Usage-Centered Engineering for Web Applications
Larry L. Constantine | Lucy A. D. Lockwood
Constantine & Lockwood, Ltd.
lconstantine@foruse.com | llockwood@foruse.com

Abstract: This paper presents a lightweight form of usage-centered design that has proved particularly effective in designing highly usable Web-based applications. Fully compatible with both traditional object-oriented software engineering methods and newer agile techniques such as Extreme Programming, this approach employs rapid, card-based techniques to develop simplified models of user roles, tasks, and user interface contents. The process attempts to resolve the conflict between the demands of rapid iterative design and incremental development on the one hand and the needs for integrity in a user interface fitted to the full set of user tasks on the other. The resolution depends on creating a navigation architecture and a visual and interaction design scheme based on quick but comprehensive task modeling. The process is illustrated with experiences from the design of a Web-deployed application for classroom teachers.

Keywords: usability, Web applications, usage-centered design, agile methods, lightweight methods, extreme programming, iterative development

Learn more about usage-centered design at http://www.forUse.com.

Introduction—Old is New

Despite breathless declarations that the Web represents a new paradigm defined by new rules, professional developers are realizing that lessons learned in the pre-Internet days of software development still apply. They are recognizing that Web pages are user interfaces, HTML programming is programming, and browser-deployed applications are software systems that can benefit from basic software engineering principles.

Nevertheless, development for the Web is distinguished in a number of ways. Although essentially anything that can be achieved in software can be delivered over the Web in a browser instance, almost everything is more difficult. A lack of standards and conventions makes development and deployment messier and more complicated. Inconsistencies among the various browser-platform-connection combinations mean reduced control over what the user actually sees and experiences. In one project, fully 70% of the programming effort went to getting supposedly standard features to work properly on one browser (Netscape) and one platform (Apple Macintosh) that together accounted for less than 10% of the customer base. Adding to the chaos, design styles
on the Web are highly varied, and even acknowledged conventions, such as a left-hand column for primary navigation or styling links as underlined blue text, are often ignored.

Three characteristics come to the fore in Web development: the acceleration of the development process, the prevalence of “green-field” applications with few if any standards or precedents, and the importance of the user experience. Nearly every project faces shortened product release cycles and escalating pressure for timely delivery, but the Web redefines the notion of accelerated development and deployment. “Web-time” [Thomas, 1998] has become synonymous with headlong desperation and virtually impossible deadlines [Constantine, 2000a]

Systematic and disciplined practices offer some hope, but slavish adherence to traditional methods may be problematic. Traditional approaches, with their plethora of documents and deliverables and emphasis on analyzing established practices and existing systems may not be well-suited to Web-time delivery of novel applications. Furthermore, with few exceptions, user interface design and usability do not figure prominently in modern software engineering approaches and are acknowledged as weak points in both the “heavyweight” processes [Constantine, 2000b], such as the so-called Unified Process, and the lightweight competitors [Constantine, 2001a], such as Extreme Programming [Beck, 2000; Jeffries, 2001].

This article describes a flexible, model-driven approach for engineering Web applications that succeeds through a focus on user interface design and usability. Its simple, model-driven techniques are well-suited to novel applications and integrate readily with various “lightweight” or “agile” development processes [Fowler, 2000] carried out under compressed development schedules.

### Web Applications for Use

Usability and the user experience are emerging as critical determinants of success in Web applications. If customers cannot find what they are looking for, they cannot buy it; if key information is buried, business decision making is impaired. Poorly designed interfaces increase user errors, which can be costly. Mistakes made entering credit card billing information, for example, can require costly manual follow-up or lead to lost sales. Web-usability guru Jakob Nielsen estimates that billions of dollars in lost Web sales can be traced to usability problems.

Technical and customer support costs in all areas of business are skyrocketing. Every time usability problems on a site or application prompt a telephone call or an email message to the help desk or customer service, an inexpensive Web session becomes an expensive support incident. On corporate intranets, hard-to-use applications will simply not be used at all or will require extensive training.

Although the focus in this article is on Web-based applications—systems in which functional capability is deployed on the Web through a browser instance as a thin client—it is difficult to draw a hard and fast line separating Web-based applications from those Web sites on the Internet that are intended for use, that is, for something more than entertainment, casual browsing, or building brand identity. The common thread is the delivery of useful services and capabilities over the Web.

