Software Engineering and the WWW: The Cobbler's Barefoot Children, Revisited

Roy T. Fielding, E. James Whitehead Jr., Kenneth M. Anderson,
Gregory A. Bolcer, Peyman Oreizy, Richard N. Taylor

Department of Information and Computer Science
University of California, Irvine, CA 92697

Technical Report 96-53

November 1, 1996

Abstract

While the World-Wide-Web is demonstrably useful for a wide variety of tasks, in its current form it is not capable of supporting wide-area software development activities. This report describes the following areas for improvement and indicates key directions and approaches for their achievement:

- Support for first-class links, thereby enabling end-user annotation and evolutionary development of relationships within the environment;
- Support for a mechanism by which clients can receive asynchronous notifications of resource changes, which is necessary to support the complex interactions found in software engineering;
- Support for equal access to hypermedia services for client-side viewer applications, thereby facilitating the integration of non-HTML data formats found in software engineering;
- Support for distributed authoring and version control, which is necessary for developers to work within a shared information space;
- Support for flexible control and coordination mechanisms, enabling cooperative software development processes and workflow over the Web.

Software Engineering and the WWW: The Cobbler's Barefoot Children, Revisited

Roy T. Fielding, E. James Whitehead, Jr., Kenneth M. Anderson,
Gregory A. Bolcer, Peyman Oreizy, Richard N. Taylor

Department of Information & Computer Science
University of California, Irvine, CA 92697-3425
{fielding, ejw, kanderso, gbolcer, peymano, taylor}@ics.uci.edu

Introduction

Software engineers have been compared to the cobbler's barefoot children: they make tools and applications that enable users in many domains to perform their work more effectively and efficiently, yet frequently they do not use tools themselves. Apart from use of editors, compilers, and debuggers, too often the analogy is true. This state of affairs is not due to any antipathy to technology. Rather, it simply reflects that relatively few tools genuinely useful in supporting software engineering have been developed. We believe, however, that the Web offers the potential for becoming one of these genuinely useful tools, and thus "providing some shoes for barefoot engineers."

The World Wide Web (WWW) has clearly been successful in helping accomplish tasks in many domains. There are also examples of where the Web has been of help in software engineering projects, and even of enabling some new types of software engineering to occur. The *Apache* web server is a case in point, and serves to motivate our discussion below. The *Apache Group* (http://www.apache.org/) was formed by a group of users to ensure the continued development of a freely-available server implementation of the Hypertext Transfer Protocol (HTTP) [5]. Starting with public-domain source code and a core group of nine volunteers, the *Apache* server was iteratively designed and developed as a collaborative project. The Internet alone was used for project communication and coordination, as a shared information space, and for distribution of the final products. In less than a year, *Apache* held the largest share of the WWW server market and continues to compete successfully against multi-billion dollar corporations. The *Apache Group* was able to collaborate effectively across the Internet because it consists of webmasters: experts in network administration, installation of new software, and communication via FTP, HTTP, and e-mail. They were thus able to do much of the tool integration and coordination work that would be beyond the abilities of most developers, let alone users. Our goal is to make the technology for remote collaboration and coordination a part of the WWW infrastructure, such that the WWW itself becomes an environment for global software engineering and collaborative projects in general.

Descriptions of "the World Wide Web" exhibit many definitions of that term. Some would define the WWW as the world-wide base of linked information. Others define the WWW as the tripartite protocols that form the basis for its use: URI, HTTP, and HTML. Still others equate the WWW with the state of existing browser and server technologies. The first definition focuses on the value provided by the infrastructure; the second focuses on the core technology required to provide that linked information space; the third focuses on the tools and technologies which reside

above the primary infrastructure. While the bulk of this paper focuses on the second and third definitions, for it is with respect to those technologies and standards that changes must be made, it is by considering the potential represented by the first definition—the world-wide information space—that one can see what impacts the Web could have on software engineering and the practice of software development. For example, the potential is present for engineering teams to be dynamically assembled from physically distant sites, for software evolution processes to be more efficient by maintaining the connection between fielded systems and their development environment, and for a world-wide marketplace of software component technology to emerge.

