

# Usage-Centered Design and Software Engineering: Models for Integration

Larry Constantine  
*Constantine & Lockwood, Ltd.*  
*University of Technology,*  
*Sydney (Australia)*  
*lconstantine@foruse.com*

Robert Biddle  
*Victoria University of*  
*Wellington (New Zealand)*  
*robert@mcs.vuw.ac.nz*

James Noble  
*Victoria University of*  
*Wellington (New Zealand)*  
*kjx@mcs.vuw.ac.nz*

## Abstract

*This paper argues for a model-driven framework for integrating human interface engineering with software engineering. The usage-centered design process, a robust and proven process grounded in software engineering, is briefly described and contrasted with traditional user-centered approaches. The connections between the core models of usage-centered design and their software engineering counterparts are outlined, and the advantages of this approach to bridging the gap are discussed.*

## 1. Introduction

All software ultimately serves human needs and interests, hence provision for effective interaction between human users and software would reasonably be expected to be integral to all software engineering. Such has not been the case, historically. Until recently, software engineering methods and practice have focused primarily or exclusively on the internal structure and functioning of computer programs, leaving the interaction with users and the user interface that supports that interaction to other disciplines and professionals. For its part, the human-factors community has shown relatively little interest in the problems of software engineering as such.

Recent years have seen an upsurge in interest in narrowing the gap between human-computer interaction (HCI) and software engineering (SE) perspectives, particularly in regards to object-oriented methods [1]. The gap is, of course, multifaceted, with differences relating to historical evolution, training, professional orientation, and technical focus, as well as in methods, tools, models, and techniques. Although the academic, research, conceptual, and professional aspects of this gap are interesting in their own right, in this paper we shall

focus on pragmatic matters in integrating the means and methods by which software, with its human interface, is engineered.

## 2. Designing the Human Interface

Responsibility for human interface design, while eschewed by software engineering, is claimed by a number of professions with varying perspectives. Terminology is often marked by vehement debate over the professional responsibilities of graphic designers versus visual designers or the distinctions between user interface design and interaction design or the precise domain covered by information architecture.

In the final analysis, however, the engineering of the interfaces between humans and software requires solving design problems in three intimately interrelated areas: architecture, presentation, and interaction. The architecture of the interface refers to the overall and large-scale organization of the interface, that is, the way in which the interface as a whole is partitioned into distinct and recognizable regions or parts, how these parts are related and interconnected, and how the user navigates among these parts. Presentation design addresses the specification of how information is presented to users by visual, audible, haptic, or other means. In conventional graphical user interfaces, presentation design involves the selection and design of visual elements as well as their layout within and distribution among the various parts of the user interface, as well as details of appearance, such as color and shape. Interaction design addresses the specification of the means by which users interact with a system and includes the organization of discrete steps in processes, the selection or specification of gestures or idioms for interaction, the sequencing of actions, and workflow.

These three aspects of interface design interact strongly and are ultimately inseparable, even if some

designers and methods strive to address them independently. The choice of a particular selection component, for example, implies constraints on appearance and forms of interaction with effects on both layout and the use of “screen real estate,” which in turn may impact what features can be combined within a single part of the interface and which must be separated.

While we are keenly aware that others have used other terms and definitions, in this paper we will use the term human interface engineering in preference to user interface design and will consider it to embrace the three areas of interface architecture, presentation design, and interaction design.

Ultimately, any integrated approach must not only address these three areas but also the design of the supporting software as such. In the usage-centered software engineering approach described in this paper, a single set of models forms the basis of integration and coordination of human interface and software engineering.

### 3. Usage-centered or user-centered?

Usage-centered design [2, 3] is a systematic, model-driven approach to human interface engineering for software and Web-based applications. Beginning with early work on task modeling based on use cases [4, 5], it has evolved into a sophisticated process that has proved itself on projects of widely varying scope and scale in a variety of application areas [3, 6, 7, 8, 9]. Because of its focus on designing software for use based on robust task models, the approach has proved particularly effective in delivering innovative—even award-winning—designs that both enhance performance and are easily learned [9, 10, 11].

