Embedding Model-Based Architecting in a Collaborative Environment

Stephen E. Dossick, Daniel Port, and Gail E. Kaiser
Dept. of Computer Science
Columbia University
1214 Amsterdam Ave. MC 0401
New York, NY 10027 USA
sdossick@cs.columbia.edu

1.0 Abstract

CHIME is an immersive, collaborative virtual environment designed to support the productivity of teams of software developers. Our initial experiences with CHIME have pointed out some deficiencies which limit its potential usefulness as a support environment for large software projects. In an attempt to address these deficiencies, we plan to create within CHIME "structured guidance" based on the MBASE software engineering approach. In this paper, we describe CHIME and our initial experiences with it (which led us to consider adding a more structured underpinning), as well as the synergies we hope to exploit by embedding MBASE.

2.0 Introduction

CHIME (the Columbia Hypermedia IMmersion Environment) is a collaborative environment which aims to support team software development productivity. The original goal of CHIME was to support project- and domain-specific Multi-User Domain (MUD) style virtual worlds whose layout (map) was to be generated, according to a supplied theme, from the relationships among the underlying software artifacts. Although basing virtual world structure on apparent connections between artifacts did provide certain benefits, it proved ineffectual; without a strong theoretical and conceptual (i.e. semantic) underpinning, it was difficult to generate meaningful and helpful worlds.

MBASE (Model-Based [system] Architecting and Software Engineering) provides a set of guidelines that describe software engineering techniques for the creation and integration of software development models. The models to be integrated extend beyond product (development) models to include process models such as lifecycle and risk models, property models such as cost & schedule, and most notably success models such as business case analysis and Win-Win. Paramount in the MBASE paradigm is the identification and resolution of "model clashes," whereby conflicts in the underlying assumptions of the models may adversely affect the outcome of the project.

Classification, organization, and collaborative use of project artifacts becomes clear when the model structures (and their respective integrations) provided by MBASE are injected into the gen-

eration of a CHIME virtual world. For example, "project houses" can be created with rooms targeting particular MBASE models, such as domain description, software design, reviews, requirements, code repositories, project tracking and control metrics, etc. Outside of a project house, stakeholders may interact purposefully. A virtual "meeting hall" may be used to negotiate requirements. Testing may take place in a virtual test center populated with support tools for generating and applying test plans. MBASE brings rigorously defined semantic meaning to the virtual spaces possible through CHIME and draws in collaborative interactions outside development staff (e.g. users, customers, consultants), both within and outside of an individual project. In addition, MBASE provides a structure for guiding the development and integrating the results of stakeholder collaboration (efforts) into the relevant project areas throughout the entire project lifecycle.Furthermore, this approach supports “open” development whereby arbitrary stakeholders can come and go at any time while the development continually to evolves. This also supports a form of generalized re-use where architectural elements as defined through MBASE and represented tangibly within CHIME can be applied to multiple projects. Within Columbia Universities software engineering course we have realized a great deal of this form of re-use. Typically a project team will be given the MBASE models for a previous similar project (either as discrete examples or the entire project) and instructed to build upon them through modifications and extensions.

The benefits of this directly address the classic problems of collaborative software engineering: Communication overhead due to effort partitioning and (re)integration, communications friction due to temporal and/or geographical distribution, and lack of structured guidance during the development process. These problems were particularly apparent within recent Software Engineering courses at Columbia University and University of Southern California which manually utilized MBASE (i.e. without automated support) as an integral part of their curricula. Students reported that well over 50% of their effort was spent managing and communicating documentation.

By embedding the MBASE guidelines into CHIME, we provide the strong underpinning needed for the creation of useful immersive virtual worlds from software projects. MBASE gains an electronic “support system” which helps users apply its guidelines without becoming overwhelmed or confused by the classic (as described by Brooks [26]) collaborative project management and communications overhead.

This paper proceeds as follows: first we describe CHIME, followed by a discussion of the shortcomings in the current system. Next, we describe MBASE and then discuss the problems with its application we hope to address through CHIME. A discussion of the synergies between the two projects follows, followed by a section describing related research. Finally, we sketch the future work we intend to do to combine the systems.

