define>

<!-- The definition of the user-story element. -->
<define name="usml.user-story.attribute-list">
  <ref name="usml.xml-id"/>
  <ref name="usml.version"/>
  <optional>
    <ref name="usml.user-story.type"/>
  </optional>
</define>
```

```
<define name="usml.user-story.content-model">
  <ref name="usml.name"/>
  <ref name="usml.date"/>
  <oneOrMore>
    <ref name="usml.author"/>
  </oneOrMore>
  <optional>
    <ref name="usml.status"/>
  </optional>
  <optional>
    <ref name="usml.iteration"/>
  </optional>
  <optional>
    <ref name="usml.theme"/>
  </optional>
  <optional>
    <ref name="usml.context-of-use"/>
  </optional>
  <ref name="usml.statement"/>
  <zeroOrMore>
    <ref name="usml.constraint"/>
  </zeroOrMore>
  <ref name="usml.estimate"/>
  <ref name="usml.priority"/>
  <optional>
    <ref name="usml.note"/>
  </optional>
  <ref name="usml.acceptance-criteria"/>
  <zeroOrMore>
    <ref name="usml.related-user-story"/>
  </zeroOrMore>
</define>

<define name="usml.user-story">
  <element name="user-story">
    <ref name="usml.user-story.attribute-list"/>
    <ref name="usml.user-story.content-model"/>
  </element>
</define>

</div>

<!-- The definition of the usml element. -->
<define name="usml.attribute-list">
  <ref name="usml.version"/>
  <ref name="usml.xml-lang"/>
  <optional>
    <ref name="usml.xml-space"/>
  </optional>
</define>
```

```
<define name="usml.content-model">
  <optional>
    <ref name="usml.title"/>
  </optional>
  <optional>
    <ref name="usml.organization"/>
  </optional>
  <optional>
    <ref name="usml.project"/>
  </optional>
  <optional>
    <ref name="usml.license"/>
  </optional>
  <optional>
    <ref name="usml.methodology"/>
  </optional>
  <optional>
    <ref name="usml.domain"/>
  </optional>
  <optional>
    <ref name="usml.maturity"/>
  </optional>
  <oneOrMore>
    <ref name="usml.user-story"/>
  </oneOrMore>
</define>

<define name="usml">
  <element name="usml">
    <ref name="usml.attribute-list"/>
    <ref name="usml.content-model"/>
  </element>
</define>

<!-- The reference to the root element. -->
<start>
  <ref name="usml"/>
</start>

</grammar>
```

The Ideal Days Estimate File (`usml_estimate_ideal_days.rng`)

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<!-- The definition of the approach attribute. -->
<define name="usml.estimate.ideal-days.approach">
  <attribute name="approach">
    <choice>
      <!-- The given choices for the approach. -->
      <value type="token">Analogy</value>
      <value type="token">Expert Judgment</value>
      <value type="token">Planning Poker</value>
      <!-- The approach is not among the given choices. -->
       <value type="token"/>
    </choice>
  </attribute>
</define>

<!-- The definition of the metric attribute. -->
<define name="usml.estimate.ideal-days.metric">
  <attribute name="metric">
    <value type="token">Ideal Days</value>
  </attribute>
</define>

<!-- The definition of user story estimate in ideal days. -->
<define name="usml.estimate.ideal-days.attribute-list">
  <ref name="usml.estimate.ideal-days.approach"/>
  <ref name="usml.estimate.ideal-days.metric"/>
</define>

<define name="usml.estimate.ideal-days.content-model">
  <data type="gDay"/>
</define>

<define name="usml.estimate.ideal-days">
  <element name="estimate">
    <ref name="usml.estimate.ideal-days.attribute-list"/>
    <ref name="usml.estimate.ideal-days.content-model"/>
  </element>
</define>

<start>
  <ref name="usml.estimate.ideal-days"/>
</start>

</grammar>
```

The Story Points Estimate File (`usml_estimate_story_points.rng`)

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<!-- The definition of the approach attribute. -->
<define name="usml.estimate.story-points.approach">
  <attribute name="approach">
    <choice>
      <!-- The given choices for the approach. -->
      <value type="token">Analogy</value>
      <value type="token">Expert Judgment</value>
      <value type="token">Planning Poker</value>
      <value type="token">Silent Grouping</value>
      <!-- The approach is not among the given choices. -->
       <value type="token"/>
    </choice>
  </attribute>
</define>


<!-- The definition of the metric attribute. -->
<define name="usml.estimate.story-points.metric">
  <attribute name="metric">
    <value type="token">Story Points</value>
  </attribute>
</define>

<!-- The definition of the range attribute. -->
<define name="usml.estimate.story-points.range">
  <attribute name="range">
    <choice>
      <value type="token">Fibonacci Sequence</value>
      <value type="token">Geometric Sequence</value>
      <!-- The name of range is not among the given choices. -->
      <value type="token"/>
    </choice>
  </attribute>
</define>

<!-- The definition of user story estimate in story points. -->
<define name="usml.estimate.story-points.attribute-list">
  <ref name="usml.estimate.story-points.approach"/>
  <ref name="usml.estimate.story-points.metric"/>
  <ref name="usml.estimate.story-points.range"/>
</define>

<define name="usml.estimate.story-points.content-model">
  <data type="positiveInteger"/>
</define>
```

```
<define name="usml.estimate.story-points">
  <element name="estimate">
    <ref name="usml.estimate.story-points.attribute-list"/>
    <ref name="usml.estimate.story-points.content-model"/>
  </element>
</define>

<start>
  <ref name="usml.estimate.story-points"/>
</start>
```

The Basic Priority File (`usml_priority_basic.rng`)

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<!-- The definition of the approach attribute. -->
<define name="usml.priority.basic.approach">
  <attribute name="approach">
    <value type="token">Basic</value>
  </attribute>
</define>

<!-- The definition of the type attribute. -->
<define name="usml.priority.basic.type">
  <attribute name="type">
    <value type="token">Relative</value>
  </attribute>
</define>

<!-- The definition of the criterion attribute. -->
<define name="usml.priority.basic.criterion">
  <attribute name="criterion">
    <list>
      <oneOrMore>
        <choice>
          <!-- The given choices for the criterion. -->
          <value type="token">Risk</value>
          <value type="token">Value</value>
          <!-- The criterion not among the given choices. -->
          <value type="token"/>
        </choice>
      </oneOrMore>
    </list>
  </attribute>
</define>
```

```
<!-- The definition of priority of a user story on a basic scheme. -->
<define name="usml.priority.basic.attribute-list">
  <ref name="usml.priority.basic.approach"/>
  <optional>
    <ref name="usml.priority.basic.type"/>
  </optional>
  <ref name="usml.priority.basic.criterion"/>
</define>

<define name="usml.priority.basic.content-model">
  <!-- The choices for prioritization. -->
  <choice>
    <value type="token">High</value>
    <value type="token">Medium</value>
    <value type="token">Low</value>
  </choice>
</define>

<define name="usml.priority.basic">
  <element name="priority">
    <ref name="usml.priority.basic.attribute-list"/>
    <ref name="usml.priority.basic.content-model"/>
  </element>
</define>

<start>
  <ref name="usml.priority.basic"/>
</start>

</grammar>
```

The MoSCoW Priority File (`usml_priority_moscow.rng`)

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<!-- The definition of the approach attribute. -->
<define name="usml.priority.moscow.approach">
  <attribute name="approach">
    <value type="token">MoSCoW</value>
  </attribute>
</define>

<!-- The definition of the type attribute. -->
<define name="usml.priority.moscow.type">
  <attribute name="type">
    <value type="token">Absolute</value>
  </attribute>
</define>
```

```
<!-- The definition of the criterion attribute. -->
<define name="usml.priority.moscow.criterion">
  <attribute name="criterion">
    <list>
      <oneOrMore>
        <choice>
          <!-- The given choices for the criterion. -->
          <value type="token">Risk</value>
          <value type="token">Value</value>
          <!-- The criterion not among the given choices. -->
          <value type="token"/>
        </choice>
      </oneOrMore>
    </list>
  </attribute>
</define>

<!--
     The definition of priority of a user story on the
     DSDM MoSCoW scheme.
-->
<define name="usml.priority.moscow.attribute-list">
  <ref name="usml.priority.moscow.approach"/>
  <optional>
    <ref name="usml.priority.moscow.type"/>
  </optional>
  <ref name="usml.priority.moscow.criterion"/>
</define>

<define name="usml.priority.moscow.content-model">
  <!-- The choices for prioritization. -->
  <choice>
    <value type="token">MUST</value>
    <value type="token">SHOULD</value>
    <value type="token">COULD</value>
    <value type="token">WOULD</value>
  </choice>
</define>

<define name="usml.priority.moscow">
  <element name="priority">
    <ref name="usml.priority.moscow.attribute-list"/>
    <ref name="usml.priority.moscow.content-model"/>
  </element>
</define>

<start>
  <ref name="usml.priority.moscow"/>
</start>

</grammar>
```

The Basic Test Style File (`usml_test_basic.rng`)

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<!-- The definition of a basic test style. -->
<define name="usml.test.basic.attribute-list">
  <parentRef name="usml.xml-id"/>
</define>

<define name="usml.test.basic.content-model">
  <text/>
</define>

<define name="usml.test.basic">
  <element name="test">
    <ref name="usml.test.basic.attribute-list"/>
    <ref name="usml.test.basic.content-model"/>
  </element>
</define>

<start>
  <ref name="usml.test.basic"/>
</start>

</grammar>
```

The BDD Test Style File (`usml_test_bdd.rng`)

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<!-- The definition of the name element. -->
<define name="usml.test.bdd.name.attribute-list">
  <optional>
    <parentRef name="usml.reference"/>
  </optional>
</define>

<define name="usml.test.bdd.name.content-model">
  <data type="token"/>
</define>
```

```
<define name="usml.test.bdd.name">
  <element name="name">
    <ref name="usml.test.bdd.name.attribute-list"/>
    <ref name="usml.test.bdd.name.content-model"/>
  </element>
</define>

<!-- The definition of a test style inspired by BDD. -->
<define name="usml.test.bdd.attribute-list">
  <parentRef name="usml.xml-id"/>
</define>

<define name="usml.test.bdd.content-model">
  <group>
    <ref name="usml.test.bdd.name"/>
    <oneOrMore>
      <element name="pre-condition">
        <text/>
      </element>
    </oneOrMore>
    <oneOrMore>
      <element name="event">
        <text/>
      </element>
    </oneOrMore>
    <oneOrMore>
      <element name="post-condition">
        <text/>
      </element>
    </oneOrMore>
  </group>
</define>

<define name="usml.test.bdd">
  <element name="test">
    <ref name="usml.test.bdd.attribute-list"/>
    <ref name="usml.test.bdd.content-model"/>
  </element>
</define>

<start>
  <ref name="usml.test.bdd"/>
</start>

</grammar>
```

A.3. Files Related to USML Extensions

There are three files related to the extensions of USML (relevant to Chapter 10).

Value Points File (`usml_value_points.rng`)

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<!-- The definition of the value-points element. -->
<define name="usml.value-points.attribute-list">
  <attribute name="range">
    <value type="token">Fibonacci Sequence</value>
  </attribute>
</define>

<define name="usml.value-points.content-model">
  <data type="positiveInteger"/>
</define>

<define name="usml.value-points">
  <element name="value-points">
    <ref name="usml.value-points.attribute-list"/>
    <ref name="usml.value-points.content-model"/>
  </element>
</define>

<start>
  <ref name="usml.value-points"/>
</start>

</grammar>
```