Usage-centered design can be distinguished from the more widely recognized and practiced user-centered design [12]. In one form or another, the latter has become the dominant design philosophy of human computer interaction. As one indication of this dominance, a Google search on user-centered design yields over 74,000 hits, in contrast to about 2,000 hits for usage-centered design. Nevertheless, the application and influence of usage-centered approaches has grown steadily, and its core concepts and techniques have been

incorporated into other methods, including the Rational Unified Process [13, 14] and even in some cases under the rubric of user-centered design [15, 16].

The tensions between software engineering and user-centered methods are widely recognized [17]. An emphasis on thorough investigation and extensive field observation, particularly in ethnographic approaches, combined with repeated cycles of prototyping and user feedback, leads to a substantial, often unpredictable, analysis and design commitment that typically must precede software design and development. Usability testing, a cornerstone of user-centered approaches, typically comes late in the development cycle when its findings are all too often ignored or deferred for later releases, especially when testing uncovers architectural defects or problems in the basic organization of the user interface.

Usage-centered design differs from its older and better established counterpart in several important ways. As the name suggests, the center of attention is not users per se but usage, that is, the tasks intended by users and how these are accomplished. This difference in emphasis is reflected in differing practices that have a significant impact on the development life cycle and on integration with software engineering. Table 1 summarizes salient differences in the two approaches.

Unlike user-centered design, whose methods and traditions are firmly rooted in the human factors and human-computer interaction world, usage-centered design is grounded in a strong engineering orientation reflecting the background of its co-developers, including one of the early pioneer software methodologists (Constantine). Usage-centered design emerged directly from software engineering and particularly from object oriented software engineering [18]. Precisely because usage-centered design is built around extensions and refinements to well established software engineering models and techniques, such as actors and use cases [19], integration with software engineering is more straightforward.

Perhaps the most fundamental difference is in the basic organization of the design process itself. Underlying much of user-centered design as practiced is a view of user interface design as a process of successive approximations wherein a final solution emerges

**Table 1. Salient Differences between User-Centered and Usage-Centered Design**

User-Centered Design	Usage-Centered Design
Focus on users: user experience, user satisfaction	Focus on usage: improved tools supporting task accomplishment
Driven by user input	Driven by models
Substantial user involvement: user studies, participatory design, user feedback, user testing	Selective user involvement exploratory modeling, model validation, structured usability inspections
Descriptions of users, user characteristics	Models of user relationships with system
Realistic or representational design models	Abstract design models
Design by iterative prototyping	Design through modeling
Varied, often informal or unspecified processes	Systematic, fully specified process

principally through repetitive cycles of trial design alternated with user testing and feedback. Although the profession may be loathe to admit it, such approaches are appropriately described as design by trial-and-error.

In contrast, usage-centered design is conceived as an integrated concurrent engineering process aimed at producing an initial design that is essentially right in the first place. This is not to say that refinement and improvement through evaluation and feedback are not employed, but these are not the driving forces in the design process. Instead, usage-centered design is driven by interconnected models from which a final visual and interaction design are derived more or less directly by straightforward transformations. Iterative refinement applies more to the models from which the final design is derived than to the design.

#### 4. The usage-centered design process

The process, outlined schematically in Figure 1, is based on concurrent engineering. Although the core

moving from model to model as information and insight emerge and as the needs of project management and problem solving dictate. In addition, the process divides into concurrent but interdependent threads, one primarily focused on designing the human interface, and the other primarily focused on designing the internal software.

A number of variants to this process and its models have been devised to suit various contexts and varying degrees of formality, ranging from highly structured forms [2, 9, 20] to agile design for Web-based applications [3, 21], and even incorporation with extreme programming [22].

In lieu of more unstructured or open-ended investigation typical of contextual inquiry or other ethnographic techniques, a model-driven process is used even for problem definition and requirements gathering. In this exploratory modeling [23] (not shown in the diagram), questions and areas for investigation are identified by constructing provisional user role and task models based on whatever background information or knowledge is at hand. Limited, sharply focused inquiries,