In the sections below we consider a series of technical issues involving change to the WWW infrastructure. We also briefly consider some needs which are consequential to distributed software engineering, namely the need for coordination and communication between engineers across the Web. The changes we recommend and the problems that the changes solve or the opportunities that they enable are summarized in the table below.

Goal	Enabling technology currently missing from the WWW
Linking all SE artifacts and processes (people and tasks)	Links as first-class objects and a client architecture for hypermedia communication between viewers of a multitude of data formats.
Flexible interaction model and hypermedia services	Component-based client architecture with hypermedia workspace manager and data-specific handlers; Notification services
Distributed annotation	Remote linking and links as first-class objects
Visibility of artifacts over time	Versioning of resources
Distributed authoring	Remote locking, linking, access control, and versioning of resources
Distributed coordination and change management	All of the above

The remaining sections of this paper discuss our recommendations for changing the Web to better support the requirements of global software engineering. We first focus on changes to the Web's underlying protocols and an alternative to the traditional WWW client architecture. Next, we consider the higher-level features required to support distributed authoring. Finally, we examine an enhanced model of supporting collaboration over the Web through the provision of tools which take advantage of the changes recommended in the previous sections, and conclude with a discussion of how these changes might affect software engineering practice.

First-Class Links

The hypermedia data model of the WWW provides only a limited notion of link. Links are unidirectional and are embedded directly within HTML documents with the use of the anchor tag. Problems associated with the *embedded links* model include difficult link maintenance and limited hypermedia functionality. An example of the former is the ubiquitous *dangling link*. Since an anchor tag directly specifies its destination, if the target page is moved or deleted without updating the anchor then the user is presented an error message when traversing the link. In response to this problem, site management tools like MOMspider [4] have been created to aid webmasters with main-

taining anchor references and catching HTML errors. These tools are hindered in their support for inter-site management by the lack of standard mechanisms for notifying remote sites about changes to a local site.

The embedded links model also places the burden of providing advanced hypermedia services (such as *guided tours* and *overviews*) on the user. *Annotation* is not directly supported and requires sophisticated programming skills, out of the reach of most end-users, to implement. As mentioned above, only *point-to-point links* are supported making it difficult to group sets of related information.

Making links first-class objects enables solutions to these problems. This approach has been used successfully by *open hypermedia systems* (OHSs) [7] which separate links from hypermedia content and provide an interface for link manipulation. The separation provided by this approach allows the links and anchors of a web to be more easily manipulated and analyzed. Changes to these structures can be made independently of changes to hypermedia content, since they are stored separately, and the external representation allows related structures to be updated automatically (avoiding the dangling link problem). This model of the hypertext, and its associated interface, makes it easier for tools to automatically generate overviews of hypermedia content. In addition, the separation enables the free creation of anchors and links on information displayed by a data-specific viewer. These features give the end-user the freedom to experiment with different link structures over a set of information. Guided tours are thus easier to author and maintain. First-class links are typically modeled as sets, enabling links with more than a single destination (*n*-ary links). Furthermore, typing can be applied to links allowing a variety of relationships to be defined with distinct run-time semantics. Annotation is easily supported by this approach, since minimally all that is required is an integrated light-weight text editor and a link type which defines links to this editor's content as annotations.

Our recommendation for the Web is to augment the existing link model with first-class links. One approach to this goal is to incorporate link server capabilities into WWW servers and protocols.[1] While this approach may require a significant amount of effort, an interim approach to ease the transition is available, namely, the integration of OHSs with the WWW [3, 6]. WWW servers can be modified to filter documents through an OHS before serving them to WWW clients. This allows the OHS to *render* first-class links into the document as embedded links without having to alter the source HTML document. In addition, WWW clients can be modified to communicate directly with the integrated OHS allowing direct access to first-class links. These integrated clients can provide the interface to support *remote linking*, i.e., creating anchors and links on content stored at remote sites.

