Chapter N

TRANSFORMING USABILITY ENGINEERING REQUIREMENTS INTO SOFTWARE ENGINEERING SPECIFICATIONS

From PUF to UML

Jim A. Carter, Jun Liu, Kevin Schneider, David Fourney
University of Saskatchewan, Saskatoon, Saskatchewan, Canada

Abstract: The Unified Modeling Language (**UML**) is widely used by Software Engineers as the basis of analysis and design in software development. While UML is very strong at specifying the structure and functionality of the application, it is seldom used to its potential to specify usability-related information. The Putting Usability First (**P U F**) methodology of Usability Engineering identifies and specifies usability-related information. This chapter discusses how requirements and other contextual information from the PUF methodology can be transformed into UML in order to specify the context information of the application to ensure the usability of the application.

Key words: integration, requirements, software engineering, specifications, unified usability engineering

1. INTRODUCTION

While the need for integrating human factors with software engineering [4, 12] has been recognized for over a decade, the reality has yet to happen to any realistic extent. Attempts to integrate human-computer interaction / usability engineering with software engineering rely on their acceptance by software engineers, who control most development projects. This is largely dependent on the impact of any proposed additions to the current software engineering practice.

This has not taken place in the process realm, where the 32 software engineering processes defined in ISO TR 15504 [18] failed to include any human-computer interaction processes. To meet this omission, software ergonomists developed ISO TR 18529 [15] which defined 43 additional human-system life cycle processes. Considering that ISO TR 15504 expects each of these processes to be evaluated in terms of 26 generic practices, this could result in 1,950 sub-processes.

The goal of integration is to improve the resulting system. Rather than focus on processes, our approach involves integrating the documentation used to develop this resulting system.

Our starting point is a set of usability engineering requirements developed by the Putting Usability First (PUF) methodology [3]. PUF is a user-centered approach to systems development. It identifies and structures a use model based on an interrelated set of task, user, content, tool and scenario descriptions. These descriptions provide a context of use description, as recommended by ISO 13407 Human Centered Design Processes for Interactive Systems [16] that can easily be used to integrate usability concerns within other software development activities.

Our target is a set of software engineering specifications expressed by the Unified Modeling Language (UML) [1], which are currently pervasive throughout major software developments. By assisting in developing UML specifications, it is anticipated that PUF can gain greater acceptance from software engineers than previous usability engineering methodologies. Applying the PUF methodology in UML can ensure the application is developed in a context rich information environment that minimizes the occurrence of usability problems.

The transformations of usability requirements to software engineering specifications, which we identify in this paper, allow usability engineers and software engineers to perform their own processes in their own manners while being able to better integrate their efforts.

2. THE PUTTING USABILITY FIRST (PUF) METHODOLOGY

Putting Usability First (PUF) is a usability engineering methodology that has evolved from previous work on Multi-Oriented Task Analysis (MOST)

[4, 5]. It has been applied to a variety of application areas including: e-Commerce [2] and educational multimedia [6].

The concept behind PUF is that for a usability engineering methodology to succeed, it must be usable by developers as well as result in a usable system for end users. ISO 9241-11 Guidance on Usability [17] defines usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". PUF involves a thorough consideration of usability issues in each activity of the development life cycle. Being thorough does not require following a single formalized highly prescriptive approach to usability. Rather, it recognizes all development decisions should be based on usability evaluations. These usability evaluations provide qualitative and quantitative information that can guide the development process.

Effectiveness is defined as, "the accuracy and completeness with which users achieve specified goals" [17]. PUF structures requirements to assist developers in accurately and completely identifying specific user groups, tasks, content types, tools and scenarios that provide the context of use for the new system being developed.

Efficiency is defined as, "the resources expended in relation to the accuracy and completeness with which users achieve goals" [17]. The PUF methodology recognizes the usefulness of a number of usability methods and allows developers the flexibility to choose those methods that are most efficient within a particular context of use. PUF provides the developers with guidance and flexibility in choosing between the usability methods identified in ISO 16982 Usability Methods Supporting Human Centered Design [14] that may be used to identify usability requirements, including user, task, environment, and system related characteristics and/or to evaluate a system or a model of a system to assess usability characteristics.

Satisfaction is defined as, "positive attitudes to the use of the product and freedom from discomfort in using it" [17]. PUF enhances satisfaction for developers by supporting the integration of usability engineering and software engineering activities and specifications.

PUF recognizes that for many applications it is impractical to try and develop the perfect system all at once. Development is often spread across a series of different releases. PUF supports release-based development in its inclusion of possibilities in its analysis of the environment of current

development. Iteration is crucial in allowing release-based development to respond to changing needs as well as to needs previously identified. This iteration involves ongoing cycles of analysis, design, and evaluation within each of the development activities identified in PUF.

2.1 Major Processes within PUF

PUF focuses on four major life cycle processes that work cooperatively towards the development of a system. These processes include: possibilities analysis, requirements analysis, design, and implementation. It is notable that testing is not considered a separate process in PUF. That is because PUF considers evaluation, which is broader than traditional testing, to be an integral part of each of these other activities. Dealing with it in this manner ensures that it is performed when it is most effective and that it is significant in determining the usability of the resulting system.

Possibilities analysis attempts to identify and briefly describe all the main scenarios, tasks, user groups, content chunks, and tools related to the intended application system for all its potential releases. Evaluation of existing possibilities plays an important role in identifying further possibilities beyond those currently existing or obvious. Possibilities analysis starts with recording narratives of existing scenarios and moves to develop records describing each possibility. Possibilities analysis involves: identifying possibilities, identifying relationships between different types of possibilities, and identifying environmental factors influencing each possibility. Possibility records describe the current and future environment for the system being developed. Possibilities analysis is far broader and more comprehensive than initial investigations typically performed by software engineering. This comprehensiveness is essential in establishing a user-centered context of use both for usability engineers and for software engineers. While it is hoped that this work leads to a PUF requirements analysis, even this initial activity can significantly improve the usability of the resulting system.

A PUF requirements analysis expands the understanding of those possibilities that have been selected as the basis for the development of the current release. It evolves their possibility records into requirement records by adding specific usability-related information and requirements. The set of additional information and requirements is based on the particular type of possibility being analyzed. Requirements analysis in PUF focuses to a greater extent on usability requirements and to a lesser extent on technical

requirements than requirements analysis typical in software engineering life cycles.

Because of the considerable overlap, it would be ideal for software engineers to make use of a PUF requirements analysis as a starting point for adding technical requirements. This can benefit software engineers by reducing the amount of work they need to do, especially in the area of gathering technical requirements. This, in turn, benefits usability engineers and end users by ensuring that usability requirements are part of future development decisions.