In addition to usability, aesthetic aspects of design may also figure in the user experience. Unfortunately, graphic design and aesthetic considerations all too often are
established early and drive the entire process at the expense of usability. For example, based on an “aesthetic brief” approved by the client’s CEO, a prominent graphic design house developed an innovative Web site for a major analyst group. Despite widespread exposure of the design concept to selected customers and the conduct of numerous market-oriented focus groups, the design proved to be a usability disaster, in no small part because of the aesthetic design. Last ditch efforts to improve the design without modifying the basic graphic design enabled the site to “go live” but ultimately proved to be only palliative and insufficient to satisfy customers. In the end, the firm undertook a complete redesign in-house based on usage-centered design.

Usage-Centered Design
Usage-centered design [Constantine and Lockwood, 1999] is a systematic process using abstract models to design the smallest, simplest system that fully and directly supports all the tasks users need to accomplish. First developed in the early 1990s, it is a proven, industrial-strength approach that has been used to design everything from industrial automation systems and consumer electronics to banking and insurance applications. Because it is a streamlined process driven by simple models, it scales readily and has been used on projects ranging from a few person-months to a 5-designer, 19-developer, 23-month project that produced Step7lite, a sophisticated new integrated development environment from Siemens AG [Constantine, 1999; Windl and Constantine, 2001]. On the Web, the techniques have been successfully employed by groups around the globe for a wide assortment of applications, including e-commerce, membership support, education, and medical informatics [Anderson et al., 2001].

User-centered or usage-centered?
Usage-centered design evolved as a software engineering alternative to user-centered design. Table 1 summarizes some of the salient differences. User-centered design [Norman and Draper, 1986] is a loose collection of human factors techniques united under a philosophy of understanding users and involving them in design. It relies primarily on three techniques: user studies to identify what users want, rapid paper prototyping to get user feedback on successive user interface design iterations, and usability testing with users to identify usability problems in working prototypes or systems. Although useful, none are substitutes for good design. User studies too easily confuse what users want with what they truly need. Rapid iterative prototyping is too often a sloppy substitute for thoughtful and systematic design. And usability testing is a relatively inefficient means of finding problems that often could have been avoided through proper design [Constantine and Lockwood, 1999; Parush, 2001].
Table 1 - Comparison of User-Centered and Usage-Centered Design

<table>
<thead>
<tr>
<th>User-Centered Design</th>
<th>Usage-Centered Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus is on users: user experience and</td>
<td>Focus is on usage: improved tools supporting</td>
</tr>
<tr>
<td>user satisfaction</td>
<td>task accomplishment</td>
</tr>
<tr>
<td>Driven by user input</td>
<td>Driven by models and modeling</td>
</tr>
<tr>
<td>Substantial user involvement</td>
<td>Selective user involvement</td>
</tr>
<tr>
<td>- User studies</td>
<td>- Explorative modeling</td>
</tr>
<tr>
<td>- Participatory design</td>
<td>- Model validation</td>
</tr>
<tr>
<td>- User feedback</td>
<td>- Usability inspections</td>
</tr>
<tr>
<td>- User testing</td>
<td></td>
</tr>
<tr>
<td>Design by iterative prototyping</td>
<td>Design by modeling</td>
</tr>
<tr>
<td>Highly varied, informal, or unspecified</td>
<td>Systematic, fully specified process</td>
</tr>
<tr>
<td>processes</td>
<td></td>
</tr>
<tr>
<td>Design by trial-and-error, evolution</td>
<td>Design by engineering</td>
</tr>
</tbody>
</table>

Driving models

Usage-centered design is driven by three simple, closely related abstract models: a role model, a task model, and a content model. The role model captures the salient characteristics of the roles that users play in relation to a system. The task model represents the structure of the work users need to accomplish in relation with a system. The content model represents the contents and organization of the user interface needed to support the identified tasks.

A user role represents one kind of relationship users could have with a system or Web site. Among numerous possible aspects of this relationship are the purpose and frequency of interaction, the volume and direction of information exchange, and the attitude toward the system of users in the role.