Notification

The WWW's primary information transfer protocol, HTTP, is based on a strict client-server model. A typical HTTP server waits for client requests, resolves and locates the requested resource, applies the requested method to that resource, and sends the response back to the client. Although this model of communication scales well for simple retrieval tasks, it is not sufficient to support the complex interactions found in software engineering (or any collaborative work process). It is often the case that a change in one resource will necessitate other changes in order to main-

1. HTTP proposals for the ability to transmit non-embedded links via the Link header field have existed for many years, but they have not been widely implemented or standardized due to incomplete specification.

tain the dependencies between resources. In a strict client-server model, the client is forced to poll the server for changes to a resource, which is extremely inefficient when the resource-space is large or when changes are infrequent. What is needed is a means for clients to register interest in a resource and for servers to supply a notification when the resource changes.

Notification is not an entirely new concept for the Web; third-party services exist which monitor a given resource (usually by periodic polling) and send an e-mail message as notification when the resource changes. However, registering for such services is a manual process, as is receipt and processing of the e-mail response, and they only improve efficiency in terms of the number of clients performing the polling. Furthermore, we would like greater flexibility in terms of the client specifying the protocol and message format of the notification, since the message granularity and delivery requirements vary by the type of application and the frequency of change.

Support for notifications could be added within the HTTP protocol as a form of first-class link. A server that supports notification could observe a change to the resource, check the resource for links of type "notify," and post a notification message to the link's destination (a URL) in a format indicated by the link attributes. Notification would thus be possible if support for remote link authoring services was added to the WWW.

Client Architecture

Software engineering involves a multitude of specialized data formats: source code, specifications, test results, project plans, design diagrams, etc. Each of these products introduce important relationships and dependencies within the overall project. Thus, one of our goals is to be able to manipulate all of this data as hypermedia, including the ability to add anchors and link relationships to the objects represented within each data type rather than to just an overlay of one particular rendition of that data. What we need are data-specific handlers (viewers, editors, and other tools) with equal access to hypermedia functionality, allowing for modes of interaction which take advantage of the properties of each particular data type.

The WWW client architecture has traditionally been dominated by the monolithic browser, a huge application that acts as window manager, hypermedia viewer, network request controller, and manager of user preferences, bookmarks, and history. It is difficult and inefficient to introduce new functionality to such an architecture, particularly for the multitude of data-specific handlers that are desired for software engineering. The client architecture needs to be replaced by one consisting of a dynamic collection of small, communicating applications, similar to how Apple's *Cyberdog* client consists of a collection of *OpenDoc* components [2]. However, we currently lack the glue—the interface specifications—which can hold these components together to form a consistent software engineering environment.

Implicit links. Implicit links are links that are derived from the nature of an artifact rather than being explicitly defined by an anchor or external link specification. For example, if a program is written in the C programming language and we have an indexed, hypertext language reference manual for C, then there exists an implicit relationship between every C keyword and operator in the program and its corresponding definition in the language reference manual. While it is possible to explicitly instantiate every one of those relationships as an independent link, it is more

efficient to define the abstract relationship

{ keyword } --> http://site/LRM?keyword

and allow the actual link to be calculated only when invoked by the user. Other implicit links commonly found in source code include def-use relationships, begin-end bracketing, and next-statement jumps, each of which can be calculated by a viewer with a knowledge of the source code language comparable to that of a compiler.

Data-specific Handlers. Current Web clients use a number of mechanisms to allow for hypermedia interaction with data formats other than HTML. The most basic is the media-type handler (sometimes referred to as the mimecap interface), which consists of a program to execute when a particular data type is retrieved. Although this is the basis for most solutions, it does not by itself include any hypermedia-aware interface, and thus the handler must invoke an additional interface if it is to do anything more than act as a read-only window. Current forms of this additional hypermedia interface include the inter-client communication protocol (ICCP) and the Netscape plug-in mechanism. Although useful, these interfaces do not support the full range of hypermedia functionality and require the constant presence of the primary browser application.

Another mechanism for data-specific handlers is the use of applets — mini-applications which supply the rendering and manipulation code for a specific data type. Although applets usually provide a hypermedia interface, that interface is secluded from the overall hypermedia workspace, and thus two or more applets are generally prevented from cooperating on a single task.

