

Usage-Centered Design: Scalability and Integration with Software Engineering

Larry Constantine

University of Technology, Sydney
Constantine & Lockwood, Ltd.
58 Kathleen Circle, Rowley, MA 01969
lconstantine@foruse.com

Helmut Windl

Siemens AG
A&D AS RD 221, Gleiwitzer Str. 555
90475 Nuremberg, Germany
helmut.windl@siemens.com

Abstract

Usage-centered design is described and compared with traditional user-centered approaches in relation to software engineering and development. Experiences in integrating usage-centered design with software engineering across a spectrum of process and project models are reviewed. Successful techniques for mitigating the effects on time and resource are reviewed, including overlapping early-cycle activities, splitting support of User Actors from System Actors, recycling of usage-centered models, and newly developed agile modeling and design techniques.

1 Introduction

Usage-centered design (Constantine and Lockwood, 1999; 2002) is a systematic, model-driven approach to visual and interaction design for user interfaces in software and Web-based applications. On projects of widely varying scope and scale in a variety of application areas (Anderson et al., 2001; Constantine and Lockwood, 2002; Patton, 2002; Strobe, 2002; Windl, 2002b), usage-centered design has proved capable of expeditious delivery of superior designs (Windl and Constantine, 2001). As the name suggests, usage-centered design differs from conventional user-centered approaches primarily in a shift of focus from users *per se* to usage, that is, to the tasks to be accomplished by users. This difference in emphasis is reflected in differing practices with a significant impact on the development life cycle and integration with software engineering and development. Most importantly, usage-centered design eschews the repetitive cycles of trial design and user testing common to traditional user-centered approaches in favor of a design process in which final solutions may be derived more or less directly from robust and highly refined models reflecting genuine user needs. The goal is an initial design requiring only limited usability testing and minimal refinement. Table 1 summarizes salient differences.

Whereas user-centered design emerged from a separate profession and evolved largely independent of software engineering, 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. Usage-centered design integrates readily with software engineering precisely because it was developed from the outset to be compatible with object-oriented software engineering, using extensions and refinements to well established models and techniques, such as actors and use cases (Constantine and Lockwood, 2001).

Table 1: Salient Differences between Usage-Centered and User-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
Design by iterative prototyping	Design by modeling
Varied, often informal or unspecified processes	Systematic, fully specified process
Evolution through trial-and-error	Derivation through engineering

2 Essential Models and Model-Driven Design

Usage-centered design employs three closely related abstract models: a role model capturing salient characteristics of relationships between users and a system, a task model representing the fine structure of work users need to accomplish with a system, and an interface model representing the contents and organization of the user interface needed to support the identified tasks. Although these correspond to and may superficially resemble more traditional user and task models and paper prototypes, they differ in important ways. For example, rather than modelling users and their characteristics, user roles model relationships between users and systems, which have a more direct and immediate impact on user interface design.

Work is modelled as task cases or “essential use cases” (Constantine, 1995; Constantine & Lockwood, 1999), a specialized and highly abstract form of use cases (Jacobson et al., 1992). Use cases as employed in software engineering model functional aspects of systems: “sequences of actions, including variant sequences and error sequences, that a system, subsystem, or class can perform by interacting with outside actors” (Rumbaugh et al, 1999: 488). Task cases, rather than describing interactions with systems, model the discrete intentions of users playing roles in relation to a system, taking the form of an interrelated collection of highly simplified narratives that are abstract, implementation independent, and devoid of technological assumptions.

A usage-centered task model differs from both use cases and ubiquitous scenarios in providing a much simpler and finer-grained representation of the elemental structure of work being accomplished by users. This simplicity and granularity enables more comprehensive modelling, promotes reuse of design elements and components, and facilitates validation of models.