Status File (`usml_status.rng`)

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<!--
     The definition of the (user story) authoring status element
     as per the User Story Process Model.
-->
<define name="usml.status.authoring.content-model">
  <choice>
    <value type="token">Incomplete</value>
    <value type="token">Complete</value>
  </choice>
</define>
```

```
<define name="usml.status.authoring">
  <element name="status">
    <ref name="usml.status.authoring.content-model"/>
  </element>
</define>

<!--
     The definition of the (user story) review status element
     as per the User Story Process Model.
-->
<define name="usml.status.review.content-model">
  <choice>
    <value type="token">Pending</value>
    <value type="token">Incomplete</value>
    <value type="token">Complete</value>
  </choice>
</define>

<define name="usml.status.review">
  <element name="status">
    <ref name="usml.status.review.content-model"/>
  </element>
</define>

<!--
     The definition of the (user story) development status element.
-->
<define name="usml.status.development.content-model">
  <choice>
    <value type="token">Not Applicable</value>
    <value type="token">In Queue</value>
    <value type="token">In Development</value>
    <value type="token">Done</value>
  </choice>
</define>


<define name="usml.status.development">
  <element name="status">
    <ref name="usml.status.development.content-model"/>
  </element>
</define>

<!-- The definition of the status element. -->
<define name="usml.status">
   <choice>
    <ref name="usml.status.authoring"/>
    <ref name="usml.status.review"/>
    <ref name="usml.status.development"/>
  </choice>
</define>

<start>
  <ref name="usml.status"/>
</start>

</grammar>
```

Remark A.1. The `status` element is a manifestation of the use of the CONTAINER ELEMENT[59] pattern.

COSMIC Function Points File (`usml_estimate_cosmic.rng`)

```
<?xml version="1.0" encoding="UTF-8"?>

<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<!-- The definition of the approach attribute. -->
<define name="usml.estimate.cosmic.approach">
  <attribute name="approach">
    <choice>
      <!-- The given choices for the approach. -->
      <value type="token">Data Movement</value>
      <!-- The approach is not among the given choices. -->
       <value type="token"/>
    </choice>
  </attribute>
</define>

<!-- The definition of the metric attribute. -->
<define name="usml.estimate.cosmic.metric">
  <attribute name="metric">
    <value type="token">COSMIC Function Points</value>
  </attribute>
</define>

<!-- The user story estimate in COSMIC Function Points. -->
<define name="usml.estimate.cosmic.attribute-list">
  <ref name="usml.estimate.cosmic.approach"/>
  <ref name="usml.estimate.cosmic.metric"/>
</define>

<define name="usml.estimate.cosmic.content-model">
  <data type="positiveInteger"/>
</define>

<define name="usml.estimate.cosmic">
  <element name="estimate">
    <ref name="usml.estimate.cosmic.attribute-list"/>
    <ref name="usml.estimate.cosmic.content-model"/>
  </element>
</define>
```

[59] URL: http://www.xmlpatterns.com/ .

```
<start>
  <ref name="usml.estimate.cosmic"/>
</start>

</grammar>
```

A.4. Annotating the RELAX NG Schema for USML

The Dublin Core Metadata Element Set (DCMES)[60] is part of the Dublin Core Metadata Initiative. DCMES is a vocabulary of fifteen properties for use in resource description.

DCMES can provide more elaborate annotation than XML comments. The following is an example of extending the RELAX NG schema for USML using DCMES:

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xml:lang="en"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:dcterms="http://purl.org/dc/terms/"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<dc:title>RELAX NG Schema for USML 1.0</dc:title>
<dc:creator>Pankaj Kamthan</dc:creator>

<!-- Other Elements -->

<dcterms:educationLevel>
  Bachelor of Computer Science (or Bachelor of Software Engineering)
</dcterms:educationLevel>

<!-- Other Elements -->

<include href="usml.rng"/>

</grammar>
```

[60] URL: http://dublincore.org/documents/dcmi-terms/ .

Appendix B The Schematron Schema for USML

This appendix includes the listing of the Schematron Schema for USML, which is relevant to Chapters 7 and 8.

The namespace name URI for Schematron 1.5 is http://www.ascc.net/xml/schematron. The namespace name URI for ISO Schematron is http://purl.oclc.org/dsdl/schematron. The support for namespace name URI for Schematron can vary across processors.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--

Filename: usml.sch
Author: Pankaj Kamthan
Date: July 15, 2011
Version: 1.0
Description: This is the Schematron schema for the User Story Markup
             Language (USML) 1.0.
License: Creative Commons Attribution-Noncommercial-No Derivative Works
         3.0 Unported

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [IETF] RFC 2119.

-->

<sch:schema xmlns:sch="http://www.ascc.net/xml/schematron"
            xml:lang="en">

<title>Schematron Schema for USML 1.0</title>

<!-- RULES AT THE LEVEL OF USER STORY BOOK -->

<sch:pattern name="User Story Identifier">
  <sch:rule context="usml/user-story">
    <sch:assert test="count(//@xml:id[. = current()/@xml:id]) = 1">
      Each user story MUST have a unique identifier.
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

```
<sch:pattern name="User Story Name">
  <sch:rule context="usml/user-story">
    <sch:assert test="count(//name[. = current()/name]) = 1">
      Each user story MUST have a unique name.
    </sch:assert>
  </sch:rule>
</sch:pattern>

<sch:pattern name="Methodology and Type of Authors of a User Story">
  <sch:rule context="usml">
    <sch:assert test="
      (not(normalize-space(methodology = 'Extreme Programming'))
      or
      (count(user-story/author/@type[.='Customer']) >= 1))
      ">
      If the methodology is Extreme Programming, then at least one
      of the authors MUST be of type Customer.
    </sch:assert>
  </sch:rule>
</sch:pattern>

<sch:pattern name="Methodology and Corresponding Iteration of a User
Story">
  <sch:rule context="usml">
    <sch:assert test="
      (not(normalize-space(methodology/@type[.='Agile'])))
      or
      (methodology[@type = 'Agile']) and user-story/iteration
      ">
      If the methodology is an agile methodology, then there MUST be an
      iteration element.
    </sch:assert>
  </sch:rule>
</sch:pattern>

<sch:pattern name="Methodology and User Story Constraint">
  <sch:rule context="usml">
    <sch:assert test="
      not(methodology)
      or
      not(normalize-space(methodology = 'User-Centered Agile Process'))
      or
      ((user-story/constraint/@type = 'Accessibility')
      or
      (user-story/constraint/@type = 'Usability'))
      ">
      If the methodology is User-Centered Agile Process, then there
      MUST be an accessibility-related constraint or a usability-
      related constraint.
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

```
<sch:pattern name="Methodology and Test Style">
  <sch:rule context="usml">
    <sch:assert test="
      not(methodology)
      or
      not(normalize-space(methodology = 'Behaviour-Driven
      Development'))
      or
      ((user-story/acceptance-criteria/test/pre-condition)
      and
      (user-story/acceptance-criteria/test/event)
      and
      (user-story/acceptance-criteria/test/post-condition))
      ">
      If the methodology is Behaviour-Driven Development, then there
      MUST be pre-condition, event, and post-condition child elements
      of the test
      element.
    </sch:assert>
  </sch:rule>
</sch:pattern>

<!-- RULES AT THE LEVEL OF USER STORY -->

<sch:pattern name="Constraint Identifier">
  <sch:rule context="user-story/constraint">
    <sch:assert test="count(//@xml:id[. = current()/@xml:id]) = 1">
      Each constraint MUST have a unique identifier.
    </sch:assert>
  </sch:rule>
</sch:pattern>

<sch:pattern name="Test Identifier">
  <sch:rule context="acceptance-criteria/test">
    <sch:assert test="count(//@xml:id[. = current()/@xml:id]) = 1">
      Each test MUST have a unique identifier.
    </sch:assert>
  </sch:rule>
</sch:pattern>

<sch:pattern name="Match of Version and Status Information of a User
Story">
  <sch:rule context="user-story">
    <sch:assert test="
      not(status)
      or
      (normalize-space(status = 'Incomplete') and (@version &lt; 1))
      ">
      If the status of a user story is Incomplete, then the version of
      that user story SHOULD be less than 1.
      If the version of a user story is less than 1, then the status of
      that user story SHOULD be Incomplete.
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

```
<sch:pattern name="Number of User Story Authors">
  <sch:rule context="user-story">
    <sch:assert test="count(author) >= 2">
      A user story MUST have at least two authors.
    </sch:assert>
  </sch:rule>
</sch:pattern>

<sch:pattern name="Type of User Story Authors">
  <sch:rule context="user-story">
    <sch:assert test="
      (((count(author/@type[.='Internal']) >= 1) and
      (count(author/@type[.='External']) >= 1)) and
      ((count(author/@type[.='Software Engineer']) = 0) and
      (count(author/@type[.='Customer']) = 0) and
      (count(author/@type[.='User']) = 0)))
      or
      (((count(author/@type[.='Internal']) = 0) and
      (count(author/@type[.='External']) = 0)) and
      ((count(author/@type[.='Software Engineer']) >= 1) and
      (count(author/@type[.='Customer']) >= 1) and
      (count(author/@type[.='User']) = 0)))
      or
      (((count(author/@type[.='Internal']) = 0) and
      (count(author/@type[.='External']) = 0)) and
      ((count(author/@type[.='Software Engineer']) >= 1) and
      (count(author/@type[.='Customer']) = 0) and
      (count(author/@type[.='User']) >= 1)))
      ">
      If one of the authors of a user story is of type Internal,
      then the other(s) MUST be of type External.
      If one of the authors of a user story is of type Software
      Engineer, then the other(s) MUST either be of type Customer or
      of type User.
    </sch:assert>
  </sch:rule>
</sch:pattern>

<sch:pattern name="Match of User Story Type and Role Type">
  <sch:rule context="user-story[@type = 'Positive']">
    <sch:assert test="statement/role/@type = 'Positive'">
      If a user story is positive, then its (user) role MUST also be
      positive.
      If a (user) role is positive, then the user story MUST also be
      positive.
    </sch:assert>
  </sch:rule>
  <sch:rule context="user-story[@type = 'Negative']">
    <sch:assert test="statement/role/@type = 'Negative'">
      If a user story is negative, then its (user) role MUST also be
      negative.
      If a (user) role is negative, then the user story MUST also be
      negative.
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

```
<sch:pattern name="User Story Constraint and Context-of-Use">
  <sch:rule context="user-story">
    <sch:assert test="
      not(constraint/@type = 'Accessibility')
      or
      (count(context-of-use) = 1)
      ">
      If there is an accessibility-related constraint, then there
      SHOULD be context-of-use.
    </sch:assert>
    <sch:assert test="
      not(constraint/@type = 'Usability')
      or
      (count(context-of-use) = 1)
      ">
      If there is an usability-related constraint, then there SHOULD be
      context-of-use.
    </sch:assert>
  </sch:rule>
</sch:pattern>

<sch:pattern name="User Story Relationship">
  <sch:rule context="user-story">
    <sch:assert test="
      normalize-space(name/text()) !=
      normalize-space(related-user-story/name/text())
      ">
      A given user story MUST only be related to other user stories.
      (The name of a user story, say US, must not appear in the list
      of other user stories to which US is related.)
    </sch:assert>
  </sch:rule>
</sch:pattern>