3.0 CHIME Description

The Oz process-centered software development environment framework [1] was perfectly poised to exploit the emergence of the World Wide Web in the mid-1990's. The proof-of-concept realization of OzWeb [7], added a new kind of built-in object base class, WebObject, to the native object management system [15]. In addition to directly storing the object content, WebObjects

also contained a URL pointing to that content's "home" at any website on the Internet (or intranet). The local content was treated as a cache, with the remote website queried via the HTTP conditional GET - which retrieves the web entity only if it has changed more recently than the cached copy. Users could access WebObjects either through the native X11 Windows client originally constructed for Oz, or through any web browser configured to use our HTTP proxy [16].

When the browser requested a URL that matched a WebObject, it was retrieved from the OzWeb server along with added-on HTML showing the attributes, relationships, etc. imposed on the entity within OzWeb. But when the browser requested any other URL, not currently known to OzWeb, the proxy forwarded the request to the appropriate external website, and only added on a frame giving the user the option of immediately adding that web entity to the OzWeb objectbase. OzWeb also supported HTTP PUT, for updating backend websites containing in-progress project materials.

Unfortunately, this approach didn't scale very well as we attempted to add on other kinds of Internet and proprietary protocols, besides WebObjects/HTTP. This is not very surprising: the OzWeb code was essentially legacy code that had far outlived its origins in the 1986 Marvel design [24]. Its over 300k source code lines had been added to or modified by about fifty students, included some code written a decade earlier, and was still based on the mid-1980's Unix/C model. OzWeb was ready to retire. We started over again, with a new design and architecture, coding in Java, and targeting the Windows NT platform - to produce Xanth [8]. We also further componentized the old OzWeb facilities, which had been in progress since the later versions of Marvel, with all the new components also written in Java. For instance, the old Pern transaction manager was redesigned and reimplemented from scratch as JPernLite [5].

Xanth neatly partitioned data access modules (DAMs) for accessing arbitrary backend data sources through their native protocols, presentation access modules (PAMs) for appearing to arbitrary front-end user interface and tool clients as their native servers, and service access modules (SAMs) for inserting hyperlinking, annotation, user authorization, workflow, transaction management, etc. services wrapped around PAM and DAM operations. The SAMs were connected to each other and the DAMs and PAMs via a novel event bus, called the Groupspace Controller, which not only propagated notification events but also supported request events that could be vetoed by any service so registered. Veto is needed to realize workflow constraints, transaction all-or-nothing guarantees, etc. The conventional event notification after the fact of a prohibited activity is obviously too late. Many events (e.g., sending email, printing) simply cannot be undone or fully compensated, and those that can incur substantial overhead that is unnecessary if the architecture had allowed for them to be prevented in the first place.

Xanth enabled us to effectively and efficiently, in about 50k lines of Java code, reimplement OzWeb through a fully scalable architecture. We easily incorporated a variety of backend data sources like CVS source code repositories, NNTP newsgroups, Ical group calendar managers, and so on. We also developed a variety of Web-oriented user interfaces for Xanth, moving away from relatively limited HTML to try browser-resident applets and host-installed apps, as well as legacy clients, e.g., University of California at Irvine's Chimera linkbase viewers [17].

But none of these user interfaces were truly satisfactory. Like all the other software development environment researchers and commercial developers we know of, we were using single-user

styles of user interface as clients on an inherently collaborative multi-user system. We realized then that we needed to develop groupviews: a user interface style whose core centers on collaboration. The best examples we could find of such user interfaces were in extremely popular on-line games and socializing forums: 3D virtual worlds and MUDs. These forums are actively used by the general populace, school age children to the elderly, with no formal computer science training and often not even computer literacy training. They pick it up through intuition from the physical world counterpart and informal peer help.