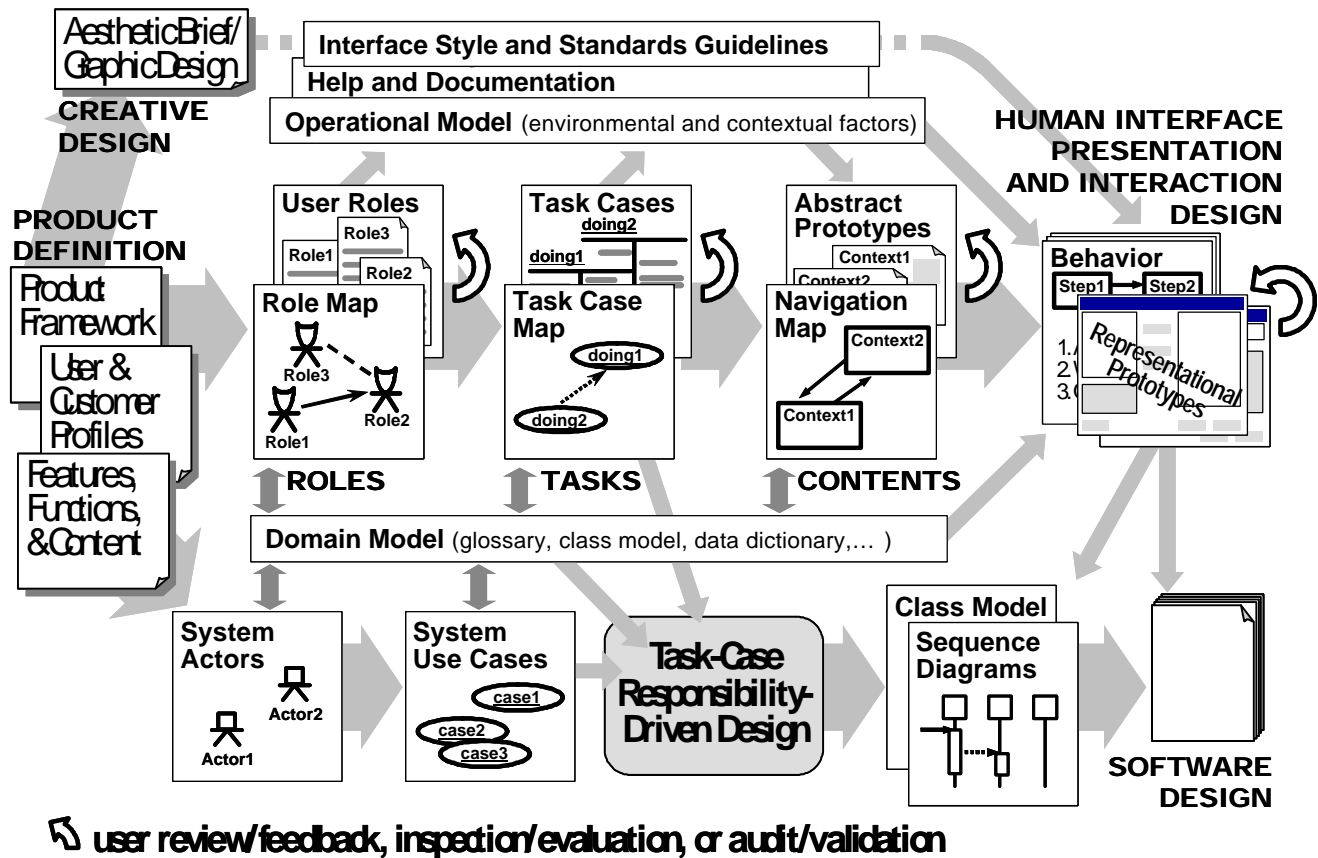


Figure 1. Schematic Outline of Usage-Centered Design Process

models are connected in a logical sequence, in reality experienced practitioners develop them concurrently,

observation, surveys, or investigations are then conducted to answer questions, resolve ambiguities, and supply

missing information as identified through the exploratory modeling.

#### 4.1. Usage-centered models

Presentation and interaction designs as is typical, are embodied as representational prototypes, that is, paper or other prototypes representing or approximating the actual appearance of the user interface. Usually these take the form of annotated drawings or renderings of screens, pages, or other interaction contexts supplemented by descriptions of the behavior of the interface in narrative or other convenient form, such as, flow charts or decision tables.

In the usage-centered process, the presentation and interaction designs derive directly from the contents of the three tightly integrated core models: a user role model, a user task model, and an interface content model. Although some form of user and task modeling is common to most user-centered approaches, usage-centered models are distinctive in the use of abstraction and simplification to focus precisely on matters most essential for informing the design process.