A PUF design focuses on new tools, scenarios and interactions that can be added to the current environment. Whereas the previous activities can easily be conducted by usability engineers apart from their software engineering colleagues, it is more likely that software engineers will be involved and often in charge of major design activities. PUF requirements need to be integrated with other software engineering requirements to ensure usability is properly considered in design.

Regardless of whether interface design is allocated to usability engineers or is included within the main design activity, PUF can provide assistance in identifying and evaluating usability issues related to this design. Design specifications recorded as or translated into PUF records combined with existing PUF records specify a use model [25] that can be subject to evaluation prior to being implemented. These new records also provide an up-to-date context for the development of future releases.

Implementation is usually in the hands of software engineers. To ensure that usability engineering requirements will be considered in implementation, it is essential that these requirements be included within the formal specifications being used to construct and test the resulting system.

2.2 Possibility Types within PUF

The PUF methodology identifies and structures requirements based on an interrelated set of five types of possibilities: tasks, users, content, tools and scenarios, as illustrated in Figure 1. The combination of these records meets all the requirements of ISO 13407 for consideration of users, goals, and the environment. The user records in PUF specify information about the users' characteristics. The task records in PUF specify information about the users'

goals. All the records and their linkage information and environmental information identify the context of the user.

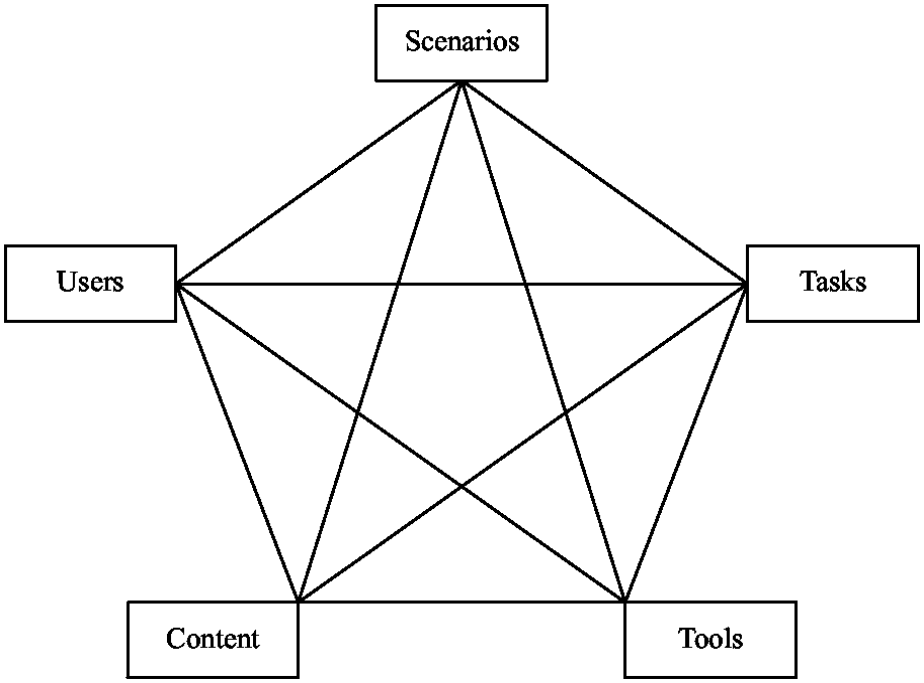


Figure 1. The five foci of PUF specifications

Users are not all the same and thus it is important to understand the characteristics of different user groups. Severe usability problems can occur in systems designed for a "generic" user who seldom exists. Users, while of penultimate importance, are only users if they use the system and thus are closely linked with the tasks that each of groups of users performs.

Tasks are specific accomplishments of one or more individuals in a group of users. The degree of accomplishment of a task is generally more important than the method of achieving it. Thus each of the users should be allowed to select the methods which are most usable for them. Tasks are the basis for individuals becoming users. This analysis of tasks should not be limited to only those tasks that are currently considered to be part of what an application should accomplish. The analysis of tasks should be expanded to include similar tasks and other potential tasks that may not be currently performed.

Content is the material processed by computer systems. Data can be presented in a variety of formats and can be processed to higher levels such as information and knowledge. Content serves the users accomplishing their desired tasks, and should be kept subservient to both users and tasks. Considerable usability problems can arise from structuring applications around their content rather than around how this content will be used. The content oriented "Field of Dreams" syndrome of "if you build it, they will come" (that is especially prevalent in the design of Web sites but also exists with many other applications) puts the ego of the developer ahead of the needs of the potential users.

Tools are any of the many things (computerized or non-computerized) that help a person accomplish some task (or set of tasks). Both developers and end users need and use tools. Developers use their tools to create or modify other tools (including software systems) for the end users. Different tools (or sets of tools) can be used to accomplish the same task. Tools exist at (and are designed for) various levels: from entire application systems down to individual controls within the system. Tools, like content, serve the tasks and users. Premature focusing on tools can lead to choosing tools that are "neat" to the developer but which are impractical due to various usability problems for the user.

Scenarios are specific instantiations of specific combinations of {users, tasks, content, and tools}. Each of the tasks, tools, users, and content can pose their own usability concerns. Further usability concerns arise in the specific interactions between them.

2.3 The PUF Record Structure

PUF uses a common format to record information and requirements for each possibility (scenario, user group, task, content, and tool), which is illustrated in Table 1.

The amount of information recorded about a particular possibility depends on the level of treatment that it has received in the development. As soon as a possibility is identified, the identification section of a PUF record can be filled out giving the possibility a unique name and a narrative description, and identifying the type of possibility that it is describing. A possibility analysis will add information about other related possibilities and some initial information about the environment of the possibility. A requirements analysis will add detailed specifications and requirements that

are based on a variety of detailed analysis questions [4]. Design adds additional records and modifies information in existing records.

Identification Information	
Name	a unique, meaningful identifier
Type	scenario/user/task/tool/content
Description	clarifies meaning of name distinguishes this component from others
Linkage Information	
Who	identifies related user groups
What	identifies related tasks
How	identifies related tools
With which content	identifies related content chunks
Scenarios	identifies related scenarios
Environmental Information	
When	identifies current and potential temporal attributes
Where	identifies current and potential physical attributes
How much	quantifies the current and potential future occurrences of the possibility
Why	identifies and evaluates the justifications for possibility
Detailed Requirements	
answers to specific questions based on the possibility type	
Formal Specification	
UML translation of the above information	

Table 1. The general format of PUF possibilities records

3. THE UNIFIED MODELING LANGUAGE

Unified Modeling Language (UML) is a meta-language for specifying, visualizing, and constructing the artifacts of a software-intensive system. UML provides a standard way to develop a system's blueprints, covering conceptual things, such as business processes and system functions, as well as concrete things, such as classes written in a specific programming language, database schemas, and reusable software components [23].