<!-- RULES AT THE LEVEL OF AN ELEMENT -->

<sch:pattern name="Information in statement Element">
  <sch:rule context="statement">
    <sch:assert test="
      count(descendant::node()[contains(.,'and')]) = 0
      or
      count(descendant::node()[contains(.,'or')]) = 0
      ">
      If there is a conjunction or disjunction, the user story
      statement MAY be a compound statement.
      (The user story may be an epic.)
    </sch:assert>
  </sch:rule>
</sch:pattern>
```

```
<sch:pattern name="Information in test Element">
  <sch:rule context="test">
    <sch:assert test="
      count(descendant::node()[contains(.,'not')]) = 0
      ">
      A test SHOULD be expressed as a positive.
      (The inclusion of a term such as 'not' may mean that the test has
      been expressed as a negative.)
    </sch:assert>
  </sch:rule>
</sch:pattern>

<!-- RULES AT THE LEVEL OF AN ATTRIBUTE -->

<sch:pattern name="Information in criterion Attribute">
  <sch:rule context="priority">
    <sch:assert test="
      normalize-space(@criterion) and string-length() != 0
      ">
      The criterion attribute MUST express relevant information.
      The presence of an empty string or only white space is not
      considered relevant.
    </sch:assert>
    <sch:assert test="
      (count(@criterion[contains(.,'Risk')]) = 1)
      or
      (count(@criterion[contains(.,'Value')]) = 1)
      ">
      Each criterion for priority of a user story MUST appear only
      once.
    </sch:assert>
  </sch:rule>
</sch:pattern>

<sch:pattern name="Information in type (of related-user-story)
                   Attribute">
  <sch:rule context="related-user-story">
    <sch:assert test="
      normalize-space(@type) and string-length() != 0
      ">
      The type attribute MUST express relevant information.
      The presence of an empty string or only white space is not
      considered relevant.
    </sch:assert>
    <sch:assert test="(
      (count(@type[contains(.,'Similar-To')]) = 1)
      or
      (count(@type[contains(.,'Resource-Contender-Of')]) = 1)
      or
      (count(@type[contains(.,'Consequence-Of')]) = 1)
      or
      (count(@type[contains(.,'Required-By')]) = 1)
      or
      (count(@type[contains(.,'Occurs-Before')]) = 1)
      or
      (count(@type[contains(.,'Occurs-Concurrently-With')]) = 1)
      or
      (count(@type[contains(.,'Occurs-After')]) = 1)
      )">
      The relationship type of a user story MUST appear only once.
    </sch:assert>
  </sch:rule>
</sch:pattern>

<sch:pattern name="Information in version Attribute">
  <sch:rule context="usml">
    <sch:assert test="@version >= 1.0">
      The version of USML MUST be greater or equal to 1.0.
    </sch:assert>
  </sch:rule>
  <sch:rule context="user-story">
    <sch:assert test="@version > 0">
      The version of a user story MUST be greater than 0.
    </sch:assert>
  </sch:rule>
</sch:pattern>

</sch:schema>
```

Appendix C A Collection of USML Instance Documents

This appendix includes the listing of USML instance documents, which are relevant to Chapter 9. These documents must, before use, be associated with appropriate processing instruction, as shown in Section 9.4.4.

C.1. File for Examples 9.2 and 9.4

USML Instance Document for Examples 9.2 and 9.4 (`usml_example.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>

<!--

Filename: usml_example.xml
Author: Pankaj Kamthan
Date: July 15, 2011
Version: 1.0
Description: This is a USML instance document for a banking
             application.
License: Creative Commons Attribution-Noncommercial-No Derivative Works
         3.0 Unported

-->

<!DOCTYPE usml [
  <!ENTITY eacute "&#233;">
  <!ENTITY le     "&#8804;">
]>

<usml version="1.0" xml:lang="en">

<!-- User Story Book Information -->
<title>
  User Story Book for iBank
</title>

<project>
  iBank
</project>

<methodology type="Agile">
  Scrum
</methodology>
```

```
<domain>
  Banking
</domain>

<!-- User Story Information -->
<user-story xml:id="US1" version="1.0">

  <name reference="http://www.google.ca/search?q=Withdraw+Money">
    Withdraw Money
  </name>

  <date>July 15, 2011</date>

  <author type="Customer">
    John Smith
  </author>
  <author type="Software Engineer">
    Jos&eacute;phine Marceau
  </author>

  <statement>
    A <role>bank customer</role> can <goal>withdraw money from a bank
    account</goal>.
  </statement>

  <estimate approach="Planning Poker"
            metric="Story Points"
            range="Fibonacci Sequence">
    3
  </estimate>

  <priority approach="MoSCoW"
            criterion="Risk Value">
    MUST
  </priority>

  <acceptance-criteria>
    <test xml:id="T1">
      The bank card inserted is valid.
    </test>
    <test xml:id="T2">
      The PIN is valid.
    </test>
    <test xml:id="T3">
      The amount requested for withdrawal is &le; the balance of the
      account.
    </test>
    <test xml:id="T4">
      The amount requested for withdrawal is &le; $500.
    </test>
    <test xml:id="T5">
      The amount dispensed is a multiple of $20.
    </test>
  </acceptance-criteria>
</user-story>

</usml>
```

C.2. File for Example 9.3

USML Instance Document for Example 9.3 (`usml_example2.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>

<!--

Filename: usml_example2.xml
Author: Pankaj Kamthan
Date: July 15, 2011
Version: 1.0
Description: This is a USML instance document for a banking
             application.
License: Creative Commons Attribution-Noncommercial-No Derivative Works
         3.0 Unported

-->

<usml version="1.0" xml:lang="en">

<!-- User Story Book Information -->
<title>
  User Story Book for iBank
</title>

<project>
  iBank
</project>

<methodology type="Agile">
  Scrum
</methodology>

<domain>
  Banking
</domain>

<!-- User Story Information -->
<user-story xml:id="US1" version="1.0">

  <name>Withdraw Money</name>

  <date>July 15, 2011</date>

  <author type="Customer">
    John Smith
  </author>
  <author type="Software Engineer">
    Jane Marshall
  </author>
  <statement>
    A <role>bank customer</role> can <goal>withdraw money from a bank
    account</goal>.
  </statement>

  <estimate approach="Planning Poker"
            metric="Story Points"
            range="Fibonacci Sequence">
    3
  </estimate>

  <priority approach="MoSCoW" criterion="Risk Value">MUST</priority>

  <acceptance-criteria>
    <test xml:id="T1">
      The bank card inserted is valid.
    </test>
    <test xml:id="T2">
      The PIN is valid.
    </test>
    <test xml:id="T3">
      The amount requested for withdrawal is less than or equal to the
      balance of the account.
    </test>
    <test xml:id="T4">
      The amount requested for withdrawal is less than or equal to
      $500.
    </test>
    <test xml:id="T5">
      The amount dispensed is a multiple of $20.
    </test>
  </acceptance-criteria>
</user-story>

</usml>
```

Appendix D A Collection of CSS and XSLT Style Sheets for USML Instance Documents

This appendix includes the listing of CSS and XSLT style sheets associated with the USML instance documents of Appendix C, and relevant to Chapter 9.

D.1. Files for Example 9.2

CSS Style Sheet (`usml_example_thumbnail.css`) for Figure 9.11(a)

```
/*

Filename: usml_example_thumbnail.css
Author: Pankaj Kamthan
Date: July 15, 2011
Version: 1.0
Description: This is a CSS style sheet for a thumbnail-style
             presentation of a USML instance document.
License: Creative Commons Attribution-Noncommercial-No Derivative Works
         3.0 Unported

*/

/* Style Rules for All */
* {
  display:     inherit;
  font-family: sans-serif;
  color:       #000000;
}

/* Style Rules for User Story Book Information */
usml {
  display:          block;
  background-color: #ffffdd;
}

/* Title */
title {
  display: none;
}

/* Project */
project {
  display: none;
}
```

```
/* Methodology */
methodology {
  display: none;
}

/* Domain */
domain {
  display: none;
}

/* Style Rules for User Story Information */

/* User Story */
user-story {
  margin-top:       40px;
  margin-left:      40px;
  display:          table;
  width:            200px;
  border:           2px #d79900 solid;
  background-color: #ffffff;
}

/* Name */
name {
  margin-top:       8px;
  margin-left:      34px;
  margin-bottom:    15px;
  display:          inline-table;
  padding-top:      5px;
  padding-bottom:   5px;
  padding-left:     8px;
  padding-right:    8px;
  background-color: #a85400;
  font-size:        0.9em;
  font-weight:      bold;
  color:            #ffffff;
  text-align:       center;
}

/* Date */
date {
  display: none;
}

/* Author */
author {
  display: none;
}

/* Statement */
statement {
  display: none;
}
```

```
/* Role */
role {
  display: none;
}

/* Goal */
goal {
  display: none;
}

/* Value */
value {
  display: none;
}

/* Estimate */
estimate:before {
  content:   'Estimate: ';
  font-size: 0.82em;
  color:     #000000;
}

estimate {
  margin-left:   10px;
  margin-bottom: 2px;
  font-size:     0.82em;
  color:         #008000;
}

estimate:after {
  content:   attr(metric);
  font-size: 0.82em;
  color:     #000000;
}

/* Priority */
priority:before {
  content:   'Priority: ';
  font-size: 0.82em;
  color:     #000000;
}

priority {
  margin-left:   10px;
  margin-bottom: 5px;
  font-size:     0.82em;
  color:         #dc143c;
}

priority:after {
  content:   ' ('attr(approach)')';
  font-size: 0.83em;
  color:     #000000;
}
```

```
/* Acceptance Criteria */
acceptance-criteria {
  display: none;
}

/* Test */
test {
  display: none;
}
```

CSS Style Sheet (`usml_example_index_card.css`) for Figure 9.11(b)

```
/*

Filename: usml_example_index_card.css
Author: Pankaj Kamthan
Date: July 15, 2011
Version: 1.0
Description: This is a CSS style sheet for an 'index card'-style
             presentation of a USML instance document.
License: Creative Commons Attribution-Noncommercial-No Derivative Works
         3.0 Unported

*/

/* Style Rules for All */
* {
  display:     inherit;
  font-family: sans-serif;
  color:       #000000;
}

/* Style Rules for User Story Book Information */
usml {
  margin-top:  50px;
  margin-left: 50px;
  display:     block;
  background:  url('index_card.png') no-repeat;
}

/* Title */
title {
  display: none;
}

/* Project */
project {
  display: none;
}

/* Methodology */
methodology {
  display: none;
}
```

```
/* Domain */
domain {
  display: none;
}

/* Style Rules for User Story Information */

/* Name */
name {
  margin-top:       10px;
  margin-left:      115px;
  margin-bottom:    15px;
  display:          inline-table;
  padding-top:      8px;
  padding-bottom:   8px;
  padding-left:     8px;
  padding-right:    8px;
  background-color: #00698C;
  font-size:        1.1em;
  font-weight:      bold;
  color:            #ffffff;
  text-align:       center;
}

/* Date */
date {
  margin-left:   15px;
  margin-bottom: 40px;
  font-size:     0.8em;
}

/* Author */
author {
  display: none;
}

/* Statement */
statement {
  margin-top:    20px;
  margin-left:   15px;
  margin-bottom: 34px;
  font-size:     0.9em;
}

/* Role */
role {
  display:       inline;
  border-bottom: 0.15em solid;
}

/* Goal */
goal {
  display:       inline;
  border-bottom: 0.15em solid;
}
```

```
/* Value */
value {
  display:       inline;
  border-bottom: 0.15em solid;
}