A task model consists of a set of task cases and a map of the interrelationships among those task cases. Task cases are a form of use cases [Jacobson et al., 1993]. Conventional use cases are models of systems, namely, the “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]. As commonly written, use cases express the concrete actions and responses taken by the system and an actor with which it interacts. Figure 1, a published example [Kruchten, 1999] of a conventional use case, tellingly omits the user’s single most important step: actually taking the cash.
Withdraw Money
The use case begins when the client inserts an ATM card. The system reads and validates the information on the card.

1. System prompts for PIN. The client enters PIN. The system validates the PIN.
2. System asks which operation the client wishes to perform. Client selects "Cash withdrawal."
4. System requests type. Client selects account type (checking, savings, credit).
5. The system communicates with the ATM network to validate account ID, PIN, and availability of the amount requested.
6. The system asks the client whether he or she wants a receipt. This step is performed only if there is paper left to print the receipt.
7. System asks the client to withdraw the card. Client withdraws card. (This is a security measure to ensure that Clients do not leave their cards in the machine.)
8. System dispenses the requested amount of cash.
10. The use case ends.

Figure 1 – Conventional use case for getting cash from an ATM [Kruchten, 1999]

Figure 2, a task case written in the simplified form employed in agile usage-centered design, is shorter, simpler, and more closely represents what the task is truly about for the user. Task cases—also called essential use cases [Constantine, 1994; Constantine, 1995; Constantine and Lockwood, 1999]—are “essential” models [McMenamin and Palmer, 1984] that are abstract, simplified, and technology-free—without built-in assumptions about the user interface.

<table>
<thead>
<tr>
<th>getting cash</th>
<th>SYSTEM RESPONSIBILITIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER INTENTIONS</td>
<td>1. Request identity.</td>
</tr>
<tr>
<td>2. Identify myself.</td>
<td>3. Verify identity.</td>
</tr>
<tr>
<td>5. Make choice.</td>
<td>4. Offer choices.</td>
</tr>
</tbody>
</table>

Figure 2 – Abstract use case (task case) in essential form.

The advantages of essential use cases are widely recognized, and a consensus is building that this style is best for both user interface design and requirements modeling in general [Constantine and Lockwood, 2001].

The third model, the content model [Constantine and Lockwood, 1999], is sometimes referred to as an abstract prototype [Constantine, 1998] because it represents in the
abstract— independent of their actual appearance or behavior in the user interface as realized—the contents of a user interface and how these are organized into interaction contexts, that is, the contexts within which users interact with the system. Abstract prototypes can take many forms, ranging from a simple inventory of the visible or accessible contents of a site or application to a highly structured abstract model based on a canonical set of abstract components [Constantine, Windl, Noble, and Lockwood, 2000]. In whatever form, abstract prototypes serve as a bridge between the task model and a realistic or representational prototype, such as conventional paper prototypes or design sketches. By adhering to a user perspective and leaving open the various implementation possibilities, abstraction encourages innovative visual and interaction designs [Constantine, 1998; Corlett, 2000].

Process overview

Figure 3 shows a logical overview of the usage-centered design process. System actors—other software and hardware systems with which the system must interact—are separated from human actors playing roles as users. The three core models are developed in coordination with and interlinked with other models, including a domain model in the form of a glossary or other more elaborate data model, a business rules model that embodies the underlying logic and constraints of the application [Gottesdiener, 1999], and an operational model that captures salient aspects of the working environment or operational context (see [Constantine and Lockwood, 1999]).

Conceptually, a straightforward and direct derivation links the final design back to task cases supporting user roles. Each page, form, or other interaction context corresponds to an abstract prototype supporting a cluster of interrelated task cases. The actual buttons, links, tables, displays, and other features are derived directly from abstract components which realize specific steps within supported task cases. These task cases, in turn, support the performance of roles users can take in relation to the site or application.
Usage-Centered Web Design

Over successive Web projects, a “lightweight” form of the process has evolved that takes certain modeling and design shortcuts [Constantine, 2000a]. Agile usage-centered design [Constantine, 2001a] employs rapid-fire modeling techniques using ordinary index cards for brainstorming, sorting, and clustering [Constantine, 1993; Jeffries, 2000], such as have been popularized by Extreme Programming [Beck, 1999; Jeffries et al., 2001]. When practical, the time to gather, model, and validate user requirements is further reduced by developing models collaboratively with end users, application domain experts, and clients.

