Narrative and Meaning in Ordinary Interactive Software

A position paper for the
Storytelling and Collaborative Activities Workshop
Computer Supported Cooperative Work (CSCW) 2002

William A. Stubblefield
Sandia National Laboratories
Albuquerque, New Mexico 87185
wastubb@sandia.gov

1.0 Introduction

Narrative representations have become recognized as an important tool for software design. Such design techniques as use-cases and the user stories of eXtreme Programming [26] exploit narrative's ability to represent a complex sequence of actions in a coherent, systematic structure. This is true, not only of games, storytelling systems and other directly narrative applications, but also of ordinary interactive software. Much of the power of narrative is in its ability to engage the user community at the level of its values, goals and relationships. However, if software designers are to create systems that people will find meaningful, intuitive, and useful, they must look beyond narrative form to its fundamental role in the social creation of meaning [1-7]. As designers, we are participants in this social process.

Software design projects begin with the stories users tell designers about the desired behavior of a proposed system. Although the stories users tell are always meaningful, finding that meaning requires an almost literary act of interpretation. Instead of trying to move directly from the stories people tell about their goals for a proposed system to an implementation, designers should interpret those stories in light of people's work practice, the influence of political and economic factors, the structure of meaning and relationship in their community, and the dynamic relationship between narrative, metaphor and meaning.

This paper outlines the foundations of an interpretive approach to the design of interactive software. The approach begins with the stories users tell about their work, community and expectations for a proposed system. It then draws on fieldwork, cognitive and social sciences and aesthetics to give the designer a set of tools for interpreting these stories against the proposed system's broader context.

2.0 Narrative, meaning and software design

The importance of narrative to software design has been long recognized. Laurel [25] has argued that we can think of interactive software as a form of improvisational theater. These insights are reflected in theoretical accounts of software use and design, and in the increasing number of narrative-based design methods such as use-cases, scenario based design [17] and the

user-stories of eXtreme Programming [26]. Although these methods have proven effective, the deeper structures of narrative and meaning are often ignored in software design.

Some years ago, I was in a meeting where a colleague presented a set of use cases for a proposed collaborative work system. The goal of this system was to help project leads manage a complex development process across a diverse set of engineering teams. All of the use-cases rested on the assumption that people would periodically stop work to tell the system their status within a general process model. This would enable management to get a global view of the project's progress, to look for potential problems, etc. Although the use-cases told a promising story of electronically mediated collaboration, they failed to demonstrate why a loosely knit community of independent-minded engineers would stop work and enter the desired data in the system. The use cases proposed no benefit to the engineers; they only described a system that would help management monitor them. In short, although the use-cases described a technically feasible system that met management's needs, their treatment of the end users lacked any sort of psychological plausibility. It soon became obvious to everyone that the proposed system simply wouldn't work.

Unfortunately, this story is true. All too often, designers construct use-cases, requirements or other system specifications from a literal interpretation of the stories customers and users tell about their work and their willingness to embrace information technology. In the above scenario, I can envision a focus group of managers, software designers and suitably intimidated project engineers sitting in a room filled with flip charts, nodding in dutiful agreement with the bright, rational story of software-mediated collaboration. The initial stories about what the system would do were not interpreted in light of any realistic understanding of the user community; people simply accepted them uncritically and tried to implement a software system to satisfy them.

A project in which I was involved illustrates the importance of interpreting user and customer stories. My design team was asked to develop a "traveler system" for a microsystems manufacturing process (a traveler system allows people to record actions and observations at different stages of a manufacturing process). In our early fieldwork, we noticed two distinct narrative threads in the stories customers and users told us about the desired system. The first, which came from the manager of the manufacturing group, was a "unity narrative" [8] that relied on technology to unify the diverse specialists involved in the manufacturing process. Although our prospective users publicly agreed with this vision, further fieldwork revealed a different set of stories. In individual sessions with our users, we heard something closer to the classic "hero story," in which each person stressed their own responsibility for the success of the manufacturing facility. These stories served to communicate deeper conflicts within the manufacturing staff, and people's sense of the obstacles facing the project. As we interpreted these different points of view, we better understood the deeper patterns of relationship and conflict in the user community, and were able to design a system that successfully met both management's requirement for a repository of manufacturing data, and the differing needs of the manufacturing engineers. If we had simply built the system we were asked for, I doubt the project would have succeeded.

3.0 Issues in the interpretation of user stories

The stories users tell us about their expectations for an information system should not be taken as requirements. Designers should treat them as data, needing further interpretation. Reasons for this include:

3.1 The role of implicit knowledge in communities of practice. A community of practice [9-11] is defined by long-term collaboration around a shared body of skills and goals. However, both the knowledge shared by members of these communities, and the structure of relationships in them are, to a large extent, implicit [9, 12]. Users simply cannot tell designers everything they need to know about their work and culture. Instead designers should perform systematic fieldwork in the user community to construct explicit models of the relationships, values and skills that tie it together.