It is only possible to offer a bare outline of the complete process here; details are available elsewhere (Constantine and Lockwood, 1999; 2002a). The entire process is model-driven and follows a straightforward logical derivation. Even initial inquiry and field investigation is guided by exploratory models of user roles and tasks (Windl, 2002a). As in conventional object-oriented software engineering (Jacobson et al., 1992), actors are identified, but user actors are distinguished from system actors (non-humans). A role model is constructed consisting of focused descriptions of the roles played by direct users and a map of the interrelationships among roles. A task model represents all the discrete task cases needed to support the user roles along with a map of the relationships among task cases. An interface model represents the abstract contents of the various parts of the user interface along with a navigation map of the interconnections among these parts. The final visual and interaction design—or implementation model—derives more or less directly

from the interface model, particularly when the latter is expressed in canonical form (Constantine, Windl, Noble, and Lockwood, 2000). Other models are developed in parallel with the three core models, most importantly a domain model in the form of a glossary, data dictionary, or domain object model. Models from the usage-centered design process also feed the software engineering process, either directly or in altered form.

3 Impact on Software Engineering and Development

The tensions between traditional user-centered methods and software engineering and development are widely acknowledged (McCoy, 2002). 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. A cornerstone of traditional methods is usability testing with working prototypes or systems, which necessarily comes late in the development cycle, typically after an alpha (internal) or beta release. Results of usability testing, because they arrive so late in the process, are all too often ignored or deferred for later releases. When testing uncovers architectural defects or problems in the basic organization of the user interface, which is not uncommon, little if anything can be done.

Usage-centered design also imposes a substantial responsibility for up-front design, but the models themselves by nature are readily partitioned and lend themselves to iterative refinement as well as to simplification and shortcuts (Constantine, 2000). Moreover, because the entire process is inherently streamlined and simplified, the actual commitment for even the most thorough and in-depth design can itself be quite modest. For example, on the STEP 7 Lite project, a 50-person-year effort of more than two years duration, the complete usage-centered design of a user interface supporting over 300 task cases took only 14 weeks (Windl, 2002b).

4 Scalability

Usage-centered design was devised from the start to be a scalable process that could yield first-quality designs on projects ranging from a few weeks to multi-year efforts. It scales well for several reasons. Both highly simplified and fully elaborated forms of the models have been fully worked out and refined through practical application. At the one end of the spectrum are structured role models and structured task cases (Constantine and Lockwood, 2001) based on multi-faceted templates. Such structured models lend themselves to large, complex, and mission-critical applications, such as the STEP 7 Lite project cited above, and to highly organized methods, such as the unified process (Jacobson et al., 1999). At the other extreme are simple inventories that merely identify all roles or tasks. These have proved sufficient for driving effective UI design on accelerated “crunch mode” projects (Constantine, 2000; Constantine and Lockwood, 2002) and for projects using the emerging “agile” methods (Cockburn, 2002). Simple role and task inventories can be elaborated into more detailed models at any time as needed, including piecemeal or iteratively in successive short release cycles.

5 Concurrency and Integration

A range of actual projects exemplify current approaches to integration of usage-centered design with software engineering. At one extreme are complex, large scale systems developed through formal or highly structured engineering processes. Such a project was STEP 7 Lite, a complete integrated development environment (IDE) for PLC programming of automation applications

(Windl, 2002b). At the other end are small to intermediate applications developed on accelerated time frames, such as a classroom information management system for K-12 teachers (Constantine and Lockwood, 2002). In the first instance, task cases in the form of essential narratives were translated into conventional “concrete” use cases for the handoff to software engineering. Thus, although expressed in different narrative idioms, use cases became a common thread connecting the entire development process, including documentation and help system design. In addition to refinements to the object model, annotated detailed drawings of 34 user interface contexts supplemented by some 200 pages of added design documentation were delivered to the software engineers. For the educational application, developed in an accelerated XP-style iterative process, task cases were used only by the UI designers and the software engineering was guided by annotated visual designs supplemented by engineering notes and in-person communication.

Creative hybrids have also been employed with success, including in a medical informatics project in which a thorough and somewhat extended usage-centered design was followed by an agile implementation process based on XP. In this project, UI designs and task cases became the basis for so-called user stories that drove successive implementation cycles.

The STEP 7 Lite project also illustrates how concurrent engineering can reduce early-stage delays or bottlenecks. Selected internal subsystems with little or no dependence on the UI design were identified for development in parallel with the UI design. Software engineering can proceed in parallel with usage-centered design. Isolation of system actors from user actors permits independent design of system interfaces and supporting internals for all system actors. A robust architecture that insulates views and presentation layers from data structures and algorithms permits additional concurrent design and development.