Coordinated Tool Interaction. Software engineers use a diverse collection of tools to support their activities, including editors, debuggers, version control systems, static and dynamic analyzers, etc. These tools are not always operated in isolation; rather, multiple tools are used in tandem to solve a particular task. What is frequently lacking in tool coordination is the interface mechanism. Current monolithic browsers serve that function internally, but only within the limited scope of their original design. The essential problem is that, in order to provide for more flexible and extensible clients, the components of the client need to be independent, yet the components cannot work effectively as a hypermedia workspace unless there is something to unify their behavior in response to hypermedia events and user actions. In other words, we need some form of hypermedia workspace manager to provide a set of services that components can access, register handlers, and initiate external hypermedia events.

For example, consider the communication patterns of cooperating code editor, design viewer, and run-time debugger tools. When executing, the run-time debugger acts as a traversal engine, where the link being traversed is the implicit one between a completed code statement and the next statement as determined by the program control flow. Each of these traversals can be considered a hypermedia event, and both the code editor and design viewer can register interest in those events, perhaps with differing granularity, in much the same way that a hypertext mapping tool would register interest in the traversals of a normal browser. Likewise, the user may wish to set a breakpoint by selecting a module in the design viewer or a statement in the code viewer. This type of complex interaction is only possible in a hypermedia-based software environment if the component architecture does not artificially constrain the hypermedia interface to those actions which are common to traditional browser applications.

HTTP was not designed to provide fine-grain interaction between arbitrary tools. However, HTTP is capable of enabling tools to locate each other via URLs and to assist the tools in determining a suitable protocol for further interaction. For example, the Chimera OHS [1] employs this strategy to enable clients and servers to locate and connect to each other: an HTTP request is used to obtain connection information and then the connecting tool uses that information to contact the desired component using a native Chimera protocol.

Distributed Authoring and Versioning

Supporting software engineering over the Web requires support for remote editing of project source code and documents. The HTTP/1.1 protocol provides the bare essentials for distributed authoring of web content, with its *PUT* method and entity tags. Unfortunately, these capabilities are inadequate to satisfy the complete set of features required by users when performing distributed authoring. Existing HTML authoring tools which support a remote "publish" capability routinely define custom extensions to HTTP to meet these needs. To date, the vast majority of authoring practice assumes direct access to the underlying storage medium for web content, typically a filesystem. However, when this access is not available, as in the remote authoring case, many weaknesses of the Web become evident. For example, the Web lacks strong support for preventing the "lost update problem," when two people collaborating on the same document overwrite each other's work. Working in a file system, this problem is alleviated by using a simple locked versioning system like RCS. Other deficiencies include the inability to version a resource, get a directory listing, make a new directory, perform a simple copy or rename, set attributes, or create relationships between resources.