These insights led to our CHIME (Columbia Hypermedia IMmersion Environment) project [6]. One of our most deeply seated tenets is to leverage success in achieving usable, useful and used groupviews. Systems constructed using the CHIME infrastructure present their users with a 3D depiction of hypermedia and/or other information resources. Users visualize, and their avatars operate within, a collaborative virtual environment based on some metaphor selected to aid their intuition in understanding and/or utilizing the information of interest or relevant to the task at hand. Users "see" and interact with each other, when in close [virtual] proximity, as well as with the encompassing information space. Actions meaningful within the metaphor are mapped to operations appropriate for the information domain, such as invoking external tools, running queries or viewing documents.

A proof-of-concept implementation of CHIME has recently been developed. In the preliminary architecture, the base data from one or more sources is first mapped to extensible subtypes of the generic components: containers, connectors, components and behaviors, in a virtual model environment (VEM). This includes specifying relationships (connection and containment) among entities from the same and different sources, which might be imposed by the application rather than inherent in the data. A VEM is then mapped to extensible subtypes of multi-user domain facilities like rooms, doors or hallways between rooms, furnishings, object manipulations, and so on. These are in turn rendered and activated according to the chosen 3D theme world "plugin", which can be dynamically loaded into the generic theme manager at run-time and thence transmitted to the user clients. This is not shown in Figure 1 for simplicity, but the same VEM can be mapped simultaneously to multiple theme managers. This permits significantly different metaphors to be seen by different users of the same virtual environment; we find this useful for debugging, administration and system monitoring, but probably too confusing for members of the same collaborative team. In practice all collaborators share the same theme world.

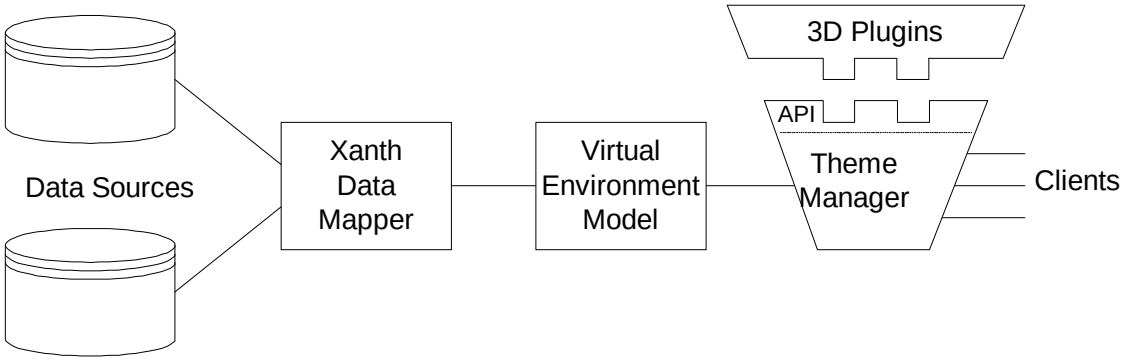


FIGURE 1. CHIME Architecture.

Thus an e-commerce web site peddling computer hardware might look and feel like an on-screen CompUSA; a digital library might be illustrated as, indeed, a library. Application domains without obvious physical counterparts might choose more whimsical themes. For example, a software development environment for an open-source system might map each source code package to a room on the Starship Enterprise, with the "main" subprogram represented by the bridge, amateur programmers proposing a modification could beam aboard, and so forth. Note these are just possibilities: CHIME is a generic architecture, no particular theme is built-in. But environment designers do not necessarily need to program since graphic textures and models can be supplied by third parties, and the specific layout and contents of a world are automatically generated according to an XML-based configuration. The environment designers must, of course, understand their backend repositories sufficiently to write the XML and corresponding processors, unless such meta-information is already supplied by the sources.

4.0 Initial Objectives and Considerations

CHIME's greatest strengths, namely its flexibility and lack of built-in constraints on the semantics and capabilities of the virtual worlds it generates, are also in many ways the greatest obstacle to the effective use of CHIME by software engineering teams. As it currently exists, CHIME models the artifacts of the software project and the relationships among the artifacts, generating the virtual world from the apparent connections between them. This type of information is useful to developers familiar with particular subcomponents of a software system, since it allows them to easily navigate among potentially large amounts of legacy code, documentation, test plans, etc. Unfortunately, this does not help new project team members or those unfamiliar with particular subcomponents of the system. They are likely to quickly get "lost" among all the project artifacts in the unfamiliar areas, experiencing the virtual environment version of the "Lost In Hypertext" problem. ([25] provides an overview of various navigational issues encountered by users of hypertext systems which often apply to virtual environment systems as well). While the goal of the CHIME virtual environment is to put an intuitive, easy to use face on top of project data, it simply cannot do so without a deeper understanding than it currently has.