In outline, agile usage-centered design proceeds roughly like this:

**Preliminary steps:**
1. Essential purpose and preconception: clarify business and user purposes then fantasize and set aside preconceptions of features, facilities, content and capabilities.
2. Exploratory modeling: identify questions, ambiguities, and areas of risk and uncertainty.

**First iteration:**
3. Role modeling: inventory and prioritize all user roles and select initially targeted subset.
4. Task modeling: inventory and prioritize all tasks and select initially targeted subset.
5. Task clustering: group all tasks by affinity and draft overall navigation architecture.
6. Design: draft visual and interaction scheme; review and revise aesthetic design.
7. Abstract prototyping: develop content model for interaction contexts supporting selected subset of tasks.
8. Design: develop detailed user interface design for selected interaction contexts.
9. Construction: program designed portions of user interface.

**Successive iterations (similarly):**
3a. Review role model and revise as needed, select next subset.
4a. Review task model and revise as needed, select next subset.
5a. Review navigation architecture and revise as needed.
6a. Review visual and interaction scheme and revise as needed.
7a. Develop content model for interaction contexts supporting next selected task subset.
8a. Design user interface for selected interaction contexts.
9a. Construct newly designed portions.

**Essential purpose and preconception**
Web projects are often characterized by vague objectives and unrealistic ambitions. Early clarification of objectives and exploration of fantasies can pay off enormously
We prefer to kick off the process by involving managers, users, developers, and other stakeholders in framing the purposes of the application from the business perspective and from the point of view of external users. Business and external purposes can be brainstormed onto cards, which can then be sorted to establish priorities. In a similar vein, we encourage participants to share their fantasies about the features, functions, facilities, content, and capabilities of the system, but we clearly label these ideas as *preconceptions* and set them aside to be treated as negotiable fantasies rather than requirements.

**Exploratory modeling**

The design is jumpstarted by an exploratory modeling process to identify questions and areas of uncertainty or ambiguity regarding user requirements. This is done by sketching out rough versions of precisely those models—roles and tasks—that will be developed in more refined form later. In this way, needed but missing information is quickly identified.

**User interface architecture**

Many agile processes [Fowler, 2000] abjure up-front design in favor of a more evolutionary approach that first constructs a limited system and then elaborates it through successive iterations. Unfortunately, when code starts small and gradually grows with expanding requirements, the original architecture, the underlying organization of the code, often proves insufficient or inappropriate to support emerging needs. Refactoring [Fowler, 1999], in which existing code is reorganized and rewritten to fit evolving and emerging requirements, is a key to successful evolution of complex software systems through iterative expansion.

User interfaces are a different story. Late refinement of the basic structure of the user interface is not acceptable because it changes the system for users who have already learned or mastered an earlier version. Even small adjustments in the placement or form of features can be problematic for users. Whereas refactoring of internal software components need not necessarily affect users, redesigning user interface architecture is unavoidably disruptive. For highly usable user interfaces, the overall organization, the navigation, and the look-and-feel, must all be designed to fit the full panoply of tasks to be covered.

It is not necessary to design every aspect of the user interface in advance, however. To avoid radical refactoring or inconsistency across design increments, a well-conceived overall map of the site or application needs to be worked out in advance [Cloyd, 2001]. To insure that the basic visual style and behavior are effective across the entire application, a comprehensive design scheme needs to be devised.

Navigation architecture specifies how the overall user interface is organized into interaction contexts, collections, and groups, how these are presented to users, and how users navigate among these. At a minimum, the navigation architecture identifies all the interaction contexts and how these are interconnected, but it is advantageous to have more. For example, the navigation architecture might specify how parts of the user interface will be organized into tabbed notebooks accessed through a set of command buttons on a central dispatch dialog, with sections within notebook pages reached through shortcuts arrayed in a menu-like bar across the top of each notebook page.
Another crucial facet of overall user interface design is the visual and interaction scheme, a sort of abstract style guide. It briefly describes the basic recurrent visual elements that will be used throughout the design as well as the common layouts, visual arrangements, or templates that apply to various interaction contexts [Pokorny, 2001]. The scheme might, for example, specify a basic layout grid with a modified tree-view in the left-most column for primary navigation and a set of view controls across the top for secondary navigation. At a more detailed level it might specify that a distinct color should identify all editable fields, which can be edited in place by double clicking. It might further spell out how certain collections of controls will be placed on slide-out tool panels that automatically open on mouseover and close after use.