3.2 Politics and economics. Introducing an information system into any organization can affect its lines of power [12, 13]. Often a proposed system can shift power among groups in the organization in unanticipated – even undesired – ways. Sources of conflict in a development effort include management's frequent goal of using technology to extend control over employees; the desire of technically savvy workers for greater autonomy and information access; gender, racial and economic stereotypes; and everyone's need for trust and privacy.

3.3 Myth and narrative stereotypes. Coyne [8] has shown that many of the stories underlying people's understanding of information technology do not reflect a realistic understanding of their own community, work and skills, so much as a romantic vision of technology's ability to solve difficult social problems. Such myths are perpetuated in popular literature on artificial intelligence, virtual worlds, artificial life and other speculative technologies, and inordinately influence people's thinking about software's capabilities. Both the high failure rate of collaborative systems and the recent failure of the "new economy" of the late 1990s underscore the power and danger of such technological myth making.

3.4 The contingent nature of meaning and personality. The stories people tell us about their work should be recognized as contextual and contingent. The meaning found in human community is not a system of "ground truths," but is influenced by history, differing points of view and the often unconscious desires of its members [9, 12]. Even individual personality should be seen as the product of a dynamic relationship between people, their history, their tools, and their position in a social context [7]. Post-modern thought has argued strongly for the recognition of meaning as dynamic and contingent [12].

3.5 The dynamics of metaphor. Narrative and metaphor are inextricably linked [4], as are metaphor and software design. However, recent years have seen a backlash against the use of metaphor in design. For example, the "desktop metaphor," once seen as the basis of windowing operating systems, now seems a quaint, rather poor representation of the actual semantics of systems like Windows and Macintosh OS X. The source of this confusion is a failure to recognize that the interpretation of metaphors inevitably changes over time [14]. Metaphor is less a foundation for defining the semantics of either human communities or computer systems, than it is a heuristic for guiding the directions in which they will grow.

4.0 Toward an interpretive design methodology

In addressing these problems, we must recognize that design is a fundamentally interpretive activity. This contrasts with the engineering model of software design characterized by such approaches as CMM (Capability Maturity Model) [15]. The most significant difference between interpretive approaches to design, and conventional software engineering is in the former's reliance on empirical fieldwork and interpretive methods to uncover the implicit patterns of meaning and relationship in a user community. Although my own efforts at developing an interpretive software design method are very much a work in progress, features of any such methodology must include:

4.1 Prototyping and Iterative Design. Iterative development methods such as eXtreme Programming [XP] [26] have demonstrated the power of using a series of prototypes to elicit user requirements. Showing users a concrete implementation of a proposed system allows them see how designers interpret their requests, and address misunderstandings in the design. Iterative prototyping methods involve two interpretive contexts: 1) designers must interpret user's initial stories of the proposed system's behavior; and 2) user's reactions to a prototype system must be correctly interpreted before the prototype can be improved. Effective application of prototyping methods requires that these interpretations be grounded in a systematic, empirical understanding of the user's community, history and values.

4.2 Fieldwork and its interpretation. Fieldwork in the user community must be the foundation of any sort of interpretive design methodology. The basis for interpreting user stories must be a systematic, empirical understanding of people's work and community. Achieving this understanding involves a variety of disciplines such as the anthropology of work, cognitive science, ethnographic methods [18, 19] and theories of human work and practice. Specific techniques include distributed cognitive analysis [16], scenario-based design [17], design ethnography [18, 19], and contextual design [20]. Theoretical tools for interpreting field observations include cognitive science, interpretive approaches to the social sciences [21], activity rheory [22], information design [23] and the use of prototyping methods to test design assumptions.

4.3 Participatory design. Although participatory design [19] is often defined as including users in the design process, its application is far from simple. Design is a specialized activity: experienced designers should manage the process and make key design decisions. Participatory design should involve users in those decisions that affect their well being and community, while letting trained designers manage this dialog and make basic design decisions.

4.4 Aesthetics. Although the growth of the World Wide Web has brought attention to the importance of aesthetics in user interface design, I still see far too many interfaces designed as rows of data input boxes on a plain gray background. Interaction design, the definition of the patterns of action and experience the tool supports, is even more neglected. If we think of software use as interactive narrative, the stories it tells tend to be rather dull. In a recent article [24] (and forthcoming book), Donald Norman has acknowledged the importance of aesthetics to the ease of use of designed objects. Similarly, Brenda Laurel [25] has examined the role of poetics in the design of interactive software. Given that interaction design is fundamentally about creating narrative descriptions of desired system behavior, designers would be foolish to

ignore the ability of good stories to engage people. We should treat use-cases and other descriptions of system functionality as stories. In creating them, we should devote the same attention to motivation, characterization, and consistency of action and consequence that we expect of good fiction.

5.0 Ramifications for software implementation and architecture

The impact of narrative on interactive software cannot be restricted to early design work. Narrative and interpretive methods also have ramifications for software implementation and architecture.

5.1 Narrative-based architectures. Although interactive software has a narrative structure, that structure is seldom made explicit in its implementation. Usually, it is implicit in the code that defines interfaces, algorithms and data-structures. Even the Model-View-Controller architecture, which is widely used to implement user interfaces, often describes system behavior in terms of programming language-specific events, rather than in terms of user narratives. My colleagues and I have explored interactive architectures that explicitly represent system behavior in higher-level terms more directly relevant to user intentions. We have implemented two of these architectures: a finite-state machine controller that models interactions as transitions through a state space; and a call-response model that implements interfaces as assemblies of general interaction patterns. Both have resulted in robust, maintainable software.