In one of our initial, proof-of-concept projects, we generated a CHIME virtual environment from the source code and documentation of the Linux 2.0.36 kernel. As this stage of CHIME's development, we were initially concerned with its scalability to larger systems. We chose the Linux kernel in particular because it was quite large (over one million lines of source code spread across over 2000 source files), had a large amount of supporting documentation (we were able to collect over 400 megabytes of design documentation, archives of email among the developers of various subsystems, and rationale on abandoned design changes from various WWW sources), and was easily available to us.

Although CHIME performed admirably in terms of scalability to a large project, we came to realize that it was not as useful as a support tool for developers as we had hoped. Immersed in the relationships between artifacts and navigating among the various rooms in the virtual world, we would experience the "Lost In Hypertext" problem firsthand. We did not experience these kinds of problems when using similar CHIME virtual worlds based on software systems we had written. While this now seems a straightforward conclusion to us, it was not obvious when first developing CHIME.

One way we plan to address this issue is by embedding a strong methodological component into CHIME. Artifacts will then be placed in the virtual world not only according to relationships among one another, but according to their role in the methodology. Our initial choice of methodology to integrate with (MBASE) provides a strong underpinning for the virtual world. We intend the semantics of the methodology to provide a structuring component which is missing from existing CHIME virtual environments.

5.0 MBASE description

The recent President's Information Technology Advisory Committee (PITAC) Report [2] emphasized the fragility of currently-produced software, and identified software as the highest-priority area for increased IT research. The corresponding NSF Software Research Workshop Report [3] emphasized the need for techniques that improve our ability to produce "no-surprise" software.

Some major sources of software surprises are the hidden conflicts among the models the software system stakeholders (users, customers, developers, maintainers, marketers, and others) bring to a software project. Model-Based [System] Architecting and Software Engineering (MBASE) is an approach towards eliminating these conflicts or "model clashes."

An example of a model clash is a situation in which the user's success model creates requirements for an ambitious set of capabilities; the customer's success model or property model requires the capabilities in 9 months; and the developer's waterfall (develop-to-requirements) process model would at best accommodate the development of the user's requirements in 21 months rather than 9 months. If the project goes forward with this model clash (as many have), none of the stakeholders will emerge as winners. Such model clashes have underlain many classic software disaster projects as American Airlines/Intrico's Confirm system and Bank of America's MasterNet system [9].

MBASE is a set of guidelines that describe software engineering techniques for the creation and integration of software development models. The models to be integrated extend beyond Product (development) models such as object-oriented analysis and design models and traditional requirements, to include Process models such as lifecycle and risk models, Property models such as cost and schedule, and most notably Success models such as business-case analysis and WinWin. Par-

amount in the MBASE paradigm is the identification and resolution of model clashes (see "Escaping the Software Tar Pit: Model Clashes and How to Avoid Them" [4]).