**4.1.1. User roles.** Rather than modeling actual users or user communities broadly, as is often the case with user-centered techniques, the user role model focuses exclusively on salient aspects of the relationships between users and the system as represented by the various roles users assume. In its simplest informal form, the role model describes context, characteristics, and criteria, that is, the context within which each role is assumed including the overall contextual purpose and responsibilities of the role, the characteristics of interaction within the role, and criteria for support of the role, such as specific functions or the relative importance of various design objectives. Relevant characteristics of roles have been cataloged and compiled into templates for structured modeling of user roles [2]. In practice, a simple summary of context, characteristics, and criteria on an ordinary index card may suffice in many projects. For example,

**Routine-Test-Running Role**

CONTEXT: Unsophisticated but trained operator runs predefined standard tests with limited modification (under supervision).

CHARACTERISTICS: frequent, regular performance of relatively simple but crucial repetitive tasks.

CRITERIA: simple, efficient, error-free operation.

**4.1.2. Task cases.** Developed originally by Ivar Jacobson for his object-oriented methodology [18], use cases have become ubiquitous in modern software engineering practice. As originally conceived and

conventionally employed, use cases comprise “sequences of actions...that a system...can perform by interacting with outside actors” [24]. They are usually expressed in terms of sequences of user actions and system responses.

For guiding human interface design, the focus on the system and on concrete action and interaction proved to be problematic, leading to the development of task cases, a more abstract form modeling user intentions rather than actions and system responsibilities rather than

Running Standard Test	
USER INTENTIONS	SYSTEM RESPONSIBILITIES
	1. show available standard tests
2. pick test	
3. optionally [modify test] }	4. show test configuration
5. confirm & start	6. run test
	7. report results

responses. An example is shown in Figure 2. Task cases, also known as essential use cases, are expressed from the perspective of users in roles.

**Figure 2. A task case or essential use case.**

Because task cases are essential in the sense introduced by McMenamin and Palmer [25]—that is, abstract, simplified, generalized, and technology and implementation-independent—they come closer to the essence of user needs than typical use cases, offering advantages for requirements modeling [26] and for reducing the complexity of sophisticated systems [9]. The advantages of abstraction and goal orientation in use cases are widely accepted [3, 27, 28, 29, 30].

In practice, task case models are more fine-grained than typical use case models for software engineering. Increased granularity has advantages for user interface design in exposing small, reusable tasks and in clarifying the interrelationships among user tasks. This facilitates devising a more robust and flexible interface architecture that closely conforms to the underlying structure of user task needs.

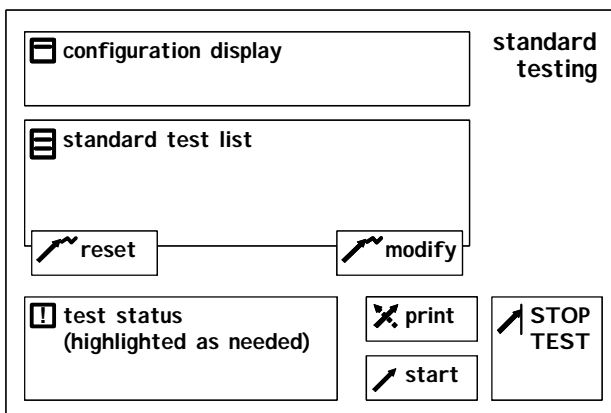
**4.1.3. Content models.** User-centered designers typically move quickly into sketching paper prototypes, but in usage-centered design, content modeling is used to first work out the architecture or overall organization of the interface and its subparts independent of their realization as actual interface components and apart from the details of appearance and behavior. The interface content model consists of a set of abstract prototypes

representing the contents of the various parts of the user interface and a navigation map representing the interconnections among all these parts.

Abstract prototypes [31] range from simple content inventories or lists of the contents of each part of the user interface to highly structured and formalized abstract prototypes based on canonical abstract components [32].

Abstract prototyping facilitates exploration and problem solving regarding the contents and organization of the various component parts of the interface without having to plunge ahead into details of appearance and layout. In their preferred form, abstract prototypes are constructed from a canonical collection of abstract components that allow for precise description of interfaces in terms of form and function independent of details of appearance and behavior and that promote the recognition of common patterns and problems in presentation and interaction design. An example of a canonical abstract prototype is shown in Figure 3.

**Figure 3. Example of an abstract prototype using**



**canonical abstract components.**

## 5. Interrelationships among models

In the process model of Figure 1, models form the bridge between human interface engineering and software engineering, providing traceability and interdependence and serving as the means for coordinating concurrent processes.