5.2 Interpretation and agile methods. By emphasizing the use of narrative methods, close, ongoing contact with users and an iterative approach to design, agile methods such as eXtreme Programming [26] provide a good starting point for an interpretive design methodology. However, many such approaches underestimate the importance of interpreting user stories in the broader context of their community, skills and history. For example, XP advocates a design cycle that starts with a set of narrative descriptions of system behavior called "user stories." All subsequent implementation and testing are aimed at satisfying these user stories. Typically, these user stories are the product of informal brainstorming sessions, with no explicit effort to interpret them in light of observational fieldwork, social science methods or cognitive analysis. Without such analysis, we run the risk of designing to superficial, technological stereotypes, and missing the deeper processes of meaning making in the user community.

5.3 Validation. The classical model of software validation holds that software should be validated against the initial requirements that guided its design. Interpretive methods challenge this view by characterizing requirements as the starting point of an interpretive process, rather than a rigid specification for system behavior. The interpretation of requirements changes over time and through their interactions with the development process itself. This observation has at least two ramifications for software validation: 1) User reviews, usability evaluations and other people centric approaches must be the central focus of the validation effort; and 2) because tests designed under one set of assumptions may not capture all patterns of system use, designers should place greater reliance on analytic methods, such as assertions, invarients and formal arguments of program correctness. The traditional approach of testing against requirements will remain part of the validation process, but only within a broader context of user-oriented and analytic methods.

6.0 Conclusion

This paper has only started to outline an interpretive methodology for designing ordinary interactive software. However, increasing recognition of the importance of narrative to software interaction design, and a deeper understanding of the position of narrative in the social construction of meaning underscores the importance of interpretive methods. Paradoxically, the empirical discipline of fieldwork and the interpretive tools of art, design and the social sciences that unleash the designer's creativity. Quality software rests on a foundation of good stories, well

7.0 References

1. Bruner, J., *Acts of Meaning*. 1990, Cambridge, Mass: Harvard University Press.

2. Bruner, J., *The narrative construction of reality.* Critical Inquiry, 1991. **18**(1): p. 1-12.

3. Campbell, J., *The Hero With a Thousand Faces*. 1949, Princeton, NJ: Princeton University Press.

4. Lakoff, G. and M. Johnson, *Metaphors We Live By*. 1980, Chicago: University of Chicago Press.

5. Murray, J., *Hamlet on the Holodeck: The Future of Narrative in Cyberspace*. 1997, New York: The Free Press.

6. Sengers, P., *Narrative Intelligence,* in *Human Cognition and Social Agent Technology,* K. Dautenhahn, Editor. 2000, John Benjamins Publishing Co.: Amsterdam/Philadelphia.

7. Turkle, S., *Life on the Screen: Identity in the Age of the Internet*. 1995, Cambridge, Mass: MIT Press.

8. Coyne, R., *Technoromanticism: digital narrative, holism and the romance of the real*. 1999, Cambridge, Mass.: MIT Press.

9. Lave, J. and E. Wenger, *Situated Learning: Legitemate Peripheral Participation*. 1991, Cambridge: Cambridge University Press.

10. Wenger, E., *Communities of Practice: Learning, Meaning and Identity*. 1999, Cambridge: Cambridge University Press.

11. Brown, J.S. and P. Duguid, *The Social Life of Information*. 2000, Boston: Harvard Business School Press.

12. Coyne, R., *Designing Information Technology in the Postmodern Age*. 1995, Cambridge, Mass: MIT Press.

13. Stubblefield, W.A. *The Social Life of Engineering Authorizations*. in *Designing Interactive Systems (DIS-2000)*. 2000. Brooklyn, New York:

14. Stubblefield, W.A. *Patterns of Change in Design Metaphor: A Case Study*. in *Computer-Human Interaction (CHI-98)*. 1998.

15. Paulk, M.C., C.V. Weber, and B. Curtis, *The Capability Maturity Model: Guidelines for Improving Software Process*. 1995, Addison Wesley.

16. Hutchins, E., *Cognition in the Wild*. 1995, Cambridge, Mass.: MIT Press.

17. Carroll, J.M., *Making Use: Scenario-based design of human-computer interactions*. 2000, Cambridge, Mass: MIT Press.

18. Suchman, L.A., *Plans and Situated Actions: The Problem of Human Machine Communication*. 1987, Cambridge: Cambridge University Press.

19. Schuler, D. and A. Namioka, *Participatory Design: Principles and Practices*. 1993, Hillsdale, NJ: Lawrence Erlbaum Associates.

20. Byer, H. and K. Holtzblatt, *Contextual Design*. 2002, Morgan Kaufmann Publishers.

21. Geertz, C., *The Interpretation of Cultures*. 1973, New York: Basic Books.

22. Nardi, B.A., *Context and Consciousness: Activity Theory and Human-Computer*