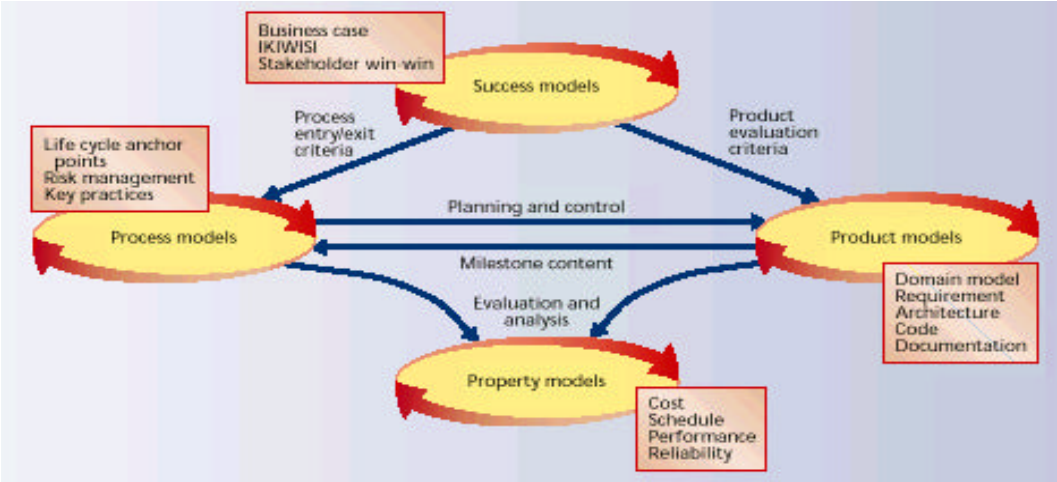


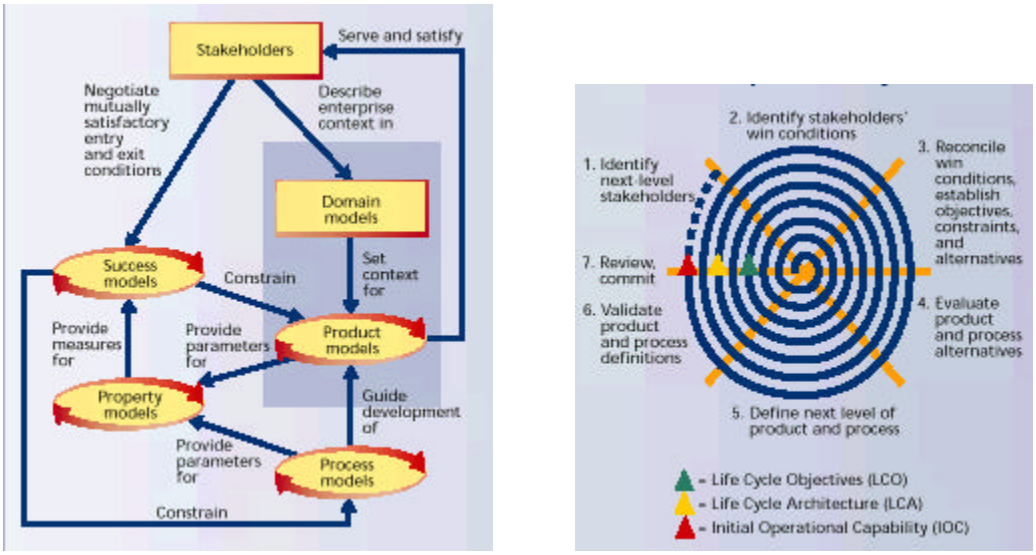
FIGURE 2. MBASE Framework.

Figure 2 [9] summarizes the overall framework used in the MBASE approach to ensure that a project's success, product, process and property models are consistent and well integrated. At the top of Figure 1 are various success models, whose priorities and consistency should be considered first. Thus, if the overriding top-priority success model is to "Demonstrate a competitive agent-based data mining system on the floor of COMDEX in 9 months," this constrains the ambition level of other success models (provably correct code, fully documented as a maintainer win condition). It also determines many aspects of the product model (architected to easily shed lower-priority features if necessary to meet schedule), the process model (design-to-schedule), and various property models (only portable and reliable enough to achieve a successful demonstration).

The achievability of the success model needs to be verified with respect to the other models. In the 9-month demonstration example, a cost-schedule estimation model would relate various product characteristics (sizing of components, reuse, product complexity), process characteristics (staff capabilities and experience, tool support, process maturity), and property characteristics (required reliability, cost constraints) to determine whether the product capabilities achievable in 9 months would be sufficiently competitive for the success models. Thus, as shown at the bottom of Figure 2, a cost and schedule property model would be used for the evaluation and analysis of the consistency of the system's product, process, and success models.