Software engineering typically begins with some form of high-level representation of the system scope in terms of the system boundary and interfaces with the environment. In the contemporary unified process and related methods, a use case diagram serves this purpose by identifying all the actors interacting with the system.

An actor, as the term is used in UML and the UP, is any entity that interacts with a system. For a usage centered approach, non-human actors, referred to as system actors, are distinguished from user actors. Indirect

users, whose interaction with the system is mediated by other users, are excluded, as they are part of the context of user actors (direct users) and are not directly involved with the user interface. User actors interact with the interface within the roles they play in relation to the system. A given actor may play in a number of different roles and the same role may be played by more than one actor.

Task cases support user roles. A given role typically requires a number of task cases, and a given task case may support multiple roles. On the software engineering side, the picture is more straightforward: system actors are supported by system use cases. Both task cases and system use cases become input for the object design, which must support them all to meet requirements. However, many details of the required software may not be determined until the presentation and interaction design are complete.

The presentation and interaction designs are based on abstract prototypes of the various interaction contexts and the navigation map modeling the interrelationships among distinct contexts. Particular user interface components with specific behavior and appearance are chosen or designed to realize the abstract components.

Abstract components are incorporated into abstract prototypes based on the particular tools and materials needed to realize each step in those task cases to be supported together within a given part of the user interface. The partitioning of the total user interface into subparts, as represented by the navigation map, is determined based on the interrelationships among task cases. A task case map in its most specific form models inclusions, extensions, and specializations relating task cases, but for many projects a clustering based on the likelihood of task cases being used together suffices.

The domain model, which captures the core concepts of the application, is developed and refined concurrently and collaboratively throughout the process and serves to link the various design models. For example, object classes referred to in the narrative body of task cases correspond to those in the contents of abstract prototypes on which the realized presentation and interaction design are based. The domain model in turn maps to the internal object design.

From this overview it should be clear that use cases, as task cases or system use cases, form the common thread that interconnects the various models and activities in the process. Indeed, task cases can directly guide the organization and contents of documentation and online help [2, 33], thus providing uniform and comprehensive traceability throughout the delivered system.

## 6. Distributed responsibilities and software

For use within conventional object-oriented software engineering approaches, such as the Unified Process, task cases (essential use cases) need to be transformed into conventional or concrete use cases. Abstract narratives defined in terms of user intentions must be elaborated into a more complex and detailed “flow of events” that refers to actual user actions in relation to the user interface as designed. In many situations, differences in granularity require combining closely related task cases into a composite concrete use cases that incorporates many if not all of the exceptional or alternative flows. Because this process of translation begins with task cases in the abbreviated form understood by the human interface designers, the detailed concrete use cases needed for software engineering must be developed by the human interface designers. Although this somewhat tedious translation introduces an added step with its attendant resource needs and opportunities for error, it has been used successfully on even very large projects [8].

The abstraction to intentions and responsibilities in the defining narrative of task cases suggests a possible more direct link to object-oriented software engineering by way of responsibility-driven design [34] and so-called CRC cards [35], both of which employ the concept of responsibility to guide software design decisions. Indeed, as has been noted [26], the role of responsibility in task cases is entirely consistent with the role of responsibility in software design. Both describe behavior without describing implementation, a commonality that enables linking the methods for determining requirements for designing human interfaces, and for designing software.

The distributed responsibility technique [26, 36] developed at Victoria University of Wellington takes task cases and system use cases in the same essential form as used for requirements definition and human interface design and uses them to drive the derivation of an object-oriented design. This technique has several demonstrated advantages. The abstraction and brevity of essential use cases allows a more agile start to early object modeling and makes possible conducting object modeling and user interface design in parallel. The key role of responsibility in both task models and object models supports traceability between the two models, as well as providing better operational guidance to developers and facilitating review processes in design evaluation.

The iterative process begins with a system object, a single class whose responsibilities are those identified as system responsibilities in the task cases and system use cases, which means the design is guaranteed to provide the behavior specified in the requirements. In successive iterations, additional classes are identified as suggested

by repeated terms within the class responsibilities or by virtue of their inclusion in the evolving domain model.