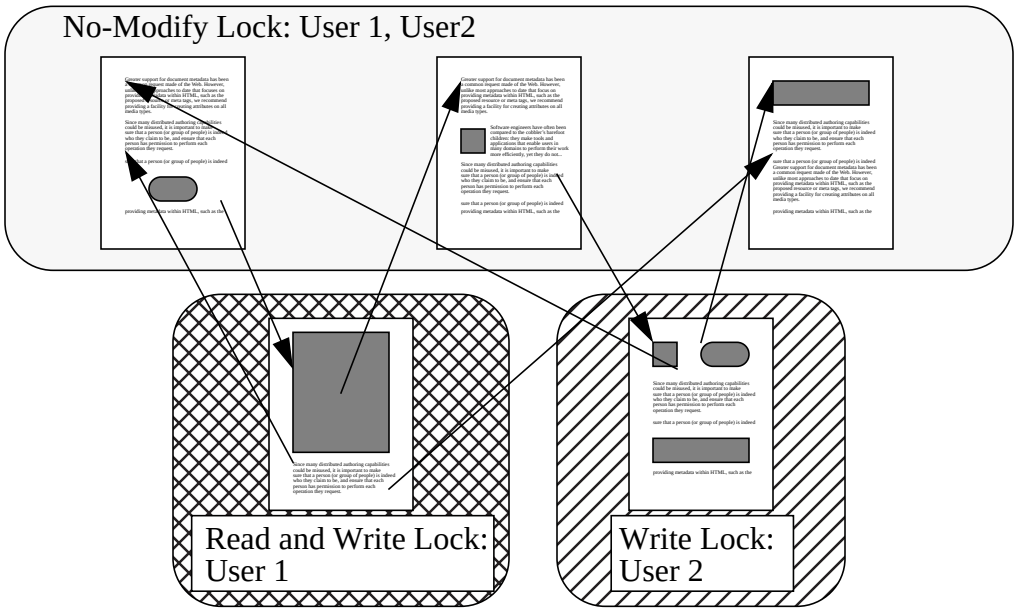

Figure 1: Two users are performing remote edits within a five document web. User 1 has taken out read and write locks on the document he is editing, which ensures that only he can read and write that document. User 2 has taken out a write lock on her document: only she can write to that document, and anyone can read her intermediate drafts. Since the documents being edited have links to the remaining three documents, both users share no-modify locks to ensure the link endpoints are not modified while they edit.

Locking. One solution for the lost update problem is to provide a *write lock* capability. If a document is being edited, the author takes out a write lock, thus preventing others from writing to the same document. Other locks are also necessary. Since many web documents contain links, to ensure these links remain consistent once editing is complete, it is necessary to have a guarantee that the linked-to documents have not changed during editing. This guarantee is provided by a *no-modify lock*, which ensures the locked document will not be modified for the duration of the lock, and which is easier to share between authors than a write lock (see Figure 1). Furthermore, to ensure that intermediate document stages are kept private, a third type of lock, a *read lock*, is necessary. A read lock prevents anyone other than the lock holder from reading the document. Each of these locks must be capable of being held by multiple people simultaneously, to provide support for workgroups. It is often necessary to guarantee that a lock or unlock operation occurs simultaneously across multiple documents, and hence it must be possible to take out multiple locks atomically across multiple documents, requiring lightweight locking transaction support. It is necessary to query a document for its current locks, so authors may discover who is working on which documents. Furthermore, in many situations it is desirable for multiple locks to be taken out on a single document, so long as the combination doesn't contain contradictory access control conditions. One example is an author who wishes to simultaneously have a write lock and a read lock on the same document, so their editing is safe from overwriting and cannot be viewed until complete. The combination of the three lock types, read, write, and no-modify, together form a simple access control scheme which prevent the lost update problem, and allow maintenance of link consistency while authoring.

Versioning. At present, the Web only supports one version of a document, the current version. However, the ability to save and retrieve past versions of a document is critical for supporting software development. We recommend that full-featured versioning capability be added to the Web. The ability to store and access previous document versions, retrieve the history of the document, annotate document revisions with comments about the changes, and retrieve information about the differences between two document versions are all useful functionality, whether they are supporting software development or HTML document editing. Furthermore, the ability to access previous versions of both a single document and collections of documents will help alleviate the current state of "web amnesia," in which old document versions are recklessly discarded in an effort to remain fresh.

Containers. The Web currently provides poor support for container objects, which may be used to group related resources. However, there is a distinct need for the ability to create, modify, and list the contents of container objects. A container can easily satisfy the existing need for access to the contents of a filesystem directory on those web servers which use a filesystem as their underlying storage medium. An immediate need for directory container capability is in remote authoring applications, which require support for "Save As..." dialog boxes which provide a listing of the current contents of a directory, and the ability to go up or down to another directory in the hierarchy. Somewhat exciting for software development is the notion that container objects on the Web need not directly map to operating system directories, and can represent many relationship structures, such as "uses," or other dependencies. Multiple simultaneous structurings are also possible, for example simultaneous maintenance of a directory and compilation dependency structure. Since it is often desirable to prevent others from knowing even the existence of a

document, and to prevent unauthorized modification of a container, container services must be part of a web server's access control scheme.