In other cases, the success model would make a process model or a product model the primary driver for model integration. An IKIWISI ("I'll know it when I see it") success model would initially establish a prototyping and evolutionary development process model, with most of the product features and property levels left to be determined by the process. A success model focused on developing a product line of similar products would initially focus on product models (domain models, product line architectures), with process models and property models subsequently explored to perform a business-case analysis of the most appropriate breadth of the product line and the timing for introducing individual products.

Figure 3 shows the process framework within which stakeholders express their initial desired success models, and proceed to adjust these and their associated product, process, and property models to achieve a consistent and feasible set of models to guide the project and its stakeholders. The actual process generally takes several iterations, and requires some common intermediate checkpoints. Figure 4 shows the MBASE extension of the original spiral model [10] to include stakeholder win-win model negotiation and a set of common anchor point milestones [11]: key life-cycle decision points at which a project verifies that it has feasible objectives (LCO); a feasible life-cycle architecture and plan (LCA); and a product ready for operational use (IOC).



FIGURES 3 and 4. MBASE Process and Spiral Model Extensions.

6.0 Synergies between CHIME and MBASE

A number of synergies exist between CHIME and MBASE. We will discuss each of these in turn.

- MBASE provides structure and guidance for development tasks, from initial concept capture, through design to coding and testing with high assurance.

  The structure of MBASE makes it easier for project members to find artifacts. In addition, CHIME environments provide a structure for communications among team members. This allows CHIME to address the classical Brooksian problem of communications overhead on a large team. The combination of MBASE and CHIME allows developers to cluster their communications among interested parties more easily than they could without support tools.

- CHIME can capture discourse among the developers inside the virtual environment

  One problem with applying MBASE manually on a project is that team members must be extremely diligent in the capture of rationale discussed in team meetings, meetings with the project's customer, etc. Typically, this does not happen -- teams attempt to reconstruct later from memory all past discussions, with less than spectacular results. CHIME can capture this discourse between avatars in the virtual environment, and can place the meeting transcriptions at a virtual world location consistent with MBASE guidelines.

- Context

  MBASE provides context for a project. By rigorously documenting each decision and supposition in a project, MBASE makes it easier for teams to maintain a focus on project goals. Similarly, CHIME eased overhead of existing team members to maintain this context and for new team members to come up to speed on project history. The virtual environment makes it easy for users to focus only on relevant project artifacts and not become overwhelmed by the sheer size of a large project. Further, contexts can be re-used to rapidly scope new projects within similar domains.

- Effective Means to Focus/Scope Distributed Collaborations

  Traditional distributed collaboration (via email, fax, pony express, etc.) is hard because the scope of an individual's efforts is not transmitted along with new versions of artifacts. As a result, participants are too narrowly focused and the collaborative process is easily stalled.

  Face-to-face collaboration is often ineffective due to the large quantities of unfocused information generated by unconstrained interactions between multiple stakeholders. This often results in an inability to apply and retain a large percentage of this valuable information.

  MBASE provides guidelines which help stakeholders collectively focus on the relevant areas and issues. Through the constraints of a virtual environment, CHIME ensures that stakeholders maintain the appropriate scope. CHIME provides a physicality for the MBASE component models, e.g. suppose project members are working together in the design room and discussions migrate into testing efforts. They will be unable to insert the testing artifacts into the design room's structure. They are forced to transport to the testing room -- preserving the integrity of the design artifacts are at hand.

  Virtual environments provide a physicality where project stakeholders can "go" to an artifact's location to discuss it rather than manually transmit revised versions back and forth. This gives the collaborators a choice of what development mode to use in working with the artifact; they may choose asynchronous individual efforts, asynchronous collaboration (in which members take turns working on an artifact), or synchronous effort (via an application sharing tool like Microsoft NetMeeting). This reduces communications overhead and friction yet provides enough constraint and structure to focus collaborative efforts on the task at hand.

  In using the semantic structures inherent in MBASE within CHIME, we partition the project artifacts along clear semantic boundaries. Thus it becomes clear to team members not only where in the virtual space to find artifacts but also what dependencies exist between various project components. This helps project teams to effectively partition along meaningful dependency sets to maximize parallel efforts.

7.0 Related Work

A number of other research efforts described in the literature are related in various ways to CHIME. LambdaMOO [18] is prototypical of many MUD systems, and many newer systems are still built around the original LambdaMOO implementation. LambdaMOO, through the use of an object-oriented database and associated programming language, explored many of the ideas in Groupviews. We chose not to build on LambdaMOO, however, because the OODB underlying the system must contain all virtual environment components, and could not easily be extended to a metadata model (like the one used in CHIME's Xanth data mapper).

Many research and commercial Groupware systems might at first glance appear to be good candidates for building CHIME-style environments. Systems like Orbit [19], TeamRooms [20] eRoom [21], and Lotus Notes [22]do have much in common with CHIME Groupspaces, but, like LambdaMoo, these systems store artifacts inside their servers. When they do allow reference to external data, it is often limited to a Web link.

Research into Software Visualization (and the related area of Algorithm Animation) looks at the design and development of techniques to show program code, algorithms, and data structures by using typography, graphics, and animation. The Software Immersion in our conceptual model for CHIME can be seen as a form of Software Visualization, as we are displaying the organization of software artifacts through the design of a virtual environment. [23] contains a good overview of research in this area.

There are a great number of software development methodologies such as Rational's Objectory process [13],[14], and Jacobson's OOSE [12]. However, MBASE has several distinct characteristics that are vital for application to education, research, and practice: in contrast to many development methodologies, MBASE directly addresses the critical issues of model integration and model clashes. Furthermore, MBASE addresses the "full lifecycle" of a software development effort, explicitly incorporating the concept of software architecture and system architecting (as discussed earlier), scales with project size, and provides explicit means of evolving to incorporate new views, models, tools, and other means of process evolution. In particular the MBASE goal oriented, risk driven milestones (LCO, LCA, IOC, etc.) are an ideal fit for many projects. These milestones assure (measure, verify and validate) project progress, unearthing potentially fatal risks early on and greatly reducing the occurrence of the all too common problems of procrastination and confusion as to what needs to be done and when. This allows for proactive intervention from the management staff in order to avoid the often disastrous consequences of such problems.

8.0 Future Work

As mentioned above, MBASE guidelines have been applied manually to a number of projects. We have raw data from 33 projects in the Spring 1999 Software Engineering course at Columbia University. We intend to study these results in detail in order to design metrics for comparison with future classes. In particular, we would like to determine a baseline for efficiency and efficacy of collaborative development using MBASE.

In parallel, we plan to develop a CHIME theme with MBASE guidelines embedded in it. This will allow us to deploy a CHIME-enabled MBASE in an upcoming Software Engineering course. We then plan to study the results of such a class and compare them to those from the manually-applied MBASE projects. We will continually revise and refine the CHIME-MBASE combination based on these results. In particular, we hope to verify the existence and utility of the synergies mentioned in Section 6.0.

9.0 Acknowledgements

The authors wish to thank Adam Stone and Janak Parekh for their input to both projects discussed here. In addition, we thank the student participants and teaching assistants of the Spring 1999 CS 3156/4156 Software Engineering courses at Columbia University.

10.0 References

1. Ben-Shaul and Kaiser, A Paradigm for Decentralized Process Modeling, Kluwer, 1995.
2. President's Information Technology Advisory Committee Report to the President, Information Technology Research: Investing in Our Future, 1999. http://www.ccic.gov/ac/report/
3. V.R. Basili, *et al.*, Final Report On a Software Research Program For the 21st Century, *ACM Software Engineering Notes*, 1999. http://www.cs.umd.edu/projects/SoftEng/tame/nsfw98/.
4. Boehm, Port, "Escaping the Software Tar Pit: Model Clashes and How to Avoid Them ", to appear in ICRE99, June 1999, http://sunset.usc.edu/TechRpts/Papers/usccse98-517/usccse98-517.pdf
5. Jingshuang Yang and Gail E. Kaiser. JPernLite: Extensible Transaction Services for WWW. To appear in *IEEE Transactions on Knowledge and Data Engineering*, Jul/Aug 1999. ftp:// ftp.psl.cs.columbia.edu/pub/psl/CUCS-009-98.pdf.gz.
6. Stephen E. Dossick and Gail E. Kaiser. CHIME: A Metadata-Based Distributed Software Development Environment. To appear in *ICSE-99 2nd Workshop on Software Engineering over the Internet*, May 1999. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-006-99.zip
7. Gail E. Kaiser, Stephen E. Dossick, Wenyu Jiang, Jack Jingshuang Yang and Sonny Xi Ye. WWW-based Collaboration Environments with Distributed Tool Services. *World Wide Web*, Baltzer Science Publishers, 1:3-25, January 1998. ftp://ftp.psl.cs.columbia.edu/pub/psl/CUCS-003-97.ps.gz.
8. Wenyu Jiang, Gail E. Kaiser, Jack Jingshuang Yang and Stephen E. Dossick. WebCity: A WWW-based Hypermedia Environment for Software Development. Poster paper in *7th Workshop on Information Technologies and Systems*, December 1997, pp. 241-245. ftp.psl.cs.columbia.edu/pub/psl/wits97.ps.gz
9. B. Boehm, M. Abi-Antoun, J. Kwan, A. Lynch, and D. Port, "Requirements Engineering, Expectations Management, and the Two Cultures," Proceedings, 1999 International Conference on Requirements Engineering, June 1999.
10. B. Boehm, "Anchoring the Software Process," *IEEE Software*, July 1996, pp. 73-82.
11. B. Boehm, "Rapid Application Development," *IEEE Computer*, March 1999, pp.
12. Kruchten, P., The Rational Unified Process, Addison-Wesley, 1998.
13. Royce, W.E., Software Project Management: A Unified Framework, Addison-Wesley, 1998
14. Jack Jingshuang Yang and Gail E. Kaiser, An Architecture for Integrating OODBs with WWW, International World Wide Web Conference, Computer Networks and ISDN Systems,

The International Journal of Computer and Telecommunications Networking, 28(7-11), 1996. http://www.psl.cs.columbia.edu/papers/CUCS-004-96.html.

15. Stephen E. Dossick and Gail E. Kaiser, WWW Access to Legacy Client/Server Applications, International World Wide Web Conference, Computer Networks and ISDN Systems, The International Journal of Computer and Telecommunications Networking, 28(7-11), 1996. http://www.psl.cs.columbia.edu/papers/CUCS-003-96.html.
16. E. James Whitehead, Jr., An Architectural Model for Application Integration in Open Hypermedia Environments, *ACM Conference on Hypermedia Technology*, 1997.
17. S.I. Feldman, Make - A Program for Maintaining Computer Programs, *Software - Practice & Experience*, 9(4):255-265, April 1979.
18. P. Curtis, MUDs Grow Up: Social Virtual Reality in the Real World, *Conference on Directions and Implications of Advanced Computing*, 1992.
19. T. Mansfield, S. Kaplan, G. Fitzpatrick, T. Phelps, M. Fitzpatrick, Evolving Orbit: a Progress Report on Building Locales, *ACM Conference on Supporting Group Work (GROUP)* 1997.
20. TeamWave Software, Ltd. TeamWave Workplace, http://www.teamwave.com/
21. Instinctive Corp., eRoom. http://www.instinctive.com/html/eroom.html
22. Lotus Development Corp., Lotus Notes. http://www.lotus.com/
23. J. Stasko et al., editor, Software Visualization: Programming as a Multimedia Experience. MIT Press, 1998.
24. Gail E. Kaiser, Peter H. Feiler and Steven S. Popovich. Intelligent Assistance for Software Development and Maintenance. *IEEE Software*, 5(3):40-49, May 1988
25. J. Rosenberg. The Structure of Hypertext Activity. *ACM Conference on Hypermedia Technology*, 1996.
26. F. Brooks, Jr. The Mythical Man-Month. Addison-Wesley, 1995.