The meta-language basis of UML already has the majority of the attributes necessary to record usability engineering requirements. However, due to the distributed nature of the location of these attributes and the lack of usability engineering experience of most people utilizing UML, few usability requirements are actually recorded in most developments.

Use cases are applied to capture the intended behavior of the system you are developing, without having to specify how behavior is implemented. Use cases provide a way for developers to come to a common understanding with the system's end users and domain experts. In addition, use cases serve to help validate and verify a system as it evolves during development. A Use case can model the context of a system, subsystem, or class, or model the requirements of the behavior of the elements [9]. However, UML says nothing about the content of a use case. The use case diagram models use cases, actors, and the interrelationships among them.

Class diagrams are commonly used when building an object-oriented system. A class is a description of a set of objects that share the same attributes, operations, relationships and semantics [1]. Classes may include abstractions that are part of the problem domain, as well as classes that make up an implementation. Classes can represent software things, hardware things, and even things that are purely conceptual.

3.1 Use Cases and Actors

Jacobson [19] introduced the concept of a use case which has taken on an increasingly important role in software development. He recognized two levels of use cases: essential use cases and use case instances (which are also known as scenarios). "An essential use case describes interaction independent of implicit or explicit assumptions regarding the technology or mechanisms of implementation." [19].

Constantine and Lockwood [9] discuss the role of essential use cases in user interface design. Constantine [10] recognizes the importance of the context for use cases and recommended that the developers should have the capability to represent and manipulate context as the resources for application development. He also separated the use case context into materials, tools and work areas.

Use case development is a discovery process. It is a process of finding out which information does not yet exist, and which may not yet be

understood. This information is generally entered in one of the many templates available for working with UML, such as the one developed by Cockburn [8]. Evans [11] states that few developers, even among those who have written numerous use cases, understand that the dynamic process for describing the use case is a process for finding new information and revising inappropriate specifications.

Malan and Bredemeyer [22] believe that current use case specification is not very appropriate for documenting usability requirements. One of the shortcomings for the use case is that it uses a "non-functional" field to specify the usability requirements. Also usability requirements are not specific to use cases. A use case defines a goal-oriented set of interaction between actors and system, both of which are documented elsewhere in UML. Use case specification needs to integrate who (actors) does what (interaction) with the system, for certain purpose (goals).

Various authors have suggested that the use cases should include a greater amount of usability-related information. Lilly [21] stated that a good use case specification should at least answer the following questions: Who (actors), why (goals and/or context), when (the triggering events), what (normal flow) and what else (alternative and/or exceptional flow). These same questions are the basis of requirements in PUF. Cockburn [7] advocates that use case descriptions should include the context and all the circumstance of the primary actor's goal. Lamsweerde [20] discusses a model of the goal specification that contains types, taxonomic categories, attributes and linkages.

Few use case templates have the fields to document usability-related information, such as combining the context of use with the users/actors and goals. Most of them treat usability information as "non-functional requirements" that are not able to be specified or applied in UML. Thus, UML does not ensure that resulting systems are usable.

While use cases can provide a starting point for incorporating usability-related information, the current structure and practice does not go far enough to meet all the information needs identified in PUF.

UML uses actors to, "represent a coherent set of roles that users of use cases play when interacting with these use cases" [1]. This purpose is equivalent to that of user groups in PUF. UML does not provide any particular guidance about what information should be recorded concerning actors. Rather UML allows developers to specify their own stereotypes to describe actors and other objects. Cockburn recognizes the importance of

specifying actor and stakeholder interests, but does not present a particular template for specifying properties of actors. He suggests that, "the use case's name is the primary actor's goal" [8].

3.2 Classes

Whereas use cases are used to document requirements, UML class diagrams are commonly used to document design. "A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics. A class implements one or more interfaces." [1]. Despite the goal of implementing one or more interfaces, classes do not directly document how they meet the interaction needs of actors. Rather, UML uses a set of associations between classes, use cases, and actors.

Because of their emphasis on designing software, class attributes and operations tend to be focused on technical aspects of classes. However, from a functional perspective, attributes and operations provide the closest concepts in UML to the PUF concepts of content and tools.

3.3 The Need to Add Usability Requirements to UML

Since current UML separates usability-related requirements from the development procedure, there is a need for some form of usability engineering, such as PUF, to supply usability-related requirements to the development. If the PUF methodology is applied ahead of the UML development, it will bridge the gap between usability requirements and functional requirements and help produce a more usable application.

4. APPLYING PUF IN UML

This section will consider the candidate notations in UML that can contain the PUF data and then, how each field in the PUF records can be mapped into these notations.

Figure 2 illustrates the high level mapping from PUF to UML. The components on the left side are from PUF and those on the right side are from UML. The layout of the PUF components has been simplified from that of Figure 1, to focus on the correspondence of PUF to UML components. Because scenarios are the hub of the other components in the PUF methodology, the tasks, users, content and tools serve together as the context

of use for the scenarios. The layout of the UML components includes: use cases, actors, and classes. These components are linked with each other by association relationships. Use cases include both essential use cases and use case instances. Attributes and operations are subcomponents of classes that relate to PUF components. The component level mappings identified here are based on similarity of purpose. The information contained in corresponding components is often at different levels of granularity.