/* Estimate */
estimate:before {
  content: 'Estimate: ';
  font:    110% sans-serif;
  color:   #000000;
}

estimate {
  margin-left:   15px;
  margin-bottom: 5px;
  font-weight:   bold;
  color:         #008000;
}

estimate:after {
  content:     attr(metric);
  font-weight: normal;
  color:       #000000;
}

/* Priority */
priority:before {
  content: 'Priority: ';
  font:    110% sans-serif;
  color:   #000000;
}

priority {
  margin-left: 15px;
  font-weight: bold;
  color:       #dc143c;
}

priority:after {
  content:     ' ('attr(approach)')';
  font-weight: normal;
  color:       #000000;
}

/* Acceptance Criteria */
acceptance-criteria {
  display: none;
}

/* Test */
test {
  display: none;
}
```

The Image of an Index Card (`index_card.png`) for Figure 9.11(b)

CSS Style Sheet (`usml_example_block.css`) for Figure 9.11(c)

```
/*

Filename: usml_example_block.css
Author: Pankaj Kamthan
Date: July 15, 2011
Version: 1.0
Description: This is a CSS style sheet for a block-style presentation
             of a USML instance document.
License: Creative Commons Attribution-Noncommercial-No Derivative Works
         3.0 Unported

*/

/* Style Rules for All */
* {
  display:     inherit;
  font-family: sans-serif;
  color:       #000000;
}

/* Style Rules for User Story Book Information */
usml {
  display:          block;
  background-color: #f0f0f0;
}

/* Title */
title {
  margin-top:    20px;
  margin-left:   80px;
  margin-bottom: 20px;
  font:          small-caps 150% sans-serif;
}
```

```
/* Project */
project:before {
  content: 'Project: ';
  font:    120% sans-serif;
}

project {
  margin-left:   80px;
  margin-bottom: 20px;
}

/* Methodology */
methodology {
  display: none;
}

/* Domain */
domain {
  display: none;
}

/* Style Rules for User Story Information */

/* User Story */
user-story {
  margin-left:      80px;
  display:          table;
  padding-top:      8px;
  padding-bottom:   8px;
  padding-left:     8px;
  padding-right:    8px;
  border:           solid 0.1em #000000;
  background-color: #ffffff;
}

/* Name */
name {
  margin-bottom:    18px;
  display:          inline-table;
  padding-top:      8px;
  padding-bottom:   8px;
  padding-left:     8px;
  padding-right:    8px;
  border:           solid 0.1em #c0c0c0;
  background-color: #3c3c3c;
  font-size:        0.9em;
  font-weight:      bold;
  color:            #ffffff;
  text-align:       center;
}

/* Date */
date {
  margin-bottom: 18px;
  font-size:     0.8em;
}
```

```
/* Author */
author:before {
  content: attr(type)': ';
}

author {
  font-size:   0.8em;
  font-weight: lighter;
}

/* Statement */
statement:before {
  content: 'User Story: ';
  font:    110% sans-serif;
}

statement {
  margin-top:    20px;
  margin-bottom: 18px;
}

/* Role */
role {
  display: inline;
  color:   #000080;
}

/* Goal */
goal {
  display: inline;
}

/* Value */
value {
  display: inline;
}

/* Estimate */
estimate:before {
  content: 'Estimate: ';
  font:    110% sans-serif;
  color:   #000000;
}

estimate {
  margin-bottom: 18px;
  color:         #008000;
}

estimate:after {
  content: attr(metric);
  color:   #000000;
}
```

```
/* Priority */
priority:before {
  content: 'Priority: ';
  font:    110% sans-serif;
  color:   #000000;
}

priority {
  margin-bottom: 18px;
  color:         #dc143c;
}

priority:after {
  content: ' ('attr(approach)')';
  color:   #000000;
}

/* Acceptance Criteria */
acceptance-criteria:before {
  display:  table-cell;
  content:  'Acceptance Criteria: ';
  font:     110% sans-serif;
}

/* Test */
test {
  margin-left: 20px;
  display:     list-item;
  list-style:  square outside;
  font-size:   0.8em;
}
```

D.2. File for Example 9.3

XSLT Style Sheet (`usml_example_text.xsl`) for Example 9.3

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!--

Filename: usml_example_text.xsl
Author: Pankaj Kamthan
Date: July 15, 2011
Version: 1.0
Description: This is an XSLT style sheet for a 'plain' text
             presentation of a USML instance document.
License: Creative Commons Attribution-Noncommercial-No Derivative Works
         3.0 Unported

-->
```

```
<xsl:output method="text"/>

<xsl:strip-space elements="*"/>

<xsl:template match="/">

<xsl:text>Title:&#x20;</xsl:text>
<xsl:value-of select="normalize-space(usml/title)"/>
<xsl:text>&#xA;&#xA;</xsl:text>

<!-- The set of rules applicable to each user story in USML. -->
<xsl:for-each select="usml/user-story">

  <xsl:text>Name:&#x20;</xsl:text>
  <xsl:value-of select="normalize-space(name)"/>
  <xsl:text>&#xA;&#xA;</xsl:text>

  <xsl:text>Date:&#x20;</xsl:text>
  <xsl:value-of select="normalize-space(date)"/>
  <xsl:text>&#xA;&#xA;</xsl:text>

  <xsl:for-each select="author">
    <xsl:text>Author:&#x20;</xsl:text>
    <xsl:value-of select="normalize-space(.)" />
    <xsl:text>&#xA;</xsl:text>
  </xsl:for-each>
  <xsl:text>&#xA;</xsl:text>

  <xsl:text>Statement:&#x20;</xsl:text>
  <xsl:value-of select="normalize-space(statement)"/>
  <xsl:text>&#xA;&#xA;</xsl:text>

  <xsl:text>Estimate:&#x20;</xsl:text>
  <xsl:value-of select="normalize-space(estimate)"/>
  <xsl:text>&#x20;</xsl:text>
  <xsl:value-of select="estimate/@metric"/>
  <xsl:text>&#xA;</xsl:text>

  <xsl:text>Priority:&#x20;</xsl:text>
  <xsl:value-of select="normalize-space(priority)"/>
  <xsl:text>&#x20;(</xsl:text>
  <xsl:value-of select="priority/@approach"/>
  <xsl:text>)</xsl:text>
  <xsl:text>&#xA;&#xA;</xsl:text>

  <xsl:text>Acceptance Criteria:</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:for-each select="acceptance-criteria/test">
      <xsl:text>-&#x20;</xsl:text>
      <xsl:value-of select="normalize-space(.)"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>

</xsl:for-each>
```

```
</xsl:template>

</xsl:stylesheet>
```

D.3. Files for Example 9.4

CSS Style Sheet (`usml_example_table.css`) for Example 9.4

```
/*

Filename: usml_example_table.css
Author: Pankaj Kamthan
Date: July 15, 2011
Version: 1.0
Description: This is a CSS style sheet for an instance document of
             USML.
License: Creative Commons Attribution-Noncommercial-No Derivative Works
         3.0 Unported

*/

/* Presentation of Table */

table {
  margin-top:       20px;
  margin-left:      40px;
  margin-bottom:    20px;
  width:            600px;
  border:           1px #3c3c3c solid;
  border-collapse:  collapse;
  background-color: #ffffff;
  font-family:      sans-serif;
  color:            #000000;
}

th {
  height:           30px;
  width:            300px;
  border:           1px #3c3c3c solid;
  background-color: #d4d4d4;
  color:            #3c3c3c;
  font-size:        0.95em;
}

td {
  width:   300px;
  border:  1px #3c3c3c solid;
  padding: 8px;
}
```

```
/* Presentation of User Story Information */

div.date {
  margin-bottom: 8px;
  font-size:     0.8em;
  color:         #000080;
}


div.author {
  font-size: 0.8em;
}

div.statement {
  margin-top:    20px;
  margin-bottom: 20px;
}

div.estimate {
  font-size: 0.9em;
}

div.priority {
  font-size: 0.9em;
}

ul.acceptance-criteria {
  margin-left: -10px;
}

li.test {
  margin-top:      2px;
  list-style-type: square;
  font-size:       0.9em;
}

/* Presentation of Links */

a:link {
  background:      none;
  border-bottom:   2px solid;
  color:           #0000ff;
  text-decoration: none;
}

a:active {
  background:      none;
  color:           #ff0000;
  text-decoration: underline;
}

a:visited {
  background:      none;
  border-bottom:   2px solid;
  color:           #400080;
  text-decoration: none;
}
```

```
a:hover {
  background: #000000;
  color:      #ffffff;
}
```

XSLT Style Sheet (`usml_example_table.xsl`) for Example 9.4

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns="http://www.w3.org/1999/xhtml">

<!--

Filename: usml_example_table.xsl
Author: Pankaj Kamthan
Date: July 15, 2011
Version: 1.0
Description: This is an XSLT style sheet for a tabular presentation of
             a USML instance document.
License: Creative Commons Attribution-Noncommercial-No Derivative Works
         3.0 Unported

-->

<xsl:output method="xml"
            media-type="application/xhtml+xml"
            encoding="UTF-8"
            doctype-public="-//W3C//DTD XHTML Basic 1.1//EN"
            doctype-system="http://www.w3.org/TR/xhtml-basic/xhtml-
basic11.dtd"
            indent="yes"/>

<xsl:template match="/">
<html>

    <!-- The head section. -->
    <head>
      <title>
        <xsl:value-of select="usml/title"/>
      </title>
      <meta name="DC.format" content="USML"/>
      <link rel="stylesheet"
            type="text/css"
            href="usml_example_table.css"/>
    </head>

    <!-- The body section. -->
    <body>

    <!-- The set of rules applicable to each user story in USML. -->
    <xsl:for-each select="usml/user-story">
```

```
<table>

  <tr>
    <!-- The arrangement for table header. -->
    <th>
      <!--
           The user story name and association of a URI with
           the name.
      -->
      <a>
        <xsl:attribute name="href">
          <xsl:value-of select="name/@reference"/>
        </xsl:attribute>
        <xsl:value-of select="name"/>
      </a>
    </th>
    <th>
      Acceptance Criteria
    </th>
  </tr>

  <tr>
    <!--
         The arrangement for table column as 'front' of index card.
    -->
    <td>
      <div class="date">
        <xsl:value-of select="date"/>
      </div>
      <xsl:for-each select="author">
        <div class="author">
          <xsl:value-of select="./@type"/>:
          <xsl:value-of select="."/>
          <br/>
        </div>
      </xsl:for-each>
      <div class="statement">
        <xsl:value-of select="statement"/>
      </div>
      <div class="estimate">
        Estimate:
        <xsl:value-of select="estimate"/>
        <xsl:value-of select="estimate/@metric"/>
      </div>
      <div class="priority">
        Priority:
        <xsl:value-of select="priority"/>
        (<xsl:value-of select="priority/@approach"/>)
      </div>
    </td>
```

```
        <!--
             The arrangement for table column as 'back' of index card.
        -->
        <td>
          <ul class="acceptance-criteria">
            <xsl:for-each select="acceptance-criteria/test">
              <li class="test">
                <xsl:value-of select="."/>
              </li>
            </xsl:for-each>
          </ul>
        </td>
      </tr>

    </table>

    </xsl:for-each>

    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Appendix E Tools for USML

There are a number of freely available tools used in the thesis. The following sections provide a summary of those tools and their purpose of use.

E.1. Tools Used for USML Schema Documents

- USML-related products (instance documents, schema documents, and style sheets) are independent of any authoring environment. However, special-purpose editors can aid authoring. The RELAX NG schema for USML, the Schematron schema for USML, the USML instance documents (given as examples), the CSS style sheets for USML, and the XSLT style sheets for USML, are authored using GNU Emacs equipped with nXML[61] mode.
- The RELAX NG schema for USML is validated using Jing[62] and Oracle Multi-Schema XML Validator (MSV)[63] .
- The USML Schematron schema is validated using Topologi Schematron Validator[64].
- The RELAX NG schema for USML can be converted to XSDL using Trang[65]. However, the thesis does not provide the result of such a conversion. (The RELAX NG schema for USML can not, as-is, be converted to XML DTD.)
- The NVDL schema is validated using oNVDL[66]. (ONVDL can be used independently, or as part of the Oxygen XML Editor[67].)

[61] URL: http://www.thaiopensource.com/nxml-mode/ .
[62] URL: http://www.thaiopensource.com/relaxng/jing.html .
[63] URL: http://msv.java.net/ .
[64] URL: http://www.topologi.com/products/validator/ .
[65] URL: http://www.thaiopensource.com/relaxng/trang.html .
[66] URL: http://www.oxygenxml.com/onvdl.html .

E.2. Tools Used for USML Style Sheets

- The CSS style sheets for USML are validated using the W3C CSS Validation Service[68].
- The XSLT style sheets for USML are validated using Saxon[69], specifically, Saxon-HE.
- The color contrast is checked using Colour Contrast Check Tool[70].

E.3. Tools Used for USML Instance Documents

- The USML instance documents are validated using Jing.
- The USML instance documents intended for relatively large-screen, stationary devices, such as a notebook computer, are rendered using Mozilla Firefox[71].
- The USML instance documents intended for relatively small-screen, non-stationary devices, such as a tablet computer, are rendered using Opera Mobile Emulator[72].

[67] URL: http://www.oxygenxml.com/ .
[68] URL: http://jigsaw.w3.org/css-validator/ .
[69] URL: http://saxon.sourceforge.net/ .
[70] URL: http://snook.ca/technical/colour_contrast/colour.html .
[71] URL: http://www.mozilla.com/ .
[72] URL: http://www.opera.com/developer/tools/mobile/ .

References

The secret to creativity is knowing how to hide your sources.
— Albert Einstein

[Adolph, Bramble, Cockburn, Pols, 2003] Patterns for Effective Use Cases. By S. Adolph, P. Bramble, A. Cockburn, A. Pols. Addison-Wesley. 2003.

[Adzic, 2009] Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing. By G. Adzic. Neuri Limited. 2009.

[Ajmeri, Sejpal, Ghaisas, 2010] A Semantic and Collaborative Platform for Agile Requirements Evolution. By N. Ajmeri, R. Sejpal, S. Ghaisas. The Third International Workshop on Managing Requirements Knowledge (MARK 2010). Sydney, Australia. September 27, 2010.

[Alexander, 1979] The Timeless Way of Building. By C. Alexander. Oxford University Press. 1979.

[Alexander, Beus-Dukic, 2009] Discovering Requirements: How to Specify Products and Services. By I. Alexander, L. Beus-Dukic. John Wiley and Sons. 2009.

[Alexander, Maiden, 2004] Scenarios, Stories, Use Cases through the Systems Development Life-Cycle. By I. Alexander, N. Maiden. John Wiley and Sons. 2004.

[Ambler, 2002] Agile Modeling. By S. W. Ambler. John Wiley and Sons. 2002.

[Anderson, Schragenheim, 2004] Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results. By D. J. Anderson, E. Schragenheim. Prentice-Hall. 2004.

[Bakalova, Daneva, Herrmann, Wieringa, 2011] Agile Requirements Prioritization: What Happens in Practice and What Is Described in Literature. By Z. Bakalova, M.

Daneva, A. Herrmann, R. Wieringa. The Seventeenth International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2011). Essen, Germany. March 28-30, 2011.

[Bang, 2007] An Agile Approach to Requirement Specification. By T. J. Bang. The Eighth International Conference on Agile Processes in Software Engineering and Extreme Programming (XP 2007). Como, Italy. June 18-22, 2007.

[Beck, 2000] Extreme Programming Explained: Embrace Change. By K. Beck. Addison-Wesley. 2000.

[Beck, Andres, 2005] Extreme Programming Explained: Embrace Change. By K. Beck, C. Andres. Second Edition. Addison-Wesley. 2005.

[Berander, Andrews, 2005] Requirements Prioritization. By P. Berander, A. Andrews. In: Engineering and Managing Software Requirements. A. Aurum, C. Wohlin (Editors). Springer-Verlag. 2005. Pages 69-94.

[Berczuk, Appleton, 2003] Software Configuration Management Patterns: Effective Teamwork, Practical Integration. By S. Berczuk, B. A. Appleton. Addison-Wesley. 2003.

[Berglund, Boag, Chamberlin, Fernández, Kay, Robie, Siméon, 2010] XML Path Language (XPath) 2.0 (Second Edition). By A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, J. Siméon (Editors). World Wide Web Consortium (W3C) Recommendation. December 14, 2010.

[Berners-Lee, Connolly, 1998] Web Architecture: Extensible Languages. By T. Berners-Lee, D. Connolly (Authors). World Wide Web Consortium (W3C) Note. February 10, 1998.

[Beyer, 2010] User-Centered Agile Methods. By H. Beyer. Morgan and Claypool. 2010.

[Boehm, 2011] Some Future Software Engineering Opportunities and Challenges. By B. Boehm. In: The Future of Software Engineering. S. Nanz (Editor). Springer-Verlag. 2011. Pages 1-32.

[Boehm, Turner, 2004] Balancing Agility and Discipline: A Guide for the Perplexed. By B. W. Boehm, R. Turner Addison-Wesley. 2004.

[Bonneau, Kohl, Tennison, Duckett, Williams, 2003] XML Design Handbook. By S. Bonneau, T. Kohl, J. Tennison, J. Duckett, K. Williams. Wrox Press. 2003.

[Bos, Çelik, Hickson, Lie, 2011] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. By B. Bos, T. Çelik, I. Hickson, H. W. Lie (Editors). World Wide Web Consortium (W3C) Recommendation. June 7, 2011.

[Bradley, 2002] The XML Companion. By N. Bradley. Third Edition. Addison-Wesley. 2002.

[Bradner, 1997] Key words for use in RFCs to Indicate Requirement Levels. By S. Bradner. Request for Comments (RFC) 2119. The Internet Society. 1997.

[Bray, Hollander, Layman, Tobin, Thompson, 2009] Namespaces in XML 1.0 (Third Edition). By T. Bray, D. Hollander, A. Layman, R. Tobin, H. S. Thompson. (Editors). World Wide Web Consortium (W3C) Recommendation. December 8, 2009.

[Bray, Paoli, Sperberg-McQueen, Maler, Yergeau, 2008] Extensible Markup Language (XML) 1.0 (Fifth Edition). By T. Bray, J. Paoli, M. Sperberg-McQueen, E. Maler, F. Yergeau (Editors). World Wide Web Consortium (W3C) Recommendation. November 26, 2008.

[Breitman, Leite, 2002] Managing User Stories. By K. K. Breitman, J. C. S. do Prado Leite. The Tenth International Requirements Engineering Conference (RE 2002). Essen, Germany. September 9-13, 2002.

[Brooks, 1987] No Silver Bullet: Essence and Accidents of Software Engineering. By F. P. Brooks, Jr. Computer. Volume 20. Number 4. 1987. Pages 10-19.

[Brown, Lindgaard, Biddle, 2008] Stories, Sketches, and Lists: Developers and Interaction Designers Interacting Through Artefacts. By J. Brown, G. Lindgaard, R. Biddle. Agile 2008 Conference. Toronto, Canada. August 4-8, 2008.

[Bunse, Feldmann, Dörr, 2004] Agile Methods in Software Engineering Education. By C. Bunse, R. L. Feldmann, J. Dörr. The Fifth International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2004). Garmisch-Partenkirchen, Germany. June 6-10, 2004.

[Buschmann, Henney, Schmidt, 2007] Pattern-Oriented Software Architecture: On Patterns and Pattern Languages, Volume 5. By F. Buschmann, K. Henney, D. C. Schmidt. John Wiley and Sons. 2007.

[Buschmann, Meunier, Rohnert, Sommerlad, Stal, 1996] Pattern-Oriented Software Architecture: A System of Patterns, Volume 1. By F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. John Wiley and Sons. 1996.

[Bush, 1945] As We May Think. By V. Bush. The Atlantic Monthly. July 1945. Pages 101-108.

[Buxton, 2007] Sketching User Experiences: Getting the Design Right and the Right Design. By B. Buxton. Morgan Kaufmann. 2007.

[Caldwell, Chisholm, Reid, Vanderheiden, 2008] Web Content Accessibility Guidelines 2.0. By B. Caldwell, W. Chisholm, L. G. Reid, G. Vanderheiden (Editors). World Wide Web Consortium (W3C) Recommendation. December 11, 2008.

[Canemaker, 1999] Paper Dreams: The Art and Artists of Disney Storyboards. By J. Canemaker. Hyperion Press. 1999.

[Chelimsky, Astels, Dennis, Hellesoy, Helmkamp, North, 2010] The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends. By D. Chelimsky, D. Astels, Z. Dennis, A. Hellesoy, B. Helmkamp, D. North. Pragmatic Bookshelf. 2010.

[Chow, Cao, 2008] A Survey Study of Critical Success Factors in Agile Software Projects. By T. Chow, D.-B. Cao. Journal of Systems and Software. Volume 81. Issue 6. 2008. Pages 961-971.

[Clark, 1999] Associating Style Sheets with XML documents Version 1.0. By J. Clark (Editor). World Wide Web Consortium (W3C) Recommendation. June 29, 1999.

[Carlson, 2001] Modeling XML Applications with UML: Practical E-Business Applications. By D. Carlson. Addison-Wesley. 2001.

[CMMI Product Team, 2010] CMMI® for Development, Version 1.3. By CMMI Product Team. Technical Report CMU/SEI-2010-TR-033 ESC-TR-2010-033. Software Engineering Institute. Carnegie Mellon University. Pittsburgh, U.S.A. 2010.

[Cockburn, 2001] Writing Effective Use Cases. By A. Cockburn. Addison-Wesley. 2001.

[Cockburn, 2007] Agile Software Development: The Cooperative Game. By A. Cockburn. Second Edition. Addison-Wesley. 2007.

[Cohn, 2004] User Stories Applied: For Agile Software Development. By M. Cohn. Addison-Wesley. 2004.

[Cohn, 2005] Agile Estimating and Planning. By M. Cohn. Prentice-Hall. 2005.

[Cohn, Paul, 2001] A Comparison of Requirements Engineering in Extreme Programming (XP) and Conventional Software Development Methodologies. By T. Cohn, R. Paul. The Seventh Americas Conference on Information Systems (AMCIS 2001). Boston, U.S.A. August 3-5, 2001.

[Colborne, 2010] Ask, "What Would the User Do?" (You Are Not the User). By G. Colborne. In: 97 Things Every Programmer Should Know: Collective Wisdom from the Experts. K. Henney (Editor). O'Reilly Media. 2010. Pages 6-7.

[Collier, 2012] Agile Analytics: A Value-Driven Approach to Business Intelligence and Data Warehousing. By K. W. Collier. Addison-Wesley. 2012.

[Coombs, Renear, DeRose, 1987] Markup Systems and the Future of Scholarly Text Processing. By J. H. Coombs, A. H. Renear, S. J. DeRose. Communications of the ACM. Volume 30. Number 11. 1987. Pages 933-947.

[Cooper, Reimann, Cronin, 2007] About Face 3: The Essentials of Interaction Design. By A. Cooper, R. Reimann, D. Cronin. John Wiley and Sons. 2007.

[Constantine, Lockwood, 1999] Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design. By L. L. Constantine, L. A. D. Lockwood. Addison-Wesley. 1999.

[Coplien, 1994] A Generative Development-Process Pattern Language. By J. O. Coplien. The First Conference on Pattern Languages of Programs (PLoP 1994). Monticello, U.S.A. August 4-6, 1994.

[Coplien, Harrison, 2005] Organizational Patterns of Agile Software Development. By J. O. Coplien, N. B. Harrison. Prentice-Hall. 2005.

[Correia, Ferreira, Flores, Aguiar, 2009] Patterns for Consistent Software Documentation. By F. Correia, H. Ferreira, N. Flores, A. Aguiar. The Sixteenth Conference on Pattern Languages of Programs (PLoP 2009). Chicago, U.S.A. August 28-30, 2009.

[Coughlan, Macredie, 2002] Effective Communication in Requirements Elicitation: A Comparison of Methodologies. By J. Coughlan, R. D. Macredie. Requirements Engineering. Volume 7. Number 2. 2002. Pages 47-60.

[Courage, Baxter, 2005] Understanding Your Users: A Practical Guide to User Requirements Methods, Tools, and Techniques. By C. Courage, K. Baxter. Elsevier. 2005.

[Decker, Ras, Rech, Jaubert, Rieth, 2007] Wiki-Based Stakeholder Participation in Requirements Engineering. By B. Decker, E. Ras, J. Rech, P. Jaubert, M. Rieth. IEEE Software. Volume 24. Issue 2. 2007. Pages 28-35.

[Derby, Larsen, 2006] Agile Retrospectives: Making Good Teams Great. By E. Derby, D. Larsen. Pragmatic Bookshelf. 2006.

[DeRose, Durand, Mylonas, Renear, 1997] What is Text, Really? By S. J. DeRose, D. G. Durand, E. Mylonas, A. H. Renear. Journal of Computer Documentation. Volume 21. Issue 3. 1997. Pages 1-24.

[DeRose, Maler, Orchard, Walsh, 2010] XML Linking Language (XLink) Version 1.1. By S. DeRose, E. Maler, D. Orchard, N. Walsh. (Editors). World Wide Web Consortium (W3C) Recommendation. May 6, 2010.

[Dey, 2001] Understanding and Using Context. By A. K. Dey. Personal and Ubiquitous Computing. Volume 5. Issue 1. 2001. Pages 4-7.

[Diaz, Garbajosa, Calvo-Manzano, 2009] Mapping CMMI Level 2 to Scrum Practices: An Experience Report. By J. Diaz, J. Garbajosa, J. A. Calvo-Manzano. The Sixteenth European Conference on Software Process Improvement (EuroSPI 2009). Madrid, Spain. September 2-4, 2009.

[Dromey, 1996] Cornering the Chimera. By R. G. Dromey. IEEE Software. Volume 13. Number 1. 1996. Pages 33-43.

[Dubost, Rosenthal, Hazaël-Massieux, Henderson, 2005] QA Framework: Specification Guidelines. By K. Dubost, L. Rosenthal, D. Hazaël-Massieux, L. Henderson (Editors). World Wide Web Consortium (W3C) Recommendation. August 17, 2005.

[El-Attar, 2009] Improving the Quality of Use Case Models and their Utilization in Software Development. By M. El-Attar. Ph.D. Thesis. The University of Alberta. Edmonton, Canada. 2009.

[Fancott, Kamthan, Shahmir, 2011] Using the Social Web for Teaching and Learning User Stories. By T. Fancott, P. Kamthan, N. Shahmir. The Sixth International Conference on e-Learning (ICEL 2011). Kelowna, Canada. June 27-28, 2011.

[Fehlmann, Santillo, 2010] From Story Points to COSMIC Function Points in Agile Software Development – A Six Sigma Perspective. By T. Fehlmann, L. Santillo. The Twentieth International Workshop on Software Measurement (IWSM 2010). Stuttgart, Germany. November 10-12, 2010.

[Fenton, Bieman, 2012] Software Metrics: A Rigorous and Practical Approach. By N. Fenton, J. Bieman. Third Edition. CRC Press. 2012.

[Fenton, Pfleeger, 1997] Software Metrics: A Rigorous and Practical Approach. By N. E. Fenton, S. L. Pfleeger. International Thomson Computer Press. 1997.

[Firesmith, 2003] Modern Requirements Specification. By D. G. Firesmith. Journal of Object Technology. Volume 2. Number 2. 2003. Pages 53-64.

[Firtman, 2010] Programming the Mobile Web. By M. Firtman. O'Reilly Media. 2010.

[Freeman, Harrison, Wicks, Parmar, Colle, 2010] Stakeholder Theory: The State of the Art. By R. E. Freeman, J. S. Harrison, A. C. Wicks, B. L. Parmar, S. de Colle. Cambridge University Press. 2010.

[Gallardo-Valencia, Olivera, Sim, 2007] Are Use Cases Beneficial for Developers Using Agile Requirements? By R. E. Gallardo-Valencia, V. Olivera, S. E. Sim. The Fifth International Workshop on Comparative Evaluation in Requirements Engineering (CERE 2007). New Delhi, India. October 16, 2007.

[Garvin, 1984] What does Product Quality Really Mean? By D. A. Garvin. MIT Sloan Management Review. Volume 26. Number 1. 1984. Pages 25-43.

[Ghezzi, Jazayeri, Mandrioli, 2003] Fundamentals of Software Engineering. By C. Ghezzi, M. Jazayeri, D. Mandrioli. Second Edition. Prentice-Hall. 2003.

[Goldfarb, 1981] A Generalized Approach to Document Markup. By C. F. Goldfarb. The ACM SIGPLAN SIGOA Symposium on Text Manipulation. Portland, U.S.A. June 8-10, 1981.

[Harold, 2003] Effective XML. By E. R. Harold. Addison-Wesley. 2003.

[Harvey, 1988] The Abilene Paradox and Other Meditations on Management. By J. B. Harvey. Jossey-Bass. 1988.

[Highsmith, 2002] Agile Software Development Ecosystems. By J. Highsmith. Addison-Wesley. 2002.

[Highsmith, 2009] Agile Project Management: Creating Innovative Products. By J. Highsmith. Addison-Wesley. 2009.

[Hiranabe, 2007] Visualizing Agile Projects with Kanban Boards. By K. Hiranabe. Dr. Dobb's. September 20, 2007.

[Hoda, Kruchten, Noble, Marshall, 2010] Agility in Context. By R. Hoda, P. Kruchten, J. Noble, S. Marshall. The Twenty Fifth Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2010). Reno/Tahoe, U.S.A. October 17-21, 2010.

[Hoover, Rosso-Llopart, Taran, 2009] Evaluating Project Decisions: Case Studies in Software Engineering. By C. L. Hoover, M. Rosso-Llopart, G. Taran. Addison-Wesley. 2009.

[Holtzblatt, Beyer, 1995] Requirements Gathering: The Human Factor. By K. Holtzblatt, H. R. Beyer. Communications of the ACM. Volume 38. Issue 5. 1995. Pages 31-32.

[Hull, Jackson, Dick, 2011] Requirements Engineering. By E. Hull, K. Jackson, J. Dick. Springer-Verlag. 2011.

[Hussain, Kosseim, Ormandjieva, 2010] Towards Approximating COSMIC Functional Size from User Requirements in Agile Development Processes Using Text Mining. By I. Hussain, L. Kosseim, O. Ormandjieva. The Fifteenth International

Conference on Applications of Natural Language to Data Bases (NLDB 2010). Cardiff, U.K. June 23-25, 2010.

[IEEE, 1990] IEEE Standard 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE Computer Society. 1990.

[IEEE, 1995] IEEE Standard 730.1-1995. IEEE Guide for Software Quality Assurance Planning. IEEE Computer Society. 1995.

[IEEE, 1998] IEEE Standard 830-1998. Recommended Practice for Software Requirements Specifications. IEEE Computer Society. 1998.

[IEEE, 2005] IEEE Standard 1012-2004. IEEE Standard for Software Verification and Validation. IEEE Computer Society. 2005.

[IIBA, 2009] A Guide to the Business Analysis Body of Knowledge, Version 2.0. By International Institute of Business Analysis (IIBA). International Institute of Business Analysis. 2009.

[Issa, Odeh, Coward, 2006] Using Use Case Patterns to Estimate Reusability in Software Systems. By A. Issa, M. Odeh, D. Coward. Information and Software Technology. Volume 48. Issue 9. 2006. Pages 836-845.

[ISO, 1986] ISO 8879:1986. Information Processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML). International Organization for Standardization (ISO). 1986.

[ISO, 1998a] ISO 9241-11:1998. Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) Part 11: Guidance on Usability. The International Organization for Standardization (ISO). 1998.

[ISO, 1998b] ISO 9241-12:1998. Ergonomic Requirements for Office Work with Visual Display Terminals Part 12: Presentation of Information. International Organization for Standardization (ISO). 1998.

[ISO, 2000] ISO 8601:2000. Data Elements and Interchange Formats -- Information Interchange -- Representation of Dates and Times. The International Organization for Standardization (ISO). 2000.

[ISO, 2008] ISO 9241-171:2008. Ergonomics of Human-System Interaction -- Part 171: Guidance on Software Accessibility. The International Organization for Standardization (ISO). 2008.

[ISO/IEC, 2001] ISO/IEC 9126-1:2001(E). Software Engineering -- Product Quality -- Part 1: Quality Model. The International Organization for Standardization (ISO)/The International Electrotechnical Commission (IEC). 2001.

[ISO/IEC, 2006a] ISO/IEC 19757-3:2006. Information Technology -- Document Schema Definition Language (DSDL) -- Part 3: Rule-Based Validation -- Schematron. The International Organization for Standardization (ISO)/The International Electrotechnical Commission (IEC). 2006.

[ISO/IEC, 2006b] ISO/IEC 19757-4:2006. Information Technology -- Document Schema Definition Language (DSDL) -- Part 4: Namespace-based Validation Dispatching Language (NVDL). The International Organization for Standardization (ISO)/The International Electrotechnical Commission (IEC). 2006.

[ISO/IEC, 2008] ISO/IEC 19757-2:2008. Information Technology -- Document Schema Definition Language (DSDL) -- Part 2: Regular-Grammar-Based Validation --

RELAX NG. The International Organization for Standardization (ISO)/The International Electrotechnical Commission (IEC). 2008.

[Jacobs, Walsh, 2004] Architecture of the World Wide Web, Volume One. By I. Jacobs, N. Walsh (Editors). World Wide Web Consortium (W3C) Recommendation. December 15, 2004.

[Jacobson, Christerson, Jonsson, Övergaard, 1992] Object-Oriented Software Engineering: A Use Case Driven Approach. By I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard. Addison-Wesley. 1992.

[Jeffries, 2001] Essential XP: Card, Conversation, and Confirmation. By R. Jeffries. XP Magazine. August 30, 2001.

[Jelliffe, 1998] The XML and SGML Cookbook: Recipes for Structured Information. By R. Jelliffe. Prentice-Hall. 1998.

[Jokela, Abrahamsson, 2004] Usability Assessment of an Extreme Programming Project: Close Co-operation with the Customer Does Not Equal to Good Usability. By T. Jokela, P. Abrahamsson. The Fifth International Conference on Product Focused Software Process Improvement (PROFES 2004). Kausai Science City, Japan. April 5-8, 2004.

[Jones, Bonsignour, 2012] The Economics of Software Quality. By C. Jones, O. Bonsignour. Addison-Wesley. 2012.

[Jung, 1971] Psychological Types. By C. G. Jung. Princeton University Press. 1971.

[Junior, Filgueiras, 2005] User Modeling with Personas. By P. T. A. Junior, L. V. L. Filgueiras. The 2005 Latin American Conference on Human-Computer Interaction (CLIHC 2005). Cuernavaca, Mexico. October 23-26, 2005.

[Juristo, Moreno, 2001] Basics of Software Engineering Experimentation. By N. Juristo, A. M. Moreno. Kluwer Academic Publishers. 2001.

[Kähkönen, Abrahamsson, 2004] Achieving CMMI Level 2 with Enhanced Extreme Programming Approach. By T. Kähkönen, P. Abrahamsson. The Fifth International Conference on Product Focused Software Process Improvement (PROFES 2004). Kausai Science City, Japan. April 5-8, 2004.

[Kaindl, Svetinovic, 2010] On Confusion between Requirements and their Representations. By H. Kaindl, D. Svetinovic. Requirements Engineering. Volume 15. Number 3. 2010. Pages 307-311.

[Kaminski, Perry, 2007] Open Source Software Licensing Patterns. By H. Kaminski, M. Perry. The Sixth Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP 2007). Porto de Galinhas, Brazil. May 27-30, 2007.

[Kamsties, Berry, Krieger, 2003] From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity, A Handbook. By E. Kamsties, D. M. Berry, M. M. Krieger. University of Waterloo. Waterloo, Canada. 2003.

[Kamthan, 2008] Towards High-Quality Mobile Applications by a Systematic Integration of Patterns. By P. Kamthan. Journal of Mobile Multimedia. Special Issue Theme: Engineering Mobile and Context-Sensitive Applications. Volume 4. Number 3/4. 2008. Pages 165-184.

[Kamthan, 2009] A Methodology for Integrating the Social Web Environment in Software Engineering Education. By P. Kamthan. International Journal of Information and Communication Technology Education. Volume 5. Number 2. 2009. Pages 21-35.

[Kamthan, 2010] A Viewpoint-Based Approach for Understanding the Morphogenesis of Patterns. By P. Kamthan. International Journal of Knowledge Management. Volume 6. Issue 2. 2010. Pages 40-65.

[Kamthan, 2011a] Towards Understanding the Use of Patterns in Software Engineering. By P. Kamthan. In: Knowledge Engineering for Software Development Life Cycles: Support Technologies and Applications. M. Ramachandran (Editor). IGI Global. 2011. Pages 115-135.

[Kamthan, 2011b] Implications of Markup on the Description of Software Patterns. By P. Kamthan. In: Knowledge Engineering for Software Development Life Cycles: Support Technologies and Applications. M. Ramachandran (Editor). IGI Global. 2011. Pages 136-160.

[Kamthan, 2011c] Using the Social Web Environment for Software Engineering Education. By P. Kamthan. In: Online Courses and ICT in Education: Emerging Practices and Applications. L. A. Tomei (Editor). IGI Global. 2011. Pages 23-45.

[Kamthan, 2011d] An Exploration of the Social Web Environment for Collaborative Software Engineering Education. By P. Kamthan. International Journal of Web-based Learning and Teaching Technologies. Volume 6. Issue 2. 2011. Pages 18-39.

[Kamthan, Fancott, 2011] A Knowledge Management Model for Patterns. By P. Kamthan, T. Fancott. In: Encyclopedia of Knowledge Management. D. G. Schwartz, D. Te'eni (Editors). Second Edition. IGI Global. 2011. Pages 694-703.

[Kamthan, Shahmir, 2010] Towards an Understanding of the User Story Environment. By P. Kamthan, N. Shahmir. The Fifteenth IBIMA Conference on Knowledge

Management and Innovation: A Business Competitive Edge Perspective. Cairo, Egypt. November 6-7, 2010.

[Karner, 1993] Use Case Points – Resource Estimation for Objectory Projects. By G. Karner. Objective Systems SF AB (Copyright Owner Rational Software). 1993.

[Kay, 2007] XSL Transformations (XSLT) Version 2.0. By M. Kay (Editor). World Wide Web Consortium (W3C) Recommendation. January 23, 2007.

[Kay, 2008] XSLT 2.0 and XPath 2.0 Programmer's Reference. By M. Kay. Fourth Edition. Wrox Press. 2008.

[Keith, 2010] Agile Game Development with Scrum. By C. Keith. Addison-Wesley. 2010.

[Kiyavitskaya, Zeni, Mich, Berry, 2008] Requirements for Tools for Ambiguity Identification and Measurement in Natural Language Requirements Specifications. By N. Kiyavitskaya, N. Zeni, L. Mich, D. M. Berry. Requirements Engineering. Volume 13. Issue 3. 2008. Pages 207-239.

[Keller, 1990] A Guide to Natural Naming. By D. Keller. ACM SIGPLAN Notices. Volume 25. Issue 5. 1990. Pages 95-102.

[Kitchenham, Charters, 2007] Guidelines for performing Systematic Literature Reviews in Software Engineering. By B. A. Kitchenham, S. Charters. Version 2.3. EBSE Technical Report EBSE-2007-01. Keele University. Keele, U.K. 2007.

[Kovitz, 2003] Hidden Skills that Support Phased and Agile Requirements Engineering. By B. Kovitz. Requirements Engineering. Volume 8. Number 2. 2003. Page 135-141.

[Langr, Ottinger, 2011] Agile in a Flash: Speed-Learning Agile Software Development. By J. Langr, T. Ottinger. Pragmatic Bookshelf. 2011.

[Larman, Basili, 2003] Iterative and Incremental Developments: A Brief History. By C. Larman, V. R. Basili. Computer. Volume 36. Number 6. 2003. Pages 47-56.

[Larman, Vodde, 2010] Practices for Scaling Lean and Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum. By C. Larman, B. Vodde. Addison-Wesley. 2010.

[Leffingwell, 2011] Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. By D. Leffingwell. Addison-Wesley. 2011.

[Lewitz, 2004] Story Management. By O. Lewitz. The Fifth International Conference on Extreme Programming and Agile Processes in Software Engineering (XP 2004). Garmisch-Partenkirchen, Germany. June 6-10, 2004.

[Li, 2005] A Framework for the Requirements Engineering Process Development. By J. Li. Ph. D. Thesis. The University of Calgary. Calgary, Canada. 2005.

[Linton, 2007] Beyond Schemas: Planning Your XML Model. By J. Linton. O'Reilly Media. 2007.

[Louridas, 2006] Using Wikis in Software Development. By P. Louridas. IEEE Software. Volume 23. Issue 2. 2006. Pages 88-91.

[Lui, Chan, 2008] Software Development Rhythms: Harmonizing Agile Practices for Synergy. By K. M. Lui, K. C. C. Chan. John Wiley and Sons. 2008.

[Madeyski, 2010] Test-Driven Development: An Empirical Evaluation of Agile Practice. By L. Madeyski. Springer-Verlag. 2010.

[Maguire, Kirakowski, Vereker, 1998] RESPECT: User-Centred Requirements Handbook. By M. C. Maguire, J. Kirakowski, N. Vereker. European Commission Telematics Applications Programme (EC-TAP). Requirements Engineering and Specification in Telematics (RESPECT) Project TE 2010. WP5 Deliverable D5.3. Version 3.3. RESPECT Consortium. July 16, 1998.

[Maler, El Andaloussi, 1996] Developing SGML DTDs: From Text to Model to Markup. By E. Maler, J. El Andaloussi. Prentice-Hall. 1996.

[Marsh, Tobin, 2009] XML Base (Second Edition). By J. Marsh, R. Tobin (Editors). World Wide Web Consortium (W3C) Recommendation. January 28, 2009.

[Marsh, Veillard, Walsh, 2005] xml:id Version 1.0. By J. Marsh, D. Veillard, N. Walsh (Editors). World Wide Web Consortium (W3C) Recommendation. September 9, 2005.

[Martin, Melnik, 2008] Tests and Requirements, Requirements and Tests: A Möbius Strip. By R. C. Martin, G. Melnik. Volume 25. Number 1. 2008. Pages 54-59.

[McAvoy, Butler, 2006] Resisting the Change to User Stories: A Trip to Abilene. By J. McAvoy, T. Butler. International Journal of Information Systems and Change Management. Volume 1. Number 1. 2006. Pages 48-61.

[McCarron, 2010] XHTML^{TM} Basic 1.1 – Second Edition. By S. McCarron (Editor). World Wide Web Consortium (W3C) Recommendation. November 23, 2010.

[McInerney, Koenig, 2011] Knowledge Management (KM) Processes in Organizations: Theoretical Foundations and Practice. By C. R. McInerney, M. E. D. Koenig. Morgan and Claypool. 2011.

[McKay, 1999] Developing User Interfaces for Microsoft Windows. By E. N. McKay. Microsoft Press. 1999.

[McManus, 2004] Managing Stakeholders in Software Development Projects. By J. McManus. Butterworth-Heinemann. 2004.

[Megginson, 1998] Structuring XML Documents. By D. Megginson. Prentice-Hall. 1998.

[Megginson, 2005] Imperfect XML: Rants, Raves, Tips, and Tricks … from an Insider. By D. Megginson. Addison-Wesley. 2005.

[Mernik, Heering, Sloane, 2005] When and How to Develop Domain-Specific Languages. By M. Mernik, J. Heering, A. M. Sloane. ACM Computing Surveys. Volume 37. Issue 4. 2005. Pages 316-344.

[Meszaros, Doble, 1998] A Pattern Language for Pattern Writing. By G. Meszaros, J. Doble. In: Pattern Languages of Program Design 3. R. C. Martin, D. Riehle, F. Buschmann (Editors). Addison-Wesley. 1998. Pages 529-574.

[Minocha, Petre, Roberts, 2008] Using Wikis to Simulate Distributed Requirements Development in a Software Engineering Course. By S. Minocha, M. Petre, D. Roberts. International Journal of Engineering Education. Volume 24. Number 4. 2008. Pages 689-704.

[Mohagheghi, Anda, Conradi, 2005] Effort Estimation of Use Cases for Incremental Large-Scale Software Development. By P. Mohagheghi, B. Anda, R. Conradi. The 2005 International Conference on Software Engineering (ICSE 2005). St. Louis, U.S.A. May 15-21, 2005.

[Mohammadi, Nikkhahan, Sohrabi, 2009] Challenges of User Involvement in Extreme Programming Projects. By S. Mohammadi, B. Nikkhahan, S. Sohrabi. International

Journal of Software Engineering and Its Applications. Volume 3. Number 1. 2009. Pages 19-32.

[Monochristou, Vlachopoulou, 2007] Requirements Specification using User Stories. By V. Monochristou, M. Vlachopoulou. In: Agile Software Development Quality Assurance. I. G. Stamelos, P. Sfetsos (Editors). Idea Group. 2007. Pages 71-89.

[Morville, 2005] Ambient Findability: What We Find Changes Who We Become. By P. Morville. O'Reilly Media. 2005.

[Morville, Rosenfeld, 2006] Information Architecture for the World Wide Web. By P. Morville, L. Rosenfeld. Third Edition. O'Reilly Media. 2006.

[Murata, Lee, Mani, Kawaguchi, 2005] Taxonomy of XML Schema Languages Using Formal Language Theory. By M. Murata, D. Lee, M. Mani, K. Kawaguchi. ACM Transactions on Internet Technology. Volume 5. Number 4. 2005. Pages 660–704.

[Murata, St. Laurent, Kohn, 2001] XML Media Types. By M. Murata, S. St. Laurent, D. Kohn. Request for Comments (RFC) 3023. The Internet Society. 2001.

[Nielsen, 1993] Usability Engineering. By J. Nielsen. Morgan Kaufmann. 1993.

[Nuseibeh, 1996] Conflicting Requirements: When the Customer is Not Always Right. By B. Nuseibeh. Requirements Engineering. Volume 1. Number 1. 1996. Pages 70-71.

[Nuseibeh, Easterbrook, 2000] Requirements Engineering: A Roadmap. By B. Nuseibeh, S. Easterbrook. The Twenty Second International Conference on Software Engineering (ICSE 2000). Limerick, Ireland. June 4-11, 2000.

[O'Reilly, 2005] What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. By T. O'Reilly. O'Reilly Network. September 30, 2005.

[Övergaard, Palmkvist, 2005] Use Cases: Patterns and Blueprints. By G. Övergaard, K. Palmkvist. Addison-Wesley. 2005.

[Paetsch, 2003] Requirements Engineering in Agile Software Development. By F. Paetsch. Diploma Thesis. Hochschule Mannheim. Mannheim, Germany. 2003.

[Palfrey, Gasser, 2008] Born Digital: Understanding the First Generation of Digital Natives. By J. Palfrey, U. Gasser. Basic Books. 2008.

[Park, Maurer, 2008] The Requirements Abstraction in User Stories and Executable Acceptance Tests. By S. Park, F. Maurer. Agile Conference 2008. Toronto, Canada. August 4-8, 2008.

[Parnas, Clements, 1986] A Rational Design Process: How and Why to Fake It. By D. L. Parnas, P. C. Clements. IEEE Transactions on Software Engineering. Volume 12. Issue 2. 1986. Pages 251-257.

[Patel, Ramachandran, 2009a] Story Card Based Agile Software Development. By C. Patel, M. Ramachandran. International Journal of Hybrid Information Technology. Volume 2. Number 2. 2009. Pages 125-140.

[Patel, Ramachandran, 2009b] Story Card Maturity Model (SMM): A Process Improvement Framework for Agile Requirements Engineering Practices. By C. Patel, M. Ramachandran. Journal of Software. Volume 4. Number 5. 2009. Pages 422-435.

[Peeters, 2005] Agile Security Requirements Engineering. By J. Peeters. The Symposium on Requirements Engineering for Information Security (SREIS 2005). Paris, France. August 29, 2005.

[Perry, Kaminski, 2005] A Pattern Language of Software Licensing. By M. Perry, H. Kaminski. The Tenth European Conference on Pattern Languages of Programs (EuroPLoP 2005). Irsee, Germany. July 6-10, 2005.

[Phalp, Vincent, Cox, 2007] Assessing the Quality in Use Case Descriptions. By K. T. Phalp, J. Vincent, K. Cox. Software Quality Journal. Volume 15. Number 1. 2007. Pages 69-97.

[Pham, Pham, 2012] Scrum in Action: Agile Software Project Management and Development. By A. Pham, P.-V. Pham. Cengage Learning. 2012.

[Phillips, Davis, 2009] Tags for Identifying Languages. By A. Phillips, M. Davis. Request for Comments (RFC) 5646. The Internet Society. 2009.

[Piaget, 1952] The Origins of Intelligence in Children. By J. Piaget. International University Press. 1952.

[Pirolli, 2007] Information Foraging Theory: Adaptive Interaction with Information. By P. Pirolli. Oxford University Press. 2007.

[PMI, 2008] A Guide to the Project Management Body of Knowledge. By Project Management Institute (PMI). Fourth Edition. Project Management Institute. 2008.

[Polanyi, 1966] The Tacit Dimension. By M. Polanyi. Doubleday and Company. 1966.

[Power, 2010] Stakeholder Identification in Agile Software Product Development Organizations: A Model for Understanding Who and What Really Counts. By K. Power. Agile 2010 Conference. Orlando, U.S.A. August 9-13, 2010.

[Pugh, 2011] Lean-Agile Acceptance Test-Driven Development: Better Software Through Collaboration. By K. Pugh. Addison-Wesley. 2011.

[Rees, 2002] A Feasible User Story Tool for Agile Software Development. By M. J. Rees. The Ninth Asia-Pacific Software Engineering Conference (APSEC 2002). Gold Coast, Australia. December 4-6, 2002.

[Robinson, Pawlowski, Volkov, 2003] Requirements Interaction Management. By W. N. Robinson, S. D. Pawlowski, V. Volkov. ACM Computing Surveys. Volume 35. Issue 2. 2003. Pages 132-190.

[Rodríguez, Yagüe, Alarcón, Garbajosa, 2009] Some Findings Concerning Requirements in Agile Methodologies. By P. Rodríguez, A. Yagüe, P. P. Alarcón, J. Garbajosa. The Tenth International Conference on Product-Focused Software Process Improvement (PROFES 2009). Oulu, Finland. June 15-17, 2009.

[Rosen, 2004] Open Source Licensing: Software Freedom and Intellectual Property Law. By L. Rosen. Prentice-Hall. 2004.

[Rule, 2009] Sizing Agile User Stories with COSMIC. By P. G. Rule. COSMICON. 2009.

[Rüping, 2003] Agile Documentation: A Pattern Guide to Producing Lightweight Documents for Software Projects. By A. Rüping. John Wiley and Sons. 2003.

[Schuler, Namioka, 1993] Participatory Design: Principles and Practices. By D. Schuler, A. Namioka (Editors). CRC Press. 1993.

[Schwaber, 2004] Agile Project Management with Scrum. By K. Schwaber. Microsoft Press. 2004.

[Schwaber, Beedle, 2002] Agile Software Development with Scrum. By K. Schwaber, M. Beedle. Prentice-Hall. 2002.

[Sharp, Galal, Finkelstein, 1999] Stakeholder Identification in the Requirements Engineering Process. By H. Sharp, G. H. Galal, A. Finkelstein. The Tenth International Conference and Workshop on Database and Expert Systems Applications (DEXA 1999). Florence, Italy. August 30-September 3, 1999.

[Singh, 2008] U-SCRUM: An Agile Methodology for Promoting Usability. By M. Singh. Agile 2008 Conference. Toronto, Canada. August 4-8, 2008.

[Sliger, Broderick, 2008] The Software Project Manager's Bridge to Agility. By M. Sliger, S. Broderick. Addison-Wesley. 2008.

[Šmite, Moe, Ågerfalk, 2010] Agility Across Time and Space: Implementing Agile Methods in Global Software Projects. By D. Šmite, N. B. Moe, P. J. Ågerfalk. Springer-Verlag. 2010.

[Soundararajan, 2008] Agile Requirements Generation Model: A Soft-structured Approach to Agile Requirements Engineering. By S. Soundararajan. M. Sc. Thesis. Virginia Polytechnic Institute and State University. Blacksburg, U.S.A. 2008.

[Spinellis, 2007] On Paper. By D. Spinellis. IEEE Software. Volume 24. Issue 6. 2007. Pages 24-25.

[Stapleton, 2003] DSDM: Business Focused Development. By J. Stapleton. Addison-Wesley. 2003.

[St. Laurent, 2004] Open Source and Free Software Licensing. By A. M. St. Laurent. O'Reilly Media. 2004.

[Stober, Hansmann, 2010] Agile Software Development Best Practices for Large Software Development Projects. By T. Stober, U. Hansmann. Springer-Verlag. 2010.

[Sy, 2007] Adapting Usability Investigations for Agile User-Centered Design. By D. Sy. Journal of Usability Studies. Volume 2. Issue 3. 2007. Pages 112-132.

[The Standish Group, 1995] Chaos. The Standish Group Report. The Standish Group. 1995.

[The Unicode Consortium, 2007] The Unicode Standard, Version 5.0.0. By The Unicode Consortium. Addison-Wesley. 2007.

[Thomas, Bevan, 1996] Usability Context Analysis: A Practical Guide. By C. Thomas, N. Bevan. National Physical Laboratory. Teddington, U.K. 1996.

[Tidwell, 2011] Designing Interfaces: Patterns for Effective Interaction Design. By J. Tidwell. Second Edition. O'Reilly Media. 2011.

[Välimäki, Kääriäinen, 2007] Requirements Management Practices as Patterns for Distributed Product Management. By A. Välimäki, J. Kääriäinen. The Eighth International Conference on Product-Focused Software Process Improvement (PROFES 2007). Riga, Latvia. July 2-4, 2007.

[Vitali, Iorio, Gubellini, 2005] Design Patterns for Descriptive Document Substructures. By F. Vitali, A. D. Iorio, D. Gubellini. The 2005 Extreme Markup Languages Conference (EML 2005). Montreal, Canada. August 1-5, 2005.

[Vlist, 2004] RELAX NG. By E. van der Vlist. O'Reilly Media. 2004.

[Wake, 2002] Extreme Programming Explored. By W. C. Wake. Addison-Wesley. 2002.

[Wallace, Raggett, Aufgang, 2002] Extreme Programming for Web Projects. By D. Wallace, I. Raggett, J. Aufgang. Addison Wesley. 2002.

[Whitehead, 2007] Collaboration in Software Engineering: A Roadmap. By J. Whitehead. The Twenty Ninth International Conference on Software Engineering (ICSE 2007). Minneapolis, U.S.A. May 20-26, 2007.

[Wiegers, 2003] Software Requirements. By K. E. Wiegers. Second Edition. Microsoft Press. 2003.

[Williams, 2010] Agile Software Development Methodologies and Practices. By L. Williams. Advances in Computers. Volume 80. 2010. Pages 1-44.

[Winbladh, Ziv, Richardson, 2008] Surveying the Usability of Requirements Approaches using a 3-Dimensional Framework. By K. Winbladh, H. Ziv, D. J. Richardson. Institute for Software Research Technical Report UCI-ISR-08-3. 2008.

[Wohlin, Runeson, Höst, Ohlsson, Regnell, Wesslén, 2000] Experimentation in Software Engineering: An Introduction. By C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén. Kluwer Academic Publishers. 2000.

[Woit, 2005] Requirements Interaction Management in an eXtreme Programming Environment: A Case Study. By D. M. Woit. The International Conference on Software Engineering 2005 (ICSE 2005). St. Louis, U.S.A. May 15-21, 2005.

[Young, 2008] Mental Models: Aligning Strategy with Human Behavior. By I. Young. Rosenfeld Media. 2008.

[Zhang, Arvela, Berki, Muhonen, Nummenmaa, Poranen, 2010] Towards Lightweight Requirements Documentation. By Z. Zhang, M. Arvela, E. Berki, M. Muhonen, J. Nummenmaa, T. Poranen. Journal of Software Engineering and Applications. Volume 3. Number 9. 2010. Pages 882-889.