Copy and Rename. File systems, configuration management systems, and document management systems all provide capabilities for copying documents and changing their names without affecting their contents. There are compelling reasons for the inclusion of this functionality into the Web as well. Copy capability can be used to duplicate a document before starting a modification sequence which results in a separate document. A copy can also be used to move a document to take advantage of the characteristics of the storage medium underlying part of the namespace, such as copying to a separate disk or database. Rename capability allows for the expansion of naming conventions as a project grows. Names which are acceptable for a small set of documents are often too informal for a larger set: a rename capability provides the bridge between the old and new naming schemes. While these operations could be supported by the Web as it now exists by loading and re-saving the contents of a document, this is extremely inefficient, especially for large documents. Furthermore, the load/save approach cannot provide support for recursive copies, a common file system feature.

Attributes. Greater support for document metadata is a common request made of the Web. However, unlike most approaches to date that focus on providing metadata within HTML, such as the `meta` element or the proposed `resource` tag, we recommend providing a facility for creating attributes on all media types. This provides the best support for software development, where embedding HTML markup into source code is typically not feasible. Providing simple attribute-value pairs composed of opaque strings would allow many valuable aspects of a document to be recorded, such as author, title, subject, organization, keywords, etc. These attributes have many uses, such as supporting searches on attribute contents, and the creation of catalog entries as a placeholder for a document which is not available in electronic form, or which may be available later.

Authentication and Access Control. Since many distributed authoring capabilities could be misused, it is important to make sure that a person (or group of people) is indeed who they claim to be, and ensure that each person has permission to perform each operation they request. Key infrastructure required to perform any of the distributed authoring and versioning capabilities outlined in this paper are authentication and access control. There have been many proposals to date for performing authentication; several others are under consideration. We claim no special insight into the problem of authentication, and simply recommend the development and adoption of a standard, robust authentication scheme by the Web community.

Control and Coordination

While the number of WWW users has increased significantly, collaboration between distributed users is limited because the complex workflow processes, i.e., the control and coordination policies, remain difficult to specify or establish. In order to participate in these processes, stakeholders not only need to exchange data, they need to negotiate the policies governing their collaboration. This requires a task-oriented view rather than a data-oriented view. The series of steps for two or more people, workgroups, or companies to collaborate on a project using the WWW are 1) matching of resources and people at the appropriate level and negotiating their roles, 2) formation of

constraints, responsibilities, deliverables, and plans, 3) execution of the process including scheduling, handoff, and sharing of data, and 4) establishing completion criteria. Augmenting the data-oriented view of the WWW with this task-oriented view can be accomplished by building atop the WWW infrastructure and tools, but only if they are changed and enhanced in accordance with the enabling recommendations made in the preceding sections.

In the paragraphs below we describe the steps involved in collaboration on a project, and show the role that a task-oriented perspective brings. Contrasts between the two perspectives are summarized in the table below.

Collaboration Steps

	Data Oriented	Task Oriented
Matching	Content based search tools; Evaluate usability and closeness of data and formats; Implicit access control, readability.	Behavior based search tools; Evaluate usefulness based on tools that can produce, consume, manipulate, or translate format; Explicit access control, changeability, permissions.
Formation	Agree on data formats; Rely on standard content handlers; Data owned by residing server.	Agree on use of compatible tools; Responsibilities and actions are defined; ownership of data is defined independent of location.
Execution	Updates or completion by polling; Email is used for announcements and notification.	Broadcast, synchronization and CSCW; Scheduling, controlled info sharing.
Completion	Each HTTP request is independent.	Each HTTP request is part of an overall task, where the task can be applied or revoked as a transaction.

Matching. "Who do I need to talk to in order to get this changed?" or "Am I talking to the right person?" are the first steps in matching the participants in a collaboration process. In most cases, the correct matching of collaborators does not occur with the first contact. Simply identifying the data to be changed isn't sufficient, even if the data is in an agreed-upon format. In a task-oriented view of the WWW, matching is augmented by examination of participant-specific tool capabilities, permissions, roles, and behaviors. Tools and individuals can be matched by their functionality, not just the data-formats they recognize.

Formation. Once the participants have been identified, agreements must be made on interchange formats and interaction protocols. As the WWW currently exists, participants depend upon standard content handlers at the distributed sites, and coordination is determined simply by ownership of the latest version, as determined by the server on which it resides. With a task-oriented view, the ownership of data can be defined independently of its location. Participants may define the expected responsibilities, which actions should be allowed or restricted, and which tools should be employed.