A sound navigation architecture and visual and interaction scheme clearly must be based on a complete task model that includes all identified tasks, not just those to be tackled on the first or early iterations.

**Card-based modeling**

The initial user role model is a simple inventory of roles users can play in relation to the site or application. One of the quickest ways to develop such an inventory is to brainstorm it directly onto index cards. Once the initial inventory is agreed upon, roles are succinctly described directly on their index cards in terms of those characteristics most relevant to the design problem: the context of the role, characteristic patterns of interaction within the role, and special criteria for supporting the role. The completed cards are then sorted to rank them in order of overall importance for the design.

After reviewing the roles in order of importance, we brainstorm an inventory of task cases, again directly onto index cards. As we review and refine this inventory—condensing, combining, eliminating, or adding tasks—we standardize the names to convey the basic purpose from a user’s perspective, such as “finding a replacement part for a specific product” or “adding an assessment to a lesson plan.” Once the initial set of task cases is established, the cards are sorted to rank them for expected frequency and for overall importance. On the basis of these two rankings, we deal them into three piles: required (do first), desired (do if time), and deferred (do next time).

At this point, we start to fill in the details, but not for all task cases. We skip task cases for which the interaction looks to be obvious or routine. For those task cases that are critical, complex, unclear, or interesting, we write out the interaction narrative directly on the index card. Working on small cards favors good modeling discipline. If the process narrative doesn't fit on one card, then either the narrative is not sufficiently abstract and simplified or the card really covers multiple task cases that need to be identified.

Using all the knowledge built from creating, refining, and interacting with the task cases, we then group the cards into affinity clusters based on how strongly they seem to be related, in particular, how likely tasks are to be performed together by users in role. Each of these clusters represents a set of capabilities that need to be available together in the application—within the same page, form, or browser window or at least within a set of closely connected interaction contexts.

Each task cluster becomes the guide for designing a part of the user interface. We prefer to construct abstract prototypes [Constantine, 1998] but under pressure may move directly into sketching realistic paper prototypes. Paper prototypes, augmented
by a description of how the various elements behave, is then evaluated to identify usability problems and areas for improvement. For speed and convenience, we prefer collaborative usability inspections [Constantine and Lockwood, 1999] involving users and clients as well as designers and developers, but heuristic inspections and other techniques can also be effective [Nielsen and Mack, 1994].

**In Practice**

The approach we took to the design of a browser-resident classroom information management system illustrates the ad hoc accommodations often necessitated for an agile process applied to Web development in Web-time. This application was a sophisticated performance-support system to empower K-12 classroom teachers by facilitating and simplifying key administrative, planning, and teaching tasks in an environment fully integrated with existing administrative systems and with software and Web-based learning resources. The application was to be deployed through a Web-based technology with the client-side user interface realized in HTML, XML, and Java JFC/Swing.

Risk factors were numerous, including a new and unproven development team in a new company, an outside consulting team for user interface design, the need for technical breakthroughs in several areas, vague and ambiguous requirements exacerbated by frequent changes in scope, and management obsessed with early delivery, frequent demonstrations, and unpredictable changes of direction.

The design team worked in collaboration with a team of PhD-level educators with extensive classroom experience. This education team served as both domain experts and representative end users while also functioning in what amounted to an XP-style customer role [Beck, 2000].

The initial user role model and inventory of task cases were developed collaboratively with the education team. In total, 14 user roles were identified and prioritized. The initial inventory of 142 task cases included 35 task cases related to lesson planning, which emerged as a critical capability for the system. Process narratives—the defining body of use cases—were written out only for a handful of “interesting” task cases [Constantine, 2000a]. A draft navigation map with just over 40 interaction contexts was constructed based on clustering of task cases (see Figure 4).

Omissions, irregularities, and ambiguities in the requirements and problems with the design were resolved on-the-fly by frequent meetings and continual consultation with the education team in a pattern we now refer to as JITR, for Just-In-Time Requirements. One of the two designers concentrated on requirements and scope issues, while the other began working on the initial design. Drafts of a navigation architecture, navigation map, and visual and interaction scheme were devised by the design team, then validated with the education team and revised.

