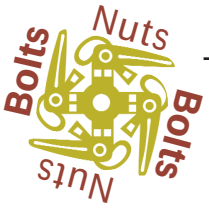


Tool Report

The accelerated pace of software development and the geographically distributed nature of many development teams demand new process models. The author describes the Agile Software Process, a model that tackles these challenges and that is already in use at Fujitsu.

Web-Based Agile Software Development

Mikio Aoyama, Niigata Institute of Technology

The Internet changed software development's top priority from *what* to *when*. Reduced time-to-market has become the competitive edge that leading companies strive for. Thus, reducing the development cycle is now one of software engineering's most important missions. The market demands that we deliver software ever more quickly, but also with richer functionality and higher quality. Further, today's time-sensitive business climate requires that we quickly accommodate requirements changes during development and, after development, be equally adept at delivering the upgrades caused by software's rapid software evolution and the customer's ever-increasing requirements.

On the other hand, many software development organizations have become dispersed, with more than 50 percent of their developers spread across multiple teams at geographically distributed sites. Thus, the friction of distributed—and therefore often delayed—communication actually works to slow software development.

To meet the sometimes conflicting demands of delivering software products faster

0740-7459/98/$10.00 © 1998

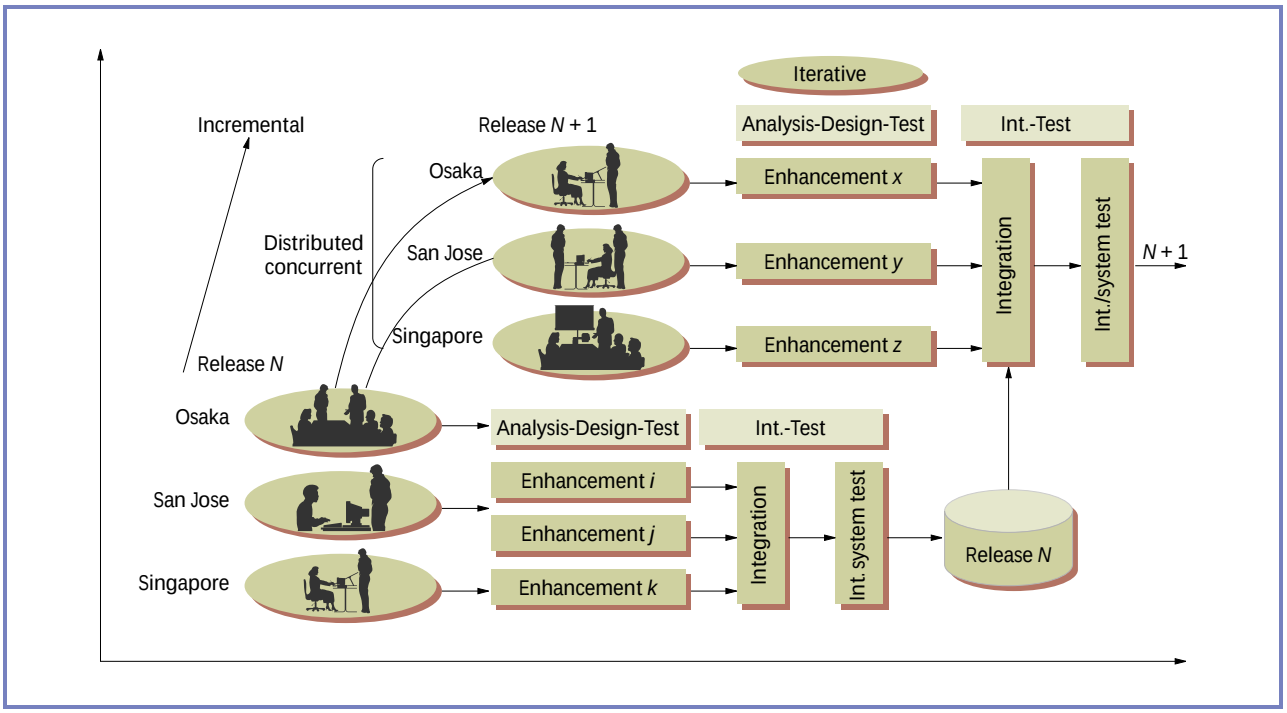


Figure 1. A process model of the ASP system. The software process is segmented so that teams can work on different releases concurrently.

while simultaneously facilitating their widely distributed development, I propose a fundamental redesign of the software development process, which I call the Agile Software Process.[1,2] ASP aims to develop software quickly while maintaining the flexibility needed to respond to changing requirements.

However, because ASP is more complex and vulnerable to disruption than are conventional software process models, it must operate within a specially tailored software engineering environment to function smoothly. Among various management techniques that I and my colleagues at Fujitsu developed, two methods—the time-based process enaction model and just-in-time process management—play a central role in managing ASP.

These methods are embedded into a process-centered software engineering environment called Prime, and a Web-based information-sharing environment, the Wide-Area Information Network. Together with other software engineering tools, Prime and WAIN form the network-centric Agile Software Engineering Environment.

We have used ASEE to develop large-scale communications software systems at Fujitsu for several years. Our experiences with the development and application of ASEE follow.

The Agile Software Process

Agility in software development means not only quick delivery of software products but also quick adaptation to changing requirements. To be agile, the process must be flexible enough to adapt smoothly to changes in requirements and delivery schedule. Conventional software process models, such as the waterfall model, are monolithic and slow, focusing as they do on a single long cycle time. Further, conventional process management is based on the volume of requirements. The greater the requirements' functionality, the longer the project's time-to-delivery and the lower the process's flexibility and productivity. Conventional management principles also lead to various scheduling problems, such as achieving on-time delivery, because it is hard to estimate the exact volume of work involved at the project planning stage.

To address these issues, ASP alters traditional management principles as follows:

- the process architecture shifts from monolithic to modular, and
- the process dynamics shift from volume-based to time-based.

Figure 1 shows the ASP model's modular and time-based nature.[2] The example depicted in the figure assumes that the development organization consists of several small development teams spread across multiple sites. From an architectural viewpoint, ASP consists of two parts:

- the upper-stream process, which covers the analysis to implementation phases, and

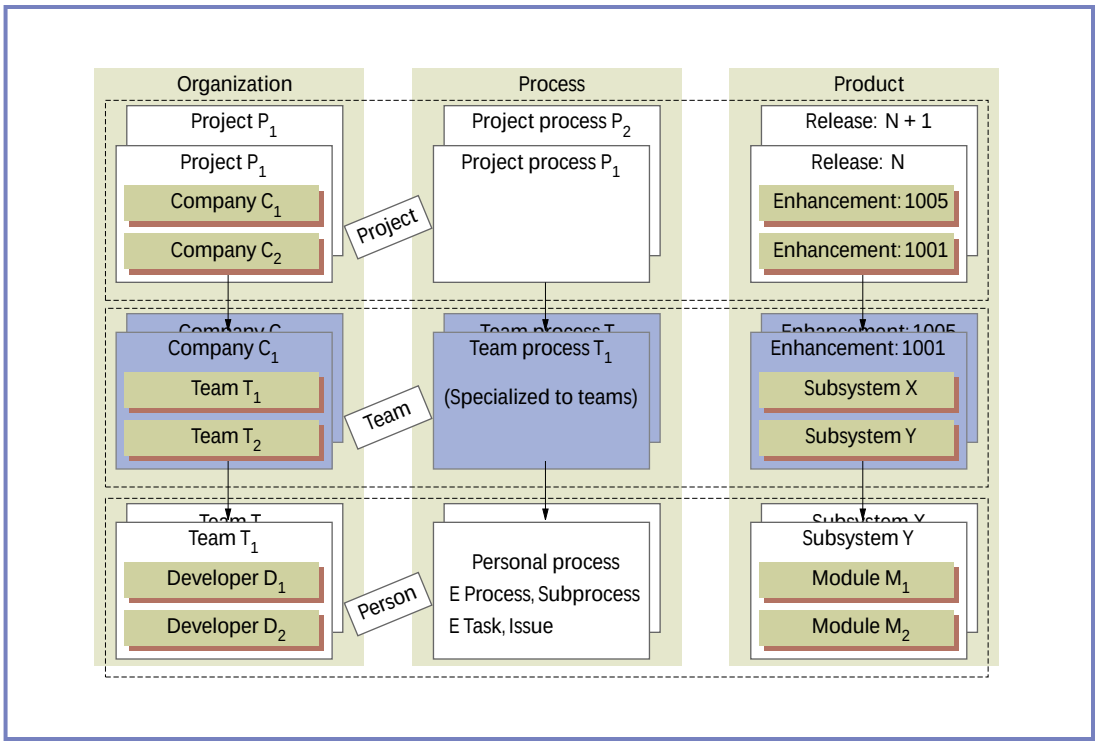


Figure 2. The ASP management hierarchy, distributed horizontally across the organization, and vertically down through the project, team, and personal levels.

♦ the lower-stream process, which covers the integration testing and system testing phases.

Typically, the upper-stream process consists of several concurrent software processes assigned across multiple teams. When each team completes its upper-stream process, it checks in the products to be integrated into a whole system, say Release *N*. Then, the Release *N* integrated software goes through the lower-stream process of integration testing and system testing.

ASP's agility stems from its macroprocess architecture. Although the unit software process is based on the waterfall model, ASP's entire architecture is incremental and iterative. Thus, the unit process has been redesigned to accommodate concurrent development. Each team iterates the two-part, upper- and lower-stream release cycle. ASP is time-based in that requirements are divided or integrated into units of incremental releases, called "enhancements," which have a fixed time for development. The development of an enhancement must be completed within a single release. Thus, enhancement allocation, which assigns enhancement work to a particular team, is a critical task whose success depends on the development teams' expertise and capabilities.

The software process is divided at the middle, so that the lower-stream process for Release *N* overlaps the upper-stream process for the next release, *N*+1. This enables each development team to work seamlessly and concurrently on multiple releases.

Managing in Real Time

Managing ASP is a real-time activity. It requires the timely management of ASP's dynamic behavior to apply its software process over multiple releases. I have identified two key principles of time-managing the ASP:

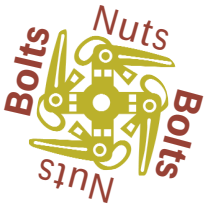
♦ a time-based process enaction mechanism, and

♦ just-in-time process management.

The time-based process enaction mechanism mandates that all of a project's development processes have a fixed development cycle time, such as six months. As Figure 1 shows, each development process overlaps at the middle. Thus, the delivery cycle time would be three months.

Just-in-time process management mandates that the right information be provided to the right people at the right time—a shorter cycle time requires such precise and systematic process management. For example, we must support people at different levels of the organization—the individual developer, the team leader, and the project manager—so that each can track, at the appropriate level of abstraction, the processes and products upon which they are working.

To design ASP's just-in-time management, I analyzed our software process and products extensively, then set up a JIT management reference model. As Figure 2 shows, the model has three levels of management hierarchy—person, team, and



project—that function across the organization, process, and product.

I developed the concept of management levels because each management level requires different information and methods. I also found it necessary to define the relationships among the organization, the process, and the product so that we can coherently manage the process and product together within the organization. Further, management must be able to move easily up and down the levels of abstraction, which required us to establish the relationships between the different management levels. For example, project leaders must be able to browse through progress reports for the project and teams, while team leaders must be able to browse reports for their team and individual developers.

To succeed at JIT management, we emphasized helping the individual developer improve his or her quality of work because, as Watts Humprhey observed when defining his Personal Software Process,[3] software's quality depends on the individual developer's performance.

Agile Software Engineering Environment

To manage the Agile Software Process, I developed an integrated software engineering environment, which I call the Agile Software Engineering Environment. I based ASEE's design on the following concepts.

♦ *Support for just-in-time management of both process and product.* ASEE supports the management model shown in Figure 2. Conventional software engineering environments, such as process-centered software engineering environments, focus on process control, while CASE environments focus on products. However, the process and product are two sides of the same coin. For a project to succeed, we must provide integrated support for both process and product. For example, we provide the design team with both a design process and design documents, and we provide the testing team with both test progress reports and quality information such as the number of bugs.

♦ *A network-centric architecture.* ASEE is network-centric because it is designed for distributed multiteam development. Collaboration over the extranet is critical to distributed development.

♦ *Support for the individual developer.* ASEE supports each individual's work by providing a personal viewpoint so that each project member can understand the role he or she plays on the team and the project. The environment provides support at the planning stage by letting team members view their own status and that of their co-workers, such as which modules or enhancements each individual is working on. Further, like conventional PSEEs, the ASEE also supports the management viewpoint of project progress, showing who is responsible for each project and enhancement.

Because distributed-computing technologies change so rapidly, I developed a reference model of the ASEE architecture. To be successful, the ASEE must evolve to incorporate the best technologies and practices. Thus, over the years, we have developed several tools that have been integrated into the ASEE. A reference model is indispensable to the planning and design of such tools, and helps us incorporate them into the ASEE.

Further, the reference model separates concerns so that different layers can evolve independently. As Figure 3 shows, the model breaks down into three layers. Each layer relates directly to one of the three major issues that the environment's architecture addresses: collaboration, information sharing, and distributed computing. The management model shown in Figure 2 is embedded into these three layers.

1. *Collaboration.* The top layer provides, through ASP, a collaboration mechanism for multiple teams.
2. *Information.* The middle layer supports the distribution of product information across the Internet.
3. *Operations.* The bottom layer is a computing and communication infrastructure that extends across the corporate extranet and connects multiple subsidiary companies and software houses.

As Humphrey observed, software's quality depends on the individual developer.

The collaboration layer is implemented as a process-centered software engineering environment named Prime; the information layer is implemented as a Web-based distributed repository named WAIN. The operations layer is implemented across a common client–server platform that consists of standardized hardware and software. For the client, we use PCs running Windows; for the server,