As classes are identified, responsibilities are delegated and the resulting provisional model is checked for reasonableness and refactored as appropriate. CRC cards are often used as convenient tools for representing classes, responsibilities, and collaborations. For greater precision, sequence diagrams can also be developed to clarify and detail responsibilities and collaborations.

## 7. Prospect

The usage-centered software engineering process outlined here is not a proposal but an already well established “industrial strength” process that has proved successful in numerous projects ranging up to 50+ person years completed by organizations around the world. Current areas of continued investigation include refinement in notation, compilation and elaboration of patterns based on canonical abstract components [32] and task cases [26], and continued work on common theory and metrics, such as cohesion and coupling, underlying both object design and human interface design [2, 37, 38, 39].

Perhaps most pressing is the need for tools that support usage-centered software engineering by incorporating its models and exploiting their interconnections for flexible concurrent modeling and systematic requirements tracing. The dominance of contemporary software engineering standards, in particular UML [24], is itself an impediment, as UML lacks constructs for user roles, task cases, and interface contents, forcing practitioners and process mentors to emulate the needed models in a procrustean bed constructed of UML “stereotypes” that are ill-suited to the purpose [13].

## 8. References

- [1] van Harmelen, M. *Object Modeling and User Interface Design*. Addison-Wesley, Boston, 2001.
- [2] Constantine, L. L., and Lockwood, L. A. D. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley, Reading, MA, 1999.
- [3] Constantine, L. L., and Lockwood, L. A. D. “Usage-Centered Engineering for Web Applications.” *IEEE Software*, 19 (2), March/April 2002.
- [4] Constantine, L. L. “Essentially Speaking.” *Software Development*, 2 (11), November 1994.
- [5] Constantine, L. L. “Essential Modeling: Use Cases for User Interfaces,” *interactions* 2 (2), March/April 1995.

- [6] Anderson, J., Fleek, F., Garrity, K., and Drake, F. "Integrating Usability Techniques into Software Development." *IEEE Software*, 18 (1), January/February, 2001.
- [7] Patton, J. "Extreme design: Usage-Centered Design in XP and Agile Development." In L. Constantine (Ed.), *forUSE 2002: Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, Rowley, MA, 2002.
- [8] Strobe, J. "Putting Usage-Centered Design to Work: Clinical Applications." In L. Constantine (Ed.), *forUSE 2002: Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, Rowley, MA, 2002.
- [9] Windl, H. "Designing a Winner: Creating STEP 7 Lite with Usage-Centered Design." In L. Constantine (Ed.), *forUSE 2002: Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, Rowley, MA, 2002.
- [10] Windl, H., and Constantine, L. "Performance-Centered Design: STEP 7 Lite." Winning submission, Performance-Centered Design 2001, <http://foruse.com/pcd/>
- [11] Constantine, L. L., and Lockwood, L. A. D. "Instructive Interaction." *User Experience*, 1 (3), Winter 2002.
- [12] Norman, D. A., and Draper, S. W. *User-Centered System Design*. Erlbaum, Hillsdale, NJ, 1986.
- [13] Heumann, J. "Use Cases, Usability Requirements, and User Interfaces." Tutorial Notes, OOPSLA 2002, 4-8 November. ACM, New York, 2002.
- [14] Kruchten, P., Ahlqvist, S., and Byland, S. "User Interface Design in the Rational Unified Process." In M. van Harmelen, ed., *Object Modeling and User Interface Design*. Addison-Wesley, Boston, 2001.
- [15] Scanlon, J., and Percival, L. "User-Centered Design for Different Project Types, Part 1: Overview of Core Design Activities." IBM DeveloperWorks, March 2002. <ftp://www6.software.ibm.com/software/developer/library/us-ucd.pdf>
- [16] Percival, L. and Scanlon, J. "User-Centered Design for Different Project Types, Part 2: Core Design Activities by Project Types." IBM DeveloperWorks, March 2002. <ftp://www6.software.ibm.com/software/developer/library/us-ucd2.pdf>
- [17] McCoy, T. "Letter from the Dark Side: Confessions of an Applications Developer." *interactions* 9 (6) November/December, 2002: 11 - 15.
- [18] Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Reading, MA, 1992.
- [19] Constantine, L. L., and Lockwood, L. A. D. "Structure and Style in Use Cases for User Interface Design." In M. van Harmelen (Ed.), *Object Modeling and User Interface Design*. Addison-Wesley, Boston, 2001.
- [20] Armstrong, C., and Underbakke, B. "Usage-Centered Design and the Rational Unified Process." In L. Constantine (Ed.), *forUSE 2002: Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, Rowley, MA, 2002.
- [21] Constantine, L. L. Process agility and software usability: Toward lightweight usage-centered design. *Information Age*, 8 (8), August 2002. Also in L. Constantine (Ed.), *Beyond Chaos: The Expert Edge in Managing Software Development*. Addison-Wesley, Boston, 2001.
- [22] Patton, J. "Extreme Design: Usage-Centered Design in XP and Agile Development." In L. Constantine (Ed.), *forUSE 2002: Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, Rowley, MA, 2002.
- [23] Windl, H. "Usage-Centered Exploration: Speeding the Initial Design Process." In L. Constantine (Ed.), *forUSE 2002: Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, Rowley, MA, 2002.
- [24] Rumbaugh, J., Jacobson, I., and Booch, E. G. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, MA, 1999.
- [25] McMenamin, S. M., & Palmer, J. *Essential Systems Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1984.
- [26] Biddle, R., Noble, J., and Tempero, E. "From Essential Use Cases to Objects." In L. Constantine (Ed.), *forUSE 2002: Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, Rowley, MA, 2002.
- [27] Cockburn, A. "Structuring Use Cases with Goals," *Journal of Object-Oriented Programming*, September/October 1997 and November/December 1997.
- [28] Graham, I. (1996) "Task Scripts, Use Cases and Scenarios in Object-Oriented Analysis," *Object-Oriented Systems* 3 (3), 1996.
- [29] Kaindl, H. "An Integration of Scenarios with Their Purposes in Task Modeling." Proc. Symposium on Designing Interactive Systems. ACM Press, Ann Arbor, 1995.