Extreme programming (Beck, 2000) and other agile methods represent a new direction in highly disciplined programming practices, such as pair programming, are used within very short release cycles to deliver successively refined versions of reliable software. Card-based modelling employing ordinary index cards is a common technique used in such methods to yield simple models very quickly (Jeffries, 2001). Usage-centered models lend themselves to card-based modelling, and usable role and task models can be constructed and prioritized very rapidly (Constantine, 2002; Constantine and Lockwood, 2002).

Yet another approach, pioneered by Biddle, Noble, and Tempero (2002), is for task cases to serve without modification or translation as a direct input to the object design process. In this technique, a variant of responsibility-driven design (Wirfs-Brock and McKean, 2002) identifies objects and allocates responsibilities to objects based on the simplified abstract narratives that define each task case. It has been used successfully on a number of projects and in teaching.

References

- Anderson, J., Fleek, F., Garrity, K., and Drake, F. (2001) Integrating usability techniques into software development. *IEEE Software*, 18 (1), January/February.
- Beck, K. (2000). *Extreme programming explained*. Reading, MA: Addison-Wesley.
- 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*. Rowley, MA: Ampersand Press.
- Cockburn, A. (2002). *Agile software development*. Boston: Addison-Wesley.
- Constantine, L. L. (1995) “Essential Modeling: Use Cases for User Interfaces,” *interactions* 2 (2).

- Constantine, L. L. (2000). Cutting corners: Shortcuts in model-driven web design. *Software Development*, 8 (2). Reprinted in L. Constantine (Ed.), *Beyond chaos: The expert edge in managing software development*. Boston: Addison-Wesley, 2001.
- Constantine, L. L. (2002). Process agility and software usability: Toward lightweight usage-centered design. *Information Age*, 8 (2). Reprinted in L. Constantine (Ed.), *Beyond chaos: The expert edge in managing software development*. Boston: Addison-Wesley, 2001.
- Constantine, L. L., and Lockwood, L. A. D. (1999) *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Reading, MA: Addison-Wesley
- Constantine, L. L., and Lockwood, L. A. D. (2001) "Structure and Style in Use Cases for User Interface Design." In M. van Harmelan (Ed.), *Object Modeling an User Interface Design*. Boston: Addison-Wesley.
- Constantine, L. L., and Lockwood, L. A. D. (2002) Usage-centered engineering for Web applications. *IEEE Software*, 19 (2), March/April.
- Constantine, L. L., Windl, H., Noble, J., and Lockwood, L. A. D. (2000) From abstraction to realization in user interface design: Abstract prototypes based on canonical abstract components." <http://www.foruse.com/articles/canonical.pdf>
- Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G (1992) *Object-Oriented Software Engineering: A Use Case Driven Approach*. Reading, MA: Addison-Wesley, 1992.
- Jacobson, I., Booch, E. G., and Rumbaugh, J. (1999) *The unified software development process*. Reading, MA: Addison-Wesley.
- Jeffries, R. (2001). Card magic for managers. *Software Development*, 8 (12), December. Reprinted in L. Constantine (Ed.), *Beyond chaos: The expert edge in managing software development*. Boston: Addison-Wesley, 2001.
- Jeffries, R., Anderson, A. Hendrickson, C. (2001) *Extreme Programming Installed*. Boston: Addison-Wesley.
- McCoy, T. (2002) Letter from the dark side: confessions of an applications developer. *interactions* 9 (6): 11 - 15.
- 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*. Rowley, MA: Ampersand Press.
- Rumbaugh, J., Jacobson, I., and Booch, E. G. (1999) *The Unified Modeling Language Reference Manual*. Reading, MA: Addison-Wesley.
- Strope, J. (2002) 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*. Rowley, MA: Ampersand Press.
- Windl, H., and Constantine, L. (2001) "Performance-Centered Design: STEP 7 Lite." Winning submission, Performance-Centered Design 2001, <http://foruse.com/pcd/>
- Windl, H. (2002a) 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*. Rowley, MA: Ampersand Press.
- Windl, H. (2002b) 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*. Rowley, MA: Ampersand Press.
- Wirfs-Brock, R. J., and McKean, A. (2002). *Object design: Roles, responsibilities, and collaborations*. Boston: Addison-Wesley.