Structure versus Context: Understanding the Design and Use of Computer Tools in Social Settings

Kevin Powell

ABSTRACT
This article explores elements of folkloristics that are applicable to the understanding and construction of computer software systems and programs. This is done by first discussing a similar tension between structuralism and contextualism in the study of folkways and, second, by examining software pattern languages (Gamma, Helm, Johnson, & Vlissides, 1995) in comparison with Vladimir Propp's (1968) narrative functions.

INTRODUCTION
A hammer is a good tool, well honed and continually adapted for millennia. In some ways it represents the ultimate in a tool but, in a very real sense, it demonstrates a tradeoff in how tools are used and adapted in context versus the broader use of a hammer as a general tool. The importance of the hammer is that it is broadly used outside its intended design and, even when it is being used inside its designed or designated realm, it still may not fit the tasks for which it is being used very well.

An example of this might be when it is needed in a tight corner at the intersection of two walls. It is possible to build a special hammer that would operate optimally in such a tight situation, but the economics of the situation do not justify this new tool (i.e., the tool is not general enough to sell large numbers, and the cost of the tool is not low enough to make its purchase viably incidental). Almost any artifact produced for broad consumption faces the dilemma of fit to task versus cost of development and maintenance. One of the ways to ameliorate this problem is to try to
understand the context of use more clearly when designing a tool. Folkloristics, with its investigation of both cross-situational structure and contextual understanding of folkways, suggests ways to consider electronic tools in a community context.

Based on the metaphor of software as a tool, this article focuses on an understanding of how software is used in context and the difficult nature of building such tools contextually in a world of limited resources.

FOLKLORE AND THE DESIGN OF COMPUTER ARTIFACTS

The hammer is a tool familiar to many folk groups and is a part of different folkways. Not surprisingly, computer software is equally embedded in current folkways. Furthermore, many systems have failed because they do not take into account the folkways in which they are to be embedded (Star & Ruhleder, 1996; Gasser, 1986). This lack of attention to the context of use becomes a major hindrance to the acceptance of these systems. In a sense, the designs of many systems are coming from a structuralist viewpoint—i.e., they are designed for broad categories of use and gloss over, or ignore, the context of the particular setting.

STRUCTURALISM/CONTEXTUALISM AND SOFTWARE DESIGN

A folklorist who limits his analysis to identification [of motifs/patterns] has stopped before asking any really important questions about his material. (Dundes, 1990, p. 52)

Alan Dundes starts his discussion of the study of folklore in literature and culture by trying to resolve the dichotomy of interpretation and identification. He does so by treating interpretation and identification as vitally complementary tools. When used together they give both the anthropological and textual folklorist a much more rigorous understanding of the materials they study. The anthropological folklorist can run the risk of misidentification of a folktale or element because of an unfamiliarity with the vast identified body of cross culturally analyzed materials which she may see as being unique to a context of activity or expression. Conversely, the textual folklorist errs when she identifies materials as being just another variant, losing all the vitality and particular meaning of the material in its original context of use. This argument applies equally to the design realm of software tools. Designers have fixated on the more readily quantifiable aspects of systems, largely ignoring the context of their use and the potential power of a tool properly fit into a set of tasks and folkways.

THE STRUCTURALISTS SPEAK

[W]hat gives the myth an operative value is that the specific pattern described is everlasting; it explains the present and the past as well as the future. (Lévi-Strauss, 1972, p. 173)
Whatever our ignorance of the language and culture of the people where it originated, a myth is still felt as a myth by any reader throughout the world. (Lévi-Strauss, 1972, p. 174)

The above quotes are meant to speak for an intellectual traditional structuralism that seeks to understand narrative forms in terms of an overarching set of structures. Lévi-Strauss and others take this analysis to its logical extreme, arguing that—like language as studied by linguists—narratives can be decomposed into elements that can be analyzed across vast gulfs of culture and geography. The power of abstracting away so many details to expose a core narrative is the production of an elemental currency that can be exchanged across many different arenas of discourse. This sort of analysis provides us with a framework that is more amenable to the designer of software due to its highly structural abstractions than are qualitative/situated methods of analysis. However, a purely structural approach is fundamentally unable to provide us with a full appreciation of narrative and setting. It fails in elucidating the narrative’s full role and uses, especially the pragmatics of its day-to-day existence as a living cultural tool.