- [30] Lee, J., and Xue, N. "Analyzing User Requirements by Use Cases: A Goal-Driven Approach." *IEEE Software*, 16 (4) July/August 1999.
- [31] Constantine, L. L. "Rapid Abstract Prototyping." *Software Development*, 6, (11), November 1998. Reprinted in S. Ambler and L. Constantine, eds., *The Unified Process Elaboration Phase: Best Practices in Implementing the UP*. CMP, Lawrence, KS, 2000.
- [32] Constantine, L. L., Windl, H., Noble, J., and Lockwood, L. A. D. "From Abstraction to Realization: Abstract Prototypes Based on Canonical Components" Working Paper, The Convergence Colloquy, July 2000.  
<http://www.foruse.com/articles/canonical.pdf>
- [33] Lynn, R. "Beyond Code Freeze: Use Cases that Do Not Leave Documentation and QA in the Cold." In L. Constantine (Ed.), *forUSE 2002: Proceedings of the First International Conference on Usage-Centered, Task-Centered, and Performance-Centered Design*. Ampersand Press, Rowley, MA, 2002.
- [34] Wirfs-Brock, R. J., and McKean, A. *Object Design: Roles, RESPONSibilities, and Collaborations*. Addison-Wesley, Boston, 2002.
- [35] Wilkinson, N. *Using CRC Cards - An Informal Approach to OO Development*. Cambridge University Press London, 1995.
- [36] Biddle, R., Noble, J. and Tempero, E. "Essential Use Cases and Responsibility in Object-Oriented Development. In M. Oudshoorn (ed.), *Proceedings of the Australasian Computer Science Conference (ACSC2002)*. Australian Computer Society, Melbourne, 2002.
- [37] Constantine, L. L. "Visual Coherence and Usability: A Cohesion Metric for Assessing the Quality of Dialogue and Screen Designs." In Grundy, J., & Apperley, M. (eds.) *Proceedings, Sixth Australian Conference on Computer-Human Interaction*. IEEE Computer Society Press, Los Alamitos, CA, 1996.
- [38] Constantine, L., and Noble, J. "Interactive Design Metric Visualization: Visual Metric Support for User Interface Design." In Grundy, J., & Apperley, M. (eds.) *Proceedings, Sixth Australian Conference on Computer-Human Interaction*. IEEE Computer Society Press, Los Alamitos, CA, 1996.
- [39] Constantine, L. L. "Usage-Centered Software Engineering: New Models, Methods, and Metrics." In Purvis, M. (ed.) *Software Engineering: Education & Practice*. IEEE Computer Society Press, Los Alamitos, CA, 1996.