Although not entirely by plan, the design proceeded in cycles of two to two-and-a-half weeks. Once full-time design work started, an initial navigation architecture and visual and interaction scheme were completed within the first cycle. The first version of the complete visual and interaction design was finished in the next cycle, after which it was inspected for usability defects, and a major design revision was undertaken on the second iteration. The third iteration, during which the design continued to be extended and refined, spread over the remainder of the 17-week usage-centered design portion
of the project. Programming followed an agile, lightweight process that proceeded in parallel with visual and interaction design.

An acceptable design needed to be immediately usable by ordinary teachers with little or no special training. It had to be flexible enough to accommodate a wide range of working styles and efficient to use, since teachers may typically have less than 15 minutes a day to plan an entire day’s lessons, for example. Technical goals that were identified based on the usability criteria included:

- Minimize window management overhead.
- Minimize modal behavior.
- Allow simple and direct switching among contexts.
- Maximize available screen real estate for document and data display.

A discussion of the complete design is beyond the scope of this article—a series of design studies based on this work have been published on the Web [Constantine and Lockwood, 2001b; 2001c; 2001d]—but the results of the first design iteration clearly illustrate the importance of an early upfront consideration of the overall navigation architecture and the visual and interaction scheme based on a full understanding of user roles and tasks, something not realizable within either traditional software engineering practices or the newer lightweight methods.

The main portion of the navigation map, as finalized in the second iteration, is shown in Figure 4. The mockup in Figure 5 is adapted directly from actual design documents produced in the first cycle of the project and illustrates how the overall navigation and visual and interaction design scheme were documented. Together, these design documents specified much of the overall architecture of the user interface.

An understanding of the demands of the various roles played by classroom teachers and a thorough exploration of their tasks made clear the importance of a fast, flexible, and easy-to-learn scheme for navigating among the various working contexts of the application. Once the structure of the various tasks was understood, the various groups and interaction contexts could be organized for single-click switching among tasks in an activity of interest. Using simple visual metaphors based on thumb tabs, hanging folders, and file folders, a three-level visible hierarchy was devised that proved far easier to learn and use than a conventional Windows-style “tree view.”

Screen real estate, always a scarce resource, is particularly valuable in an application running within a browser instance on low-to-moderate-resolution displays. To address this, the design scheme included components that would expand and contract or reveal and hide themselves automatically as needed. For example, the top-level navigation panel (“thumb tabs”) contracts to provide more working space once a starting point is selected. Another example seen in Figure 4 is the dynamic workspace, which anticipated rather than mimicked the Apple OS-X “dock.” The workspace provides a compact holding bin that allows the user to easily gather various documents or fragments from a variety of personal and public resources for assembly into a lesson plan or assignment to students.
Figure 4 – Main portion of the navigation map for the classroom application.

Figure 4 – Navigation architecture and design scheme.
The success of any product design is ultimately measured by its acceptance. Classroom teachers in a variety of settings were able to make immediate use of the facility with only minimal instruction. In fact, the designers were able to achieve the stringent objective of enabling immediate productive use of the system by the average teacher given only a “one-page tutorial.”

**Lessons Learned**

We learned a lot from this project [Constantine, 2000b]. Bad management ultimately took the company building the system into bankruptcy before the entire ambitious suite of facilities could be completed. However, despite frequent and capricious changes from management and repeated diversion of programming resources into creation of fruitless demonstration systems, the lesson-planning component as well as some other core facilities were developed and delivered.

This and other experiences in agile usage-centered design have made clear to us that improved usability does not come without cost or risk, even when the methods are streamlined and the schedule is compressed. Special training and skills are necessary, and these may not be available to every development group, especially the lean teams typical in agile development. Moreover, the best design is easily corrupted by casual or even well-intentioned alterations and elaborations during coding. Close coordination between user interface designers and programmers is essential, and programmers must be fully committed to the process and convinced of the benefits even if they do not understand every design decision.

A flexible and condensed usage-centered design process is a good starting point for collaboration, but for agile practices such as Extreme Programming, which abjures anything resembling BDUF (big design up front), some of the rules and philosophy can get in the way. Just which rules need to be bent or rewritten and what philosophy may need to be compromised is the subject of much current debate and experimentation. The outline presented here should be regarded more as a draft than as a definitive model.

**References**