Execution. Once the responsibilities have been divided up, the creation and handoff of artifacts such as documents or source code can commence. In the data-oriented view completion of an activity is usually indicated by the existence of a new link or creation of a new artifact. Other sites obtain status information by polling for the existence or checking the date of the link; they become entangled in a "keep checking back" syndrome. Notification

usually occurs outside of the WWW through announcements to specific participants using e-mail. With notification support built into the web, as discussed above, an event-driven, task-oriented model is possible, in which participants can be notified directly of key project events, as needed and when appropriate. Similar to how some Computer Supported Cooperative Work systems allow controlled exchanges by “granting the floor” to the current speaker, such events could be broadcast, synchronized, and scheduled in order to create policies for controlled information sharing between distributed sites.

Completion. HTTP requests are normally performed as individual actions. Since HTTP is a stateless protocol, there is no current support for considering a group of related actions as a single task. However, many software engineering activities require changes to multiple resources to occur atomically (i.e. as transactions), at least from the point of view of users of those resources. None of the changes should be applied if any fail. In order to provide support for transactions, it may be necessary to establish a completion criterion or agreement between client and server such that a group of HTTP requests can be applied in isolation and only committed when both sides agree.

Conclusion

Software engineering is fundamentally about the principles, methods, and processes of the production of complex information products. The Web could enable those complex information products and their associated processes to be globally dispersed, highly interconnected, and dynamic.

- Software products may be better designed and constructed, for globally-dispersed teams of specialist designers, analysts, programmers, and testers could be assembled in cyberspace, chosen to fit the particular needs of a development task. Such teams could be assembled even for short-term collaborations.
- Software products could become of much higher quality, and support for software products could similarly be more responsive and effective. Why? Because software products could in many cases remain linked to their development environment. Such linkages could support optimizations based on observed usage patterns, in-field updates and product support, and on-going quality assessment. Even training in product usage or in system maintenance and evolution could be facilitated with linkages to process support in the development environment.
- Software reuse may be greatly increased, as a convenient, world-wide marketplace of software components becomes available.

Naturally none of these potential benefits will be realized easily, for there are both technical and social barriers. Nonetheless, these potentials have already been partially realized in projects like *Apache*. While the WWW has already proven to be of enormous use in many application domains and situations, we believe that making the changes recommended above will yield an infrastructure rich enough to support complex software engineering environments and tasks. Software engineers do not need to be the barefoot children of the computing world any longer. It should also be apparent that the recommended changes will be of value to more than just software engineering, since there are many activities which would benefit from support for collaborative authoring and project coordination, particularly when they can be applied on a global scale.

References

1. Anderson, K. M., Taylor, R. N., and Whitehead, Jr., E. J. (1994). Chimera: Hypertext for heterogeneous software environments. In *Proceedings of the ACM Conference on Hypertext*, pages 94–107, Edinburgh, Scotland. See also <http://www.ics.uci.edu/pub/chimera/papers/ECHT/>.
2. Bortman, H. Walking the Cyberdog. *MacUser* 12,7 (July 1996), 69--74.
3. Carr, L., Hill, G., De Roure, D., Hall, W., and Davis, H. (1996). Open information services. *Computer Networks and ISDN Systems*, 28, pages 1027–1036.
4. Fielding, R.T., Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's Web. *Computer Networks and ISDN Systems* 27,2 (Nov. 1994), 193 --204.
5. Fielding, R.T., Gettys, J., Mogul, J.C., Nielsen, H.F., and Berners-Lee, T. Hypertext Transfer Protocol -- HTTP/1.1. Internet RFC, UC Irvine, DEC, MIT/LCS, Oct. 1996.
6. Maurer, H. (1996). *Hyper-G now, HyperWave: The Next Generation Web Solution*. Addison-Wesley, Harlow, England.
7. Wiil, U. K. and Demeyer, S. (1996). Proceedings of the 2nd workshop on open hypermedia systems. UCI–ICS Technical Report UCI-96-10, University of California.