**The Contextualists Speak**

We must recognize that the symbolic forms we call folklore have their primary existence in the action of people and their roots in social and cultural life. The texts we are accustomed to viewing as the raw materials of oral literature are merely the thin and partial record of deeply situated human behavior. My concern has been to go beyond a conception of oral literature as disembodied superorganic stuff and to view it contextually and ethnographically, in order to discover the individual, social, and cultural factors that give it shape and meaning in the conduct of social life. (Bauman, 1986, p. 2)

As Dundes (1990) points out, there has been a long, and sometimes bitter, dialogue between the contextualists and the structuralists. This dialogue is useful in that it also frames a discussion of how to understand the design of software systems across several different analytical perspectives. Bauman’s quote emphasizes the situated nature of folklore and, by extension, folkways. Again, this framing has meaning beyond the scope of folklore and into the scope of software systems.¹

These points are especially salient in terms of software explicitly tailored for collaborative use. Software designed explicitly for use as a social tool needs to focus almost entirely on the underlying issues of context and use in order to be successful. Only recently has the mismatch between the design of software tools and their settings become an area of active research. This article suggests that the methodology of folkloristics may be useful in understanding how to design systems with respect to their settings.
SOFTWARE DESIGN PRACTICE

Designing computer software for a particular task, or a set of tasks, involves a long series of complex negotiations among a number of different factors and interests. Traditionally, computer-related disciplines have identified the primary factors to be performance, reliability, and cost. Rarely do we see factors such as usability and fit-to-task figuring into these negotiations. This emphasis on the former factors stems from the need to legitimate the practice of software design as a science; as such, quantifiability and reproducibility are pushed to the fore. In addition, the simultaneously more difficult and ambiguous nature of studying usability and work practice as related to a software system has strongly limited efforts to include analysis of social context in traditional software design.

Designing computer software is, by its nature, a highly structured activity; in turn, this mitigates against designing software that is easily adaptable to differing contexts. Alternatively stated: One of the central challenges faced by software designers is how to balance the highly structured nature of computer artifacts with the need to integrate them into different settings. Several different communities of scholars are working on this set of problems. Of these groups, the Computer Supported Cooperative Work (CSCW) and Participatory Design (PD) communities are most directly addressing the question of building tools for use in particular settings. CSCW focuses on designing and understanding computer tools as used in and by groups. PD is interested in involving the users in the design of the system itself.

Both these communities approach the problem as a matter of correct design. Computer Supported Cooperative Work offers techniques for collaboration and, in certain limited cases, techniques for evaluating the effectiveness of the tools. Participatory Design is less concerned with the design elements and technology; rather, it emphasizes the need for the user's voice in the design of the software (Engeström, 1990). These two approaches often use ethnographic techniques to explore the meaning of a tool in its context of use. However, neither addresses how the design of these tools can be analyzed directly in terms of the structures of the artifacts themselves rather than in the fits/misfits with the users. One role folkloristics can play in the analysis of these artifacts is to examine them in terms of their structure and functions, just as traditional tales are analyzed and classified by their shared use of motifs and functions (Dundes, 1990; Propp, 1968). Folklorists also contribute to an understanding, or perhaps reconciliation with a continuum between structural understandings at one end and contextualized/situated knowledge at the other.

Designing for folk groups or communities is an extremely difficult problem; however, designers often face the challenge of designing for nontraditional communities as well. Electronic communities confound
this already difficult problem by introducing a number of mediating factors that further complicate the process of design. Electronic communities offer a number of challenges for researchers of folkloric processes. Such communities are set in what might be thought of as an abstract, or perhaps intangible, domain; ultimately they exist through the actions and in the imaginations of the participants. The everyday material lore of more conventional folk groups is not to be found in this electronic realm. It is difficult to find a central locus, or set of loci, of activity. The boundaries are defined and maintained only through the weak ties of e-mail, netnews, and other computer mediated communications tools (Pickering & King, 1992).

The traditional idea of community has been based on the notion of a co-located, regional, or area-bounded group (Cohen, 1985; Jones, 1995; Ried, 1995). This notion quickly breaks down when applied directly to electronic communities or other distributed folk groups. For the sake of discussion, this article adopts a fundamental notion of community—that of a collection of individuals who share some common interest. This notion, while highly simplistic, frames and limits the rich, but endless, dialogue on what constitutes a community or folk group. In turn, this leads to a useful reexamination of the notion of co-location, which can be effectively redefined as the notion of individuals together in some “space” bound not by physical location but by shared interests. In essence this is a redefinition of “community space” where the spaces are potentially virtual, textual, or in other forms.

In trying to understand software as a situated set of tools, we are faced with a number of daunting issues:

- the over-determined nature of software design: its inherently structural nature;
- the forces which have shaped the research interests in favor of quantification over qualification; and
- the difficulty of understanding the complexities of community and folkways to better fit the software produced for their needs.

**PROGRAMMING FOLKLORISTICS**

It is important to distinguish between two forms of folklore that are active when software is produced. The first form is the familiar, more traditional, form of folklore, or folkways, consisting of the interactions between the programming team's members and others. Programmers are embedded in many contexts of activities and, even if we limit our scope to just those folkways directly related to their work practices, we cannot fully address the role such practices play in shaping the ultimate product of their labors. The focus of this section, then, will be on a new approach to understanding the structure of programs. This approach bears some resemblance to the basic form of the structural analysis of text.
Programming Folkways: "Pattern Languages"

"Pattern language" is a term coined by Christopher Alexander (1977), an architectural researcher. His work centers on ways to express a language used to build and organize structures of all scales, ranging from single family homes to whole cities. A number of people researching software engineering have adopted the thrust of Alexander's work to construct pattern languages of reusable software designs. These designs can be considered to be roughly analogous to the basic "functional" units Propp (1968) discusses.

In the patterns derived by Gamma et al. (1994), we see that the exact meaning of the individual entries is not strictly important for this discussion (see Table 1). What needs to be emphasized about these entries is their relative sparcity and the lack of interconnection between the individual patterns themselves. For instance, the Singleton pattern represents a situation in which only one member of this entity is allowed to exist at any one time. This is a common occurrence when building objects which coordinate the actions of many other objects. However, no satisfactory structure has been found to tie this low-level pattern explicitly to a broader set of tale types or program types, if you will. By comparison, Propp's (1968) Morphology of the Folktale is a much better elucidated breakdown of how one understands a narrative (see Table 2).

Table 1.
Listing of Patterns Found in Gamma et al. (1994).

<table>
<thead>
<tr>
<th>Creational Patterns</th>
<th>Structural Patterns</th>
<th>Behavioral Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract Factory</td>
<td>Adapter</td>
<td>Chain of Responsibility</td>
</tr>
<tr>
<td>Builder</td>
<td>Bridge</td>
<td>Command</td>
</tr>
<tr>
<td>Factory Method</td>
<td>Composite</td>
<td>Interpreter</td>
</tr>
<tr>
<td>Prototype</td>
<td>Decorator</td>
<td>Iterator</td>
</tr>
<tr>
<td>Singleton</td>
<td>Façade</td>
<td>Mediator</td>
</tr>
<tr>
<td></td>
<td>Flyweight</td>
<td>Memento</td>
</tr>
<tr>
<td></td>
<td>Proxy</td>
<td>Observer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>State</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Strategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Template Method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Visitor</td>
</tr>
</tbody>
</table>
Table 2.
The Initial Situation: The first nine of twenty-four functions (adapted from Propp, 1968).

The Initial Situation

1. Temporal-spatial determination ("in a certain kingdom").

2. Composition of the family:
   a. according to nomenclature and status;
   b. according to categories of dramatis personae (dispatcher, seeker, etc.)

3. Childlessness.

4-5. Prayer for the birth of a son:
4. Form of the prayer;
5. Motivation of the prayer.

6. Cause of pregnancy:
   a. intentional (a fish which is eaten);
   b. accidental (a swallowed pea, etc.);
   c. forced (girl is abducted by bear, etc.).

7. Form of miraculous birth:
   a. from a fish and from water;
   b. from a hearth;
   c. from an animal;
   d. otherwise.