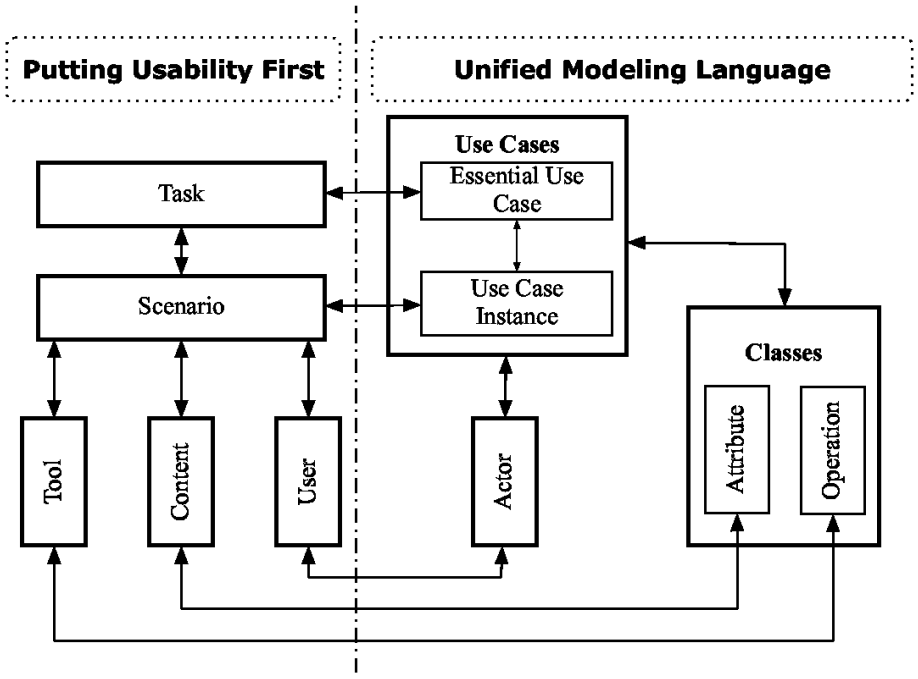


Figure 2. High level relationships between PUF and UML components

Tasks in PUF correspond to essential use cases and map to use cases in UML. Scenarios in PUF correspond to use case instances and also map to use cases in UML. As discussed, use case instances are sometimes also referred to as scenarios in the use case community. A scenario highlights the interaction between the user, the context, and the system.

Users in PUF map to actors in UML. Both user records and actors focus on the role of the user and relate users to other components that need to be designed to meet the needs of these users.

Contents in PUF map to attributes in UML. Content is used by PUF to describe the widest range of data types and modalities involved in an application. Attributes specify the data and information used in the

application. Content information in PUF is usually more abstract than attribute information in UML.

Tools in PUF map to operations in UML. The tools in PUF exist at various abstraction levels from a complete application to individual operations. The operations in UML are focused on specific operations performed by a given object. Both tools and operations deal with how a task/use case is accomplished.

The following sections provide details on mapping PUF tasks/scenarios, users, content and operations to UML. Usability properties that do not directly map to a concept in UML are also discussed for each PUF component.

4.1 Tasks, Scenarios and Use Cases

Constantine and Lockwood's definition of essential use case indicates that, like the task record in PUF, the essential use case specifies what should be done without specifying how to do it.

> *An essential use case is a structured narrative, expressed in the language of the application domain and of users, comprising a simplified, generalized abstract, technology-free and implementation-independent description of one task or interaction that is complete, meaningful, and well-defined from the point of view of users in some role or roles in relation to a system and that embodies the purpose or intentions underlying the interaction*. [9]

According to Rosson scenarios, "are similar to instances of use-cases in that they capture a single thread of execution in a given usage context" [24]. The scenario in PUF can map to use case instances. Scenarios can be more elaborate than use case instances because they narrate not only the interaction events but also the experience of the user(s) – the usage goals, expectations and reactions convey information about the system's usefulness and usability.

PUF task records map to essential use cases and PUF scenario records map to use case instances based on Cockburn's basic use case template [8] and his one-column table format of a use case [7]. In UML diagrams, most of this information is not available. Table 2 illustrates the mapping from PUF task records to UML use case diagrams. This table shows that the identification and linkage information in PUF task records can easily be

modeled in UML. This provides a good starting point for ensuring that records can be mapped successfully.

Some of the detailed requirements fields in PUF task records also readily link with UML specifications. Task operations map to actions in the use cases and to operations in classes. Requirements of users map to actors.

PUF: *Task / Scenario*	**UML:** *Use cases*
Identification Information	
name	use case name
type: task / scenario	**use case**
description	-- via a new property
Linkage Information	
who	associated **actors**
what	(other) associated **use cases**
how	**operations** associated via **classes** and **use cases**
with which content	**attributes** associated via **classes** and **use cases**
scenarios	(other) associated **use cases**
Environment Information	
when	-- via a new property
where	-- via a new property
how much	-- via a new property
why	-- via a new property
Detailed Requirements	
task operations	-- via a new property
requirements of users	-- via a new property
communications	-- via a new property
learning	-- via a new property
error handling	-- via a new property
problem details	-- via a new property

Table 2. Mappings from tasks and scenarios to use cases

In the environmental information section the why, which is used to record justification details, does not map into the current use case template. Detailed requirements in PUF task records that do not map into UML specifications include: communications, learning, error handling and problem details. These PUF fields need to be added as new properties to UML essential use case specifications.

The use case template also includes some information not provided by PUF. Use cases may specify a sequence of actions, the specified route of achieving the task, either success or failure, kinds of association and interface interactions. PUF only identifies that there are relationships among components, while UML subdivides these relationships into include and extend associations. The interface attribute describes the interaction between actors and use cases. The PUF task record only specifies the context information and does not specify the kinds of linkage and interaction.

4.2 Additional Usability-Related Properties for Use Cases

Although the use case and other linked UML components provide a location for some of the following information, there may be a need for additional properties to further expand upon. The following new properties should be added to UML use case diagrams, to record additional usability-related information provided by PUF:

When and Where task used. The use case may be limited by its environment, or might be used in a broader environment to achieve some greater benefits. If the use case is used in a different location, different frequency of use, and/or different distribution of peak usage, the design for the use case will be different. Designers should know how to design the use case to make it still achievable in various situations.

Why. Justification is an important predictor of potential future success. If a use case does not fit the overall development, or costs of the use case exceed either the benefits of serving it or available resources, it will be impractical to develop special tools for the use case. Developers should know the factors that will influence the feasibility and acceptability of possible designs.

Task operations. This PUF field describes operational concerns. Elaboration is needed to understand the operations of essential use cases to evaluate how well current operations work and how future operations might provide improvements. When designers start interaction design, they should know whether the interaction meets the goal of the use case, whether there are alternatives to achieve the use case, and what feedbacks the use case should provide.

Requirements of users. It is important to recognize the requirements that use cases place on actors. The developers should know how to design the interfaces or interactions for the use case to meet users' current skills and mental and physical capabilities. This information may require additional use cases or tools to help users reduce the impact of these factors.

Communications. When users interact with the application, communications take place. The task may require users to communicate with other users or tools. Developers should know how to design the current use case for different language, different frequency, different media, and different security levels in communications or whether to create some new potential use cases to better serve these communications.

Learning. To accomplish a use case, users need to learn how to interact with the application. This implies that there might be a new use case for training. Training learning through different methods, feedback, time and environments, create different learning outcomes. Developers should consider the learning needs and capabilities of the intended users. Developers should know how to design a usable learning system and be aware that different methods, feedback, time and environments could influence the users' learning results.

Error handling. Use cases should acknowledge where and when errors may occur. Developers should recognize these situations and determine how to help users avoid or handle them.

Problem details. When developers design the solution for problems, the problem details should be thoroughly known. This information will help developers develop a more effective and more efficient design solution.