8. Prophecies, forewarnings.

9. Well-being, prior to complication (zavjázka):
   a. fantastic;
   b. domestic;
   c. agrarian;
   d. in other forms.

Propp’s functions serve to structure and move a narrative along its path to completion. This analytical thrust is somewhat at odds with the approach of Alexander (1977) and Gamma et al. (1994), for their patterns serve primarily a structural role and, to a much lesser extent, can be recombined to form a full narrative. Additionally, Gamma’s patterns are also highly discrete and therefore even more difficult to combine to make a higher order meaning/narrative. Ultimately, Gamma’s identified patterns suggest that some higher order of structural analysis can be performed on programs. Furthermore, the structural analysis has, so far, been limited to very small units of structure and has proven difficult to extend into a higher order of understanding of a program’s meaning. By contrast, Propp’s functions have clear ordering and grouping properties. It is difficult to directly determine if such properties could be discovered in terms of software design. If this level of structure could be reliably
derived from source code, it would allow for a much deeper and more rigorous understanding of software systems.

**Programming as Narrative Construction**

Analyzing programs and software systems involves understanding the higher order or conceptual meaning of the program. This is often thought of as the intent of the programmer, or programmers, of that system. Programs are typically structured by a functional or object-oriented breakdown of the particular task being designed for. This analysis requires a basic understanding of the problem domain; then the designer iteratively divides the tasks required to model the problem domain into a set of methods that implement a particular domain's requirements.

This sort of iterative creative process is akin to the creative process that goes into telling a new tale. However, most tales are not created whole cloth from the teller's mind. They are a dynamic evolving co-constructed narrative that uses themes and motifs honed over time (Bauman, 1977; Toelken, 1996). It is not surprising, then, that software construction is more akin to trying to tell a story without the benefit of a well-honed repertoire of themes, motifs, and tales from which to work. Like the early scholars studying folklore, scholars studying computer programming have yet to discover a method, or set of methods, to understand programs and their structures across different domains. This problem of a lack of cross-contextual analysis method suggests that looking at a program as a sort of narrative may help in understanding its broader meanings.

**Narrative and Language**

Construction of even such a simple tool as the hammer (which itself is a nonhomogeneous category) is considerably complicated by its various contexts of use. The construction of software is situated in a very complex confluence of the plasticity/manipulability of verbal art and the constraints of the physical material composing a hammer. This tension is often exacerbated by the common misconception of thinking of computer languages as real (or natural) languages.

The performance of a newly composed story involves the artful recombination of motifs and elements in new and innovative ways. More generally and just as importantly, any performance of a narrative involves an understanding of the audience hearing the story as well and the effects of the feedback they provide to the process itself (Bauman, 1978; Goodwin, 1982). Procedurally, this construction of narrative resembles the construction of software. In addition, both verbal art and software require an understanding of the motifs in a particular domain of discourse and activity. Of critical importance is the difference in the degree of plasticity in the "languages" used for the construction of these different narratives.
Computer languages, while having some degree of ambiguity in the interpretation of their meaning, usually are intended to be completely unambiguous. Ambiguity is considered to lead to any number of errors in the construction of software. Not surprisingly then, ambiguity is hunted down and eradicated in the language's grammar or in its interpretation. Many of the historical trends in language design can be seen moving toward elimination of ambiguity from any program's meaning. This reduction of ambiguity is pragmatically useful, because ambiguity in the meaning of a program almost certainly leads to the incorrect function of that program. So, unlike the storyteller's case, the basic form of the computer language used acts as a strong limiter on the "artfulness" that can be expressed directly in a program's text.\(^8\)

While computer languages are designed with grammars that minimize ambiguity, simultaneously they are a very careful mixture of maximum meaning in natural language terms while preserving the simplest, absolute unambiguous meaning in terms of the computer's interpretation of it. In practice, this means that the programmer can express the "natural" meaning of a particular task by using descriptive names and conventions when writing a program. This represents the most basic level in which the microstructural, or unambiguous instructional, meaning of the program is integrated with the human-conceptual meaning of the program's text. This is the starting point in which "artfulness" of expression is found in programming. It is in the further structuring of these smaller units of function/meaning that the overall conceptual meaning of the program is made. At this point, we have a rough equivalent of the motif or function as seen in the classical structuralist analysis of narratives. In summary, we have a number of narrative analogies:

- like folktales and their telling, programmers, in a very loose sense, tell "stories" in code;
- these stories are only really interpretable by other programmers who "speak" the same language and are in the same, or related, folk groups;
- as discussed earlier, a number of groups in the computer science community have been analyzing code to discover "patterns"; these patterns can be loosely thought of as equivalent to motifs or functions;
- understanding the practices of the folk groups and communities for which a software system is designed is vital for the success of that system. Without this understanding, which is equivalent to storytellers' understanding their audiences, the most likely outcome is failure or under-utilization of the system.

**Conclusion**

Folkloristics offers some hope that a method or methods can be found or borrowed that will enable software engineers to understand the fine
grained structure of both programs and systems. Furthermore, if successful, this form of analysis might provide the tools to reveal a new understanding of a broader typology of programs as well as aid in their construction. The greatest challenge lies in identifying and adapting the many techniques used by folklorists that would be applicable to software design. This process will require a deep understanding of both fields—unfortunately, a very rare combination. If such a process was undertaken, the potential benefits could be quite extensive. Given the state of the art in terms of pattern languages, a great deal more can be done to aid in the construction of good programs.

Even without a revolution in understanding and analyzing programs as specialized folklore or narrative, simply adopting some of the philosophy of folklorists would be helpful for designers of systems for use in social settings as social systems. Sensitizing designers to the notions of the situated nature of folkways and knowledge would enable them to design artifacts that were, at a minimum, less poorly mismatched to social settings and practices. A true mixture of design and social pragmatics would provide the designer with a new set of tools that can begin to answer questions that are currently impossible to grapple with in the default quantitative manner.

Suggestions for Building Software

As a beginning to this process of integrating folkloristics and software development, a few suggestions for building software are presented here. These suggestions are only starting points and introduce as many questions as they address. However, even rough guidelines are often better than none, and these are offered in that spirit:

- provide generic small tools, proto-tools (i.e., build a few types of hammers, not a nail gun);
- allow for the tailoring and recombination of these proto-tools. They are partially able to be tailored because of their simplicity, but they must also be able to be explicitly adapted through their underlying structures;
- fit these tools into a loosely knit overarching structure, a structure composed of a number of loosely coupled interrelated substructures; and
- examine other systems for motifs/functions in software systems.

Of these points, the last deserves special attention. Examining other software systems for patterns/motifs/functions may provide the most benefits to the average programmer. All of these guidelines collectively point the way toward a new method of structuring software after a model that attempts to integrate both the contextual and the structural.
NOTES

1 Alternatively, as a colleague suggested: Perhaps software systems must be included as a “record of deeply situated human behavior” that reflects contemporary folkways.

2 A large part of CSCW and PD analysis of tools is in terms of breakdowns in the use of a tool or set of tools. Very rarely do we see an analysis of the tools in terms of their structures in relationship with their context of use. In a simplified form this can be thought of as the difference between saying what is wrong with a system, and what is right with a system and what parts are responsible for this success.

3 There are a number of studies on e-mail which discuss usage patterns but do not directly deal with how this medium is used to create and maintain community (see Bizot et al., 1995; Kiesler et al., 1984; Siegel et al., 1986). They do provide a useful backdrop to frame how many have approached studying CMC (i.e., a tool-dominated approach with an overemphasis on the medium). Similar studies on usenet newsgroups (Baym, 1995a, 1995b) support this as well.

4 Gamma et al have done the most thorough job to date recognizing these broader structures. The patterns community is still working on how best to understand and determine the larger patterns that are present in many programs and systems.

5 Although Propp’s functions are perhaps no more easily combined to produce an artful narrative.

6 The pattern languages emerging in the software design realm are a first attempt at providing a repertoire of “motifs” for the programmer to work from.

7 The non-homogeneous category is meant to reinforce the myriad roles and meanings any artifact has in different contexts, i.e., it will be placed into a number of differing categories which are not necessarily inter-compatible.

8 This is not to say that natural languages are infinitely plastic. The basic grammatical structure of a language must limit the ambiguity of an utterance; otherwise meaningful communication would be nearly impossible. However the degree of flexibility seen in a natural language is so much greater than that of a computer language that even at the level of single utterances the flexibility may as well be infinite when viewed in contrast with computer languages.

REFERENCES