4.3 Users and Actors

PUF identifies different user groups based on their different characteristics and interaction needs. An actor in UML identifies a role that a user can play without necessarily specifying any characteristics or interaction needs. Table 3 illustrates the mapping from PUF user records to UML actors based on UML specification version 1.5 [23] and Booch, Rumbaugh and Jacobson [1]. This table shows that most identification and linkage information in PUF user records can easily be transferred directly to

UML. This provides a good starting point for ensuring that records can be mapped successfully.

PUF: *User*	**UML:** *Actor*
Identification Information	
name	actor name
type: user	**actor**
description	--- via a property of a stereotype
Linkage Information	
who	other associated **actors**
what	associated **essential use cases**
how	**operations** associated via **classes** and **use cases**
with which content	**attributes** associated via **classes** and **use cases**
scenarios	associated **use case instances**
Environment Information	
when	-- via a property of a stereotype
where	-- via a property of a stereotype
how much	-- via a property of a stereotype
why	-- via a property of a stereotype
Detailed Requirements	
physical characteristics	-- via a property of a stereotype
mental characteristics	-- via a property of a stereotype
social characteristics	-- via a property of a stereotype
group characteristics	-- via a property of a stereotype

Table 3. Mapping from users to essential actors

Many useful fields in PUF user records are not found in current UML actor records. Without recording and using this information, there is no way of ensuring that resulting systems will meet the unique usability needs of different groups of users. A UML stereotype can be used to define these additional properties of actors for transferring user description from PUF.

The user record description field specifies the users' characteristics of membership in this group, especially focusing on how the user group is different from other related groups.

If we want to build a more usable application, we should identify all the possible contents and tools that might be used by users. Although UML does

not have direct linkages from actors to operations and attributes, actors can indirectly touch the operations and attributes through linkage between use case and class responsibility. However, without more direct linkages, developers may fail to recognize situations where new tools need to be compatible with existing tools and content.

While some environmental information can be obtained by linkages to use case instances, this structure does not allow easy identification and differentiation of environmental factors that are unique to a particular user group. In user detailed requirements, PUF specifies the physical characteristics and capabilities, mental characteristics and capabilities, and social characteristics and capabilities of individuals, and characteristics of groups. This information can further help the designers to design the application according to the users' unique characteristics.

4.4 Additional Usability-Related Properties for Actors

The following new properties should be added as a basic structure for stereotypes for UML actors, to record additional usability-related information provided by PUF:

When and Where actors operate. Different users may operate in different environments. Each different environment, may involve different usability and accessibility challenges that need to be handled by a system for it to successfully meet the needs of that group of users.

How much. Different users may have differing levels of involvement with different use cases. High levels of involvement generally mean that users will stay familiar with the operations of systems used for the use case. Infrequent involvement may suggest the need for refresher style retraining before performing a use case or higher levels of help to assist in their performance.

Why. Justification is an important predictor of potential future success. If a user's needs do not fit the overall development, or the cost of serving the user exceeds the resources available or the benefits of such service, it will be impractical to develop special tools for the user. Developers should know the factors that will influence the feasibility and acceptability of possible designs.

Physical. There are various physical limitations and impairments the users may experience. Identifying this information, developers should consider how to design the application to fit the range of physical capabilities experienced by intended users, and whether they should design some new tools for users to reduce the impact of any physical limitations.

Mental. Users' mental characteristics influence how they typically react to a variety of interactions and interfaces. Developers should consider whether to create new tools and how to design the application to fit or change actors' mental capabilities.

Social characteristics and capabilities. Users may come from various social communities with different social backgrounds. This information is important for designers to determine how to design the interfaces and interaction sequences for users who have the cultural and/or linguistic differences with each other.

Groups. Membership in a group and or acting as a representative of a group may influence a user's actions. Developers should be made aware of group membership situations that may influence the actions of a user.

4.5 Content and Attributes

PUF content records specify high level logical content chunks of data or information. Attributes in a class identify particular data components, generally at a detailed level. PUF content records map to high level data structures of attributes in UML. The content component in PUF may be implemented by one or more attributes in UML.

Table 4 illustrates the mapping from PUF content records to UML attributes based on UML specification version 1.5 [23] and Booch, Rumbaugh and Jacobson [1]. This table shows that most identification and linkage information in PUF user records can easily be transferred directly to UML. This provides a good starting point for ensuring that records can be mapped successfully. PUF detailed requirements are generally closer to implementation considerations and also map to UML components.

ISO 14915-3 [13] defines content in terms of various information types that serve particular tasks and users. It classifies content type using various dimensions including: physical or conceptual content and static or dynamic

content. PUF content chunks, while serving tasks and users, need not be limited to a single set of dimensions. PUF uses content descriptions to identify relevant dimensions and other attributes that may influence the use and usability of the content chunk.

UML attributes currently focus on the data contents of the attribute without considering its environment. New properties of attributes are necessary to incorporate descriptions of the content and environmental information about content chunks from PUF.

PUF: *Content*	**UML:** *Attribute*
Identification Information	
name	attribute name
type: user	**attribute** within a **class**
description	-- via a new property
Linkage Information	
who	**actors** associated via use cases, with **visibility**
what	associated **essential use cases**
how	**operations** within the **class**
with which content	other **attributes** within the class
scenarios	associated **use case instances**
Environment Information	
when	-- via a new property
where	-- via a new property
how much	-- via a new property
why	-- via a new property
Detailed Requirements	
structure	via **subclasses** also includes **multiplicity**
semantics	via **constraints**
requirements on users	**visibility**
how content handled	**operations** associated via **classes** & associated **use case instances**
when content used	**state machines** associated to **operations**
where content comes from & is used	**interaction diagrams**

Table 4. Mapping from content to attributes

4.6 Additional Usability-Related Properties for Attributes

The following new properties should be added to UML descriptions of attributes, to record additional usability-related information provided by PUF:

When and Where attributes are used. Attributes may need to be handled differently in different temporal and environmental situations. For example, some situations may call for precise details while others may prefer summary data. Each different situation may involve different usability and accessibility challenges that need to be handled by a system for it to successfully work with an attribute.

How much. Attributes may be used in a system at considerably different frequencies of use. High frequencies of use generally mean that users will stay familiar with the meaning, format, and use of attributes. Infrequent use may suggest the need for higher levels of assistance in working with particular attributes.

Why. Justification is an important predictor of potential future success. If the cost of including an attribute exceeds the resources available or the benefits of such inclusion, it will be impractical to consider it for inclusion. Developers should know the factors that will influence the feasibility and acceptability of possible designs.

Although various UML components provide a location for some of the following information, there may be a need for additional properties to further expand upon:

Structure. Interface designers should consider where it is necessary to organize several linked attributes or whether they should create some new attributes to design more meaningful information for the users.

Semantics. The interface designer should understand the purpose of the attribute, and consider how to design the attribute to let users understand the information so that it can be used for different purposes in various use cases.

Requirements on users. There are many users who will use the attribute. The interface designer should consider where it is necessary to design the same information in different manners to be easily understood for various users with different characteristics.

How content handled. Attributes will be operated by users with various tools. Developers should know how to design a more usable attribute that can be easily handled by all of the input tools, output tools and operation tools.

When and where content is used and comes from. This concerns the environment in which the attribute is obtained and used, and what environmental factors may limit the usability of the attribute.

4.7 Tools and Operations

The PUF perspective on tools is that they are developed and used to serve the needs of users and tasks and contents. Tools include: physical tools, software tools, and procedural tools. An operation is a service that an instance of the class may be requested to perform. Operations are detailed software tools.

Table 5 illustrates the mapping from PUF tool records to UML operations based on UML specification version 1.5 [23] and Booch, Rumbaugh and Jacobson [1]. This table shows that most identification and linkage information in PUF user records can easily be transferred directly to UML. This provides a good starting point for ensuring that records can be mapped successfully.

PUF uses a tool description to provide an initial narrative description of the nature and operations of the tool. This description is later refined in the detailed requirements. However, it remains useful for help narratives and other more general purposes. UML operations currently focus on the internal processing of the operation without considering its environment or many of the usage-related detailed requirements identified by PUF. PUF detailed requirements of tools are critical because they relate directly with detailed requirements of tasks. This relationship is important in insuring that tools are designed to meet usability requirements identified for the tasks they serve. New properties of operations are necessary to incorporate descriptions of the operations, environmental information, and detailed requirements about tools from PUF.

PUF: *Tool*	**UML:** *Operation*
Identification Information	
Name	operation name
Type: tool	**operation** within a **class**
description	-- via a new property
Linkage Information	
Who	**actors** associated via use cases
What	associated **essential use cases**
How	other **operations** within the **class**
with which	**attributes** within the **class**
scenarios	associated **use case instances**
Environment Information	
When	-- via a new property
where	-- via a new property
how much	-- via a new property
Why	-- via a new property
Detailed Requirements	
tool operations	actions of **use cases**
requirements of users	-- via a new property
communications	-- via a new property
learning	-- via a new property
error handling	-- via a new property
problem details	-- via a new property

Table 5. Mapping from tools to operations

4.8 Additional Usability-Related Properties for Operations

The following new properties should be added to UML descriptions of operations, to record additional usability-related information provided by PUF:

When and Where attributes are used, **How much, Why**. The rationale is similar to that for attributes discussed in section 4.6.

Requirements of users. Different tools require different skills and abilities to operate them successfully. It is important to recognize the abilities and skills that will be necessary for a given tool and then to compare them with the skills and abilities of the various proposed users.

Communications. Tools are created to communicate with users and other tools. Developers should consider the potential impact of different media and methods that might be used to communicate with users and with other linked tools. This involves recognizing the effectiveness of various current media and methods.

Learning. Developers should consider the learning needs and capabilities of the intended users. It is possible that additional tools must be built to help users learn tools being developed that serve application-related use cases.

Error handling. Tools should acknowledge what errors might occur from the user-side and tool-side. Developers should recognize these situations and determine how to help users avoid or handle them.

Problem details. This field/property is used to document known problems with this tool and tools that it is designed to replace.

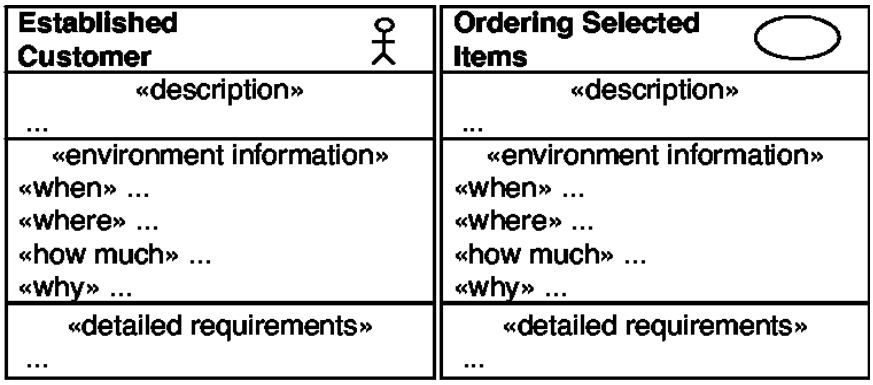
5. IMPLEMENTING THESE ADDITIONS IN UML

As noted in Section 4, a number of the concepts in PUF are directly supported in UML. For example, PUF users are mapped directly to UML actors, PUF tasks are mapped to UML use cases, PUF contents are mapped to UML attributes and PUF tools are mapped to UML operations. As well, linkage information between users, tasks, content and tools are captured in UML with associations. The additional usability-related properties discussed in Section 4 that are not directly supported in UML include descriptions, environment information and detailed requirements. However, UML was designed to be extensible using annotations, stereotypes, constraints and tagged values. In this section we describe how UML can be used to express these additional PUF usability-related properties. We describe one approach; alternative approaches are possible given the flexibility of UML.

In our approach, each usability property that does not map directly to a UML concept is associated with a stereotype (cf. Table 6). These additional stereotypes will be associated with actors, use cases, attributes, and operations either using notes or classes.

Stereotype	Use Cases	Actors	Attributes	Operations
«description»	X	X	X	X
«environment information»	X	X	X	X
«when»	X	X	X	X
«where»	X	X	X	X
«how much»	X	X	X	X
«why»	X	X	X	X
«detailed requirements»	X	X	X	X
«task operations»	X			
«requirements of users»	X			X
«communications»	X			X
«learning»	X			X
«error handling»	X			X
«problem details»	X			X
«physical characteristics»		X		
«mental characteristics»		X		
«social characteristics»		X		
«group characteristics»		X		

Table 6. Stereotypes used to identify PUF usability properties in UML



(a) (b)

Figure 3. UML notation for associating PUF properties to user and task

Classes with compartments for ‘description’, ‘environment information’ and ‘detailed requirements’ are used to associate usability property stereotypes with users and tasks. To distinguish a class as being an actor or a user case, an actor icon or a use case oval icon is positioned in the top right corner of the class rectangle. Figure 3 is an example of an actor (a) and a use case (b) using this notation.

Stereotyped notes are used to annotate attributes and operations with usability information. The note is attached to the attribute or operation with a dependency relationship. The note will be stereotyped depending on the information that has been provided and may be structured with multiple stereotypes to reduce clutter. To further reduce clutter in the diagram, the note may be used to link to a document with the detailed information. Figure 4 shows examples of using stereotyped notes to express the PUF usability properties for attributes and operations.

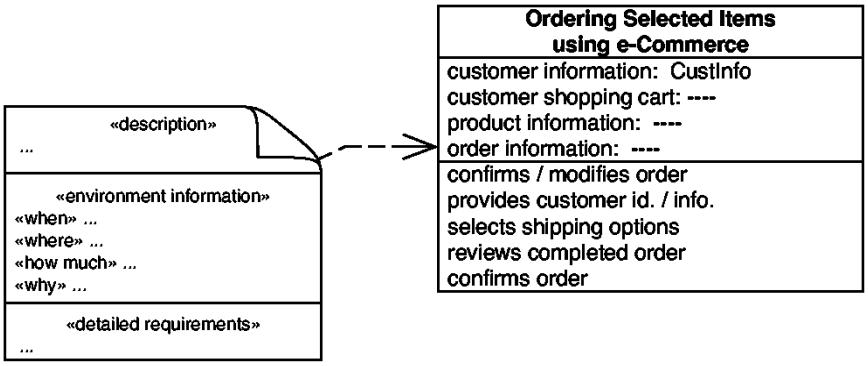


Figure 4. UML notation for associating PUF properties to attributes and operations

6. EXAMPLE TRANSFORMATIONS

This section provides examples of transformations of two PUF records from an e-Commerce application: a user record describing established customers and a task record describing placing an order for items already in a virtual shopping cart.

6.1 User Example

Consider the following PUF user record:

Identification Information
Name: established customer
Type: user
Description: a customer who has an existing account and who has made previous purchases from this e-Commerce site.

Linkage Information
Who: a specialization of a customer
What: identifying items to order; selecting and deselecting items; ordering selected items; enquiring about the status of orders; returning items.
How: the user may choose to perform tasks via telephone, via e-Commerce, in person at a physical location, or using some combination of these three tools.
With which content: customer information, customer shopping cart contents, product information, order information.
Scenarios: created from all combinations of tasks (from *What*) and tools (from *How*) performed by established customers.

Environment Information
When: whenever the user needs one or more products.
Where: at home; in an office; in an internet café; in a store.
How much: between 1 and 6 times a month.
Why: either to meet personal needs or the needs of some organization.

Detailed Requirements
Physical characteristics: requires the ability to use the methods specified in the scenarios; may include disabilities that will require use of assistive technologies.
Mental characteristics: ability to make purchase decisions; may want help to understand processing options and product features.
Social characteristics: understands English language.
Group characteristics: may act as an individual or as a member of an organization; greater accountability will be expected of purchases made for an organization

Figure 5 shows how the user ‘established customer’ fits into the inheritance structure of users involved in the e-Commerce application. As well, the stereotypes outlined in Section 5 are used to specify the usability properties. Common properties need only be specified once for the common ancestor.

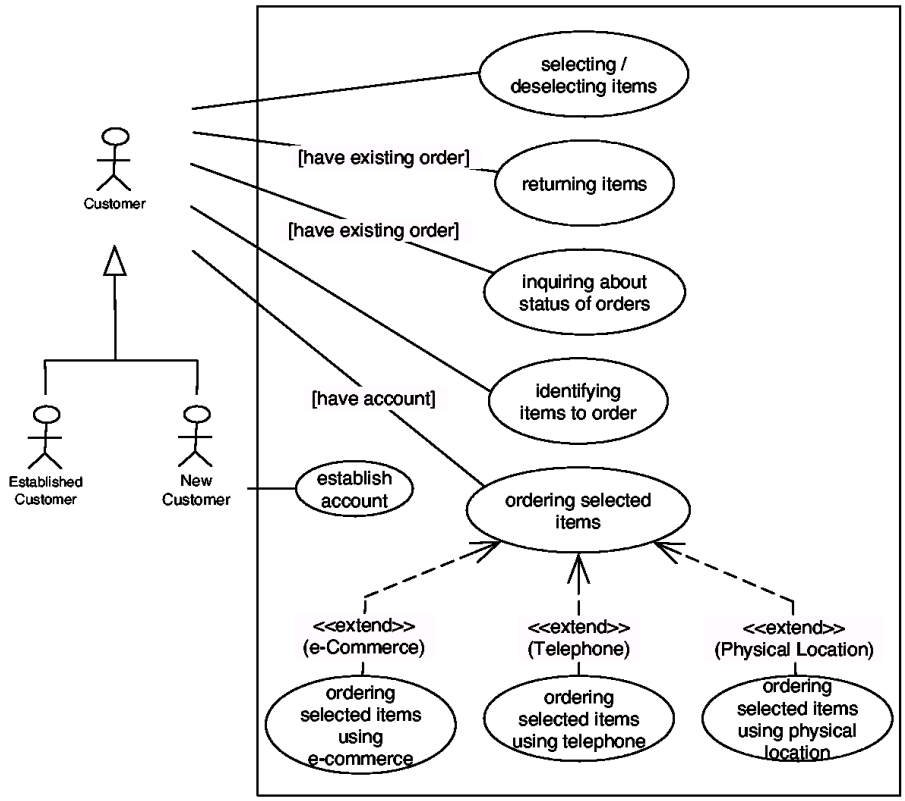


Figure 5. Users in the e-Commerce application

6.2 Task Example

Consider the following PUF task record:

Identification Information
Name: ordering selected items using e-Commerce
Type: task
Description: placing an order for items already selected and currently in the customer's virtual shopping cart.

Linkage Information
Who: customers; sales clerks
What: associated with selecting and deselecting items; enquiring about the status of orders.
How: part of an e-Commerce application.
With which content: customer information, customer shopping cart contents, product information, order information.

Scenarios: new customer ordering selected items; established customer ordering selected items; sales clerk ordering selected items.

Environment Information
When: after selecting items for shopping cart.
Where: via the internet from at home; in an office; in an internet café; in a store.
How much: 2 minutes per order times 1000 customer orders per day.
Why: to allow ordering from a wide range of locations is expected to increase sales by 2000 items per day.

Detailed Requirements
Task operations: user confirms/modifies order; user provides customer identification / information; user selects shipping options; user reviews completed order; user confirms order.
Requirements of users: must have credit card; must use supported Web browser; must understand English language.
Communications: this task involves formal interactive communications between a single user and the e-Commerce system
Learning: the system must be self-descriptive and not require any training; the user may wish to access descriptive help while performing this task.
Error handling: the system should validate data at each step before proceeding and should help the user identify and make any required changes; the system should allow the user to edit all user input fields prior to confirming the order; items in an established customer's virtual shopping cart should remain until the customer deselects or orders them or until they have remained there for over one month.
Problem details: the system must be at least as usable as Amazon.com.

Figure 6 shows the task 'ordering selected items using e-Commerce' as a use case in the context of other use cases in the e-Commerce application. As well, its linkage with the actors is shown. Since 'customer' is able to 'order selected items' it is possible, given the generalization relationships, for an 'established customer' to 'order selected items using e-Commerce'.

The task may be implemented as a class as design proceeds. The class will contain attributes and operations corresponding to the content and tools of the task (cf. Figure 6). Note that when a task corresponds one-to-one to a class it is possible to use additional compartments in the class to repeat the task's PUF usability properties.

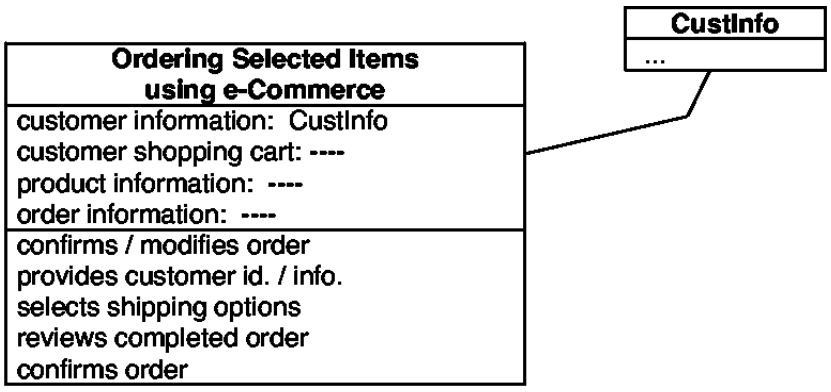


Figure 6. Use cases in the e-Commerce application

7. CONCLUSION

Putting Usability First (PUF) methodology is a user centered approach to systems development. In this chapter we describe a mapping of PUF descriptions to a common software modeling language: the Unified Modeling Language (UML). It is hoped that expressing the PUF methodology in UML can ensure the application is developed in a context rich information environment that minimizes the occurrence of usability problems.

We describe how many of the concepts in PUF are directly expressed in UML. We also describe how PUF usability properties can be specified using UML annotations and stereotypes. Mapping PUF to UML makes it possible to trace the usability requirements to the design specified in UML and helps bridge the gap between usability engineering and software engineering. As the design is refined during the software engineering process the usability properties can also be maintained and refined.

The transformations of usability requirements to software engineering specifications described here allow usability engineers and software engineers to integrate their efforts while performing their own processes. This correspondence between usability and software design will hopefully result in more usable software products and improve the traceability of usability requirements.

REFERENCES

1. Booch, G., Rumbaugh, J. and Jacobson, I. The Unified Modeling Language User Guide. Reading MA: Addison-Wesley, 1999.
2. Carter, J. Developing e-Commerce Systems. Upper Saddle River, NJ: Prentice-Hall, 2002.
3. Carter, J. Putting usability first in the design of Web sites, *Proceedings of WebNet'97*, (Toronto, Nov. 1997), 142-148.
4. Carter, J. "Combining Task Analysis with Software Engineering in a Methodology for Designing Interactive Systems," Taking Software Design Seriously: Practical Techniques for Human-Computer Interaction Design*,* J. Karat (Ed.). Boston: Academic Press Inc., 1991, 209-234.
5. Carter, J. Juggling Concern for Completeness and Consistency with Concerns for Flexibility and Adaptability Using MOST, *Proceedings of the 34th Annual Meeting of the Human Factors Society*, (Oct. 1990), 341-345.
6. Carter, J. A framework for the development of multimedia systems for use in engineering education, to appear in *Computers & Education*.
7. Cockburn, A. Writing Effective Use Cases. Boston, MA: Addison-Wesley, 2001.
8. Cockburn, A. Basic Use Case Template, TR 96.03a, Humans and Technology, version 2, October 26, 1998. http://alistair.cockburn.us/usecases/uctempla.htm
9. Constantine, L. and Lockwood, L. Software for Use: A Practical Guide to the Models and Methods of Usage Centered Design. Reading, MA: Addison-Wesley, 1999.
10. Constantine, L. Essential Modeling Use Cases for User Interfaces Interaction, Constantine & Lockwood Ltd, 1995.
11. Evans, G., Why Are Use Cases So Painful? http://www.evanetics.com/articles/Modeling/UCPainful.htm
12. Hefley, W., Buie, E., Lynch, G., Muller, M., Hoecker, D., Carter, J. and Roth, J. Integrating Human Factors With Software Engineering. *Proceedings of the 1994 Annual Meeting of the Human Factors and Ergonomics Society*, 1994, 315-319.
13. International Organization for Standard, ISO Standard 14915-3 Software ergonomics for multimedia user interfaces – Media selection and combination, International Standard, 2003.
14. International Organization for Standardization, ISO Technical Specification 16982 Ergonomics of human-system interaction – Usability methods supporting human centered design, 2002.
15. International Organization for Standardization, ISO Technical Report 18529 Human-centred life cycle process descriptions, 2000.
16. International Organization for Standardization, ISO International Standard 13407 Human-centred Design Processes for Interactive Systems, 1999.
17. International Organization for Standardization, ISO International Standard 9241-11 Guidance on Usability, 1998.
18. International Organization for Standardization, ISO/IEC Technical Report 15504-2 Information technology – Software process assessment, 1998.
19. Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G. Object-Oriented Software Engineering: A Use case Driven Approach. Addison-Wesley, 1992.
20. Lamsweerde, A van. Goal-Oriented Requirements Engineering: From System Objectives to UML Models to Precise Software Specifications. *Proceedings of the 25th International Conference on Software Engineering* (ICSE.03), 2003, 744-745.

21. Lilly, S. “How to Avoid Use-Case Pitfalls?”, Software Development Magazine, Jan. 2000. http://www.sdmagazine.com/articles/2000/0001/
22. Malan, R. and Bredemeyer, D. Functional Requirements and Use Cases, Bredemeyer Consulting, June, 1999. http://www.bredemeyer.com/pdf_files/functreq.pdf
23. Object Management Group, Inc. OMG Unified Modeling Language Specification Version 1.5, March 2003.
24. Rosson, M. “Integrating Development of Task and Object Models”, Communications of the ACM, 42(1), Jan 1999, 49-56.
25. Rubenstein, R. and Hersh, H. The Human Factor: Designing Computer Systems for People. Maynard, MA: Digital Press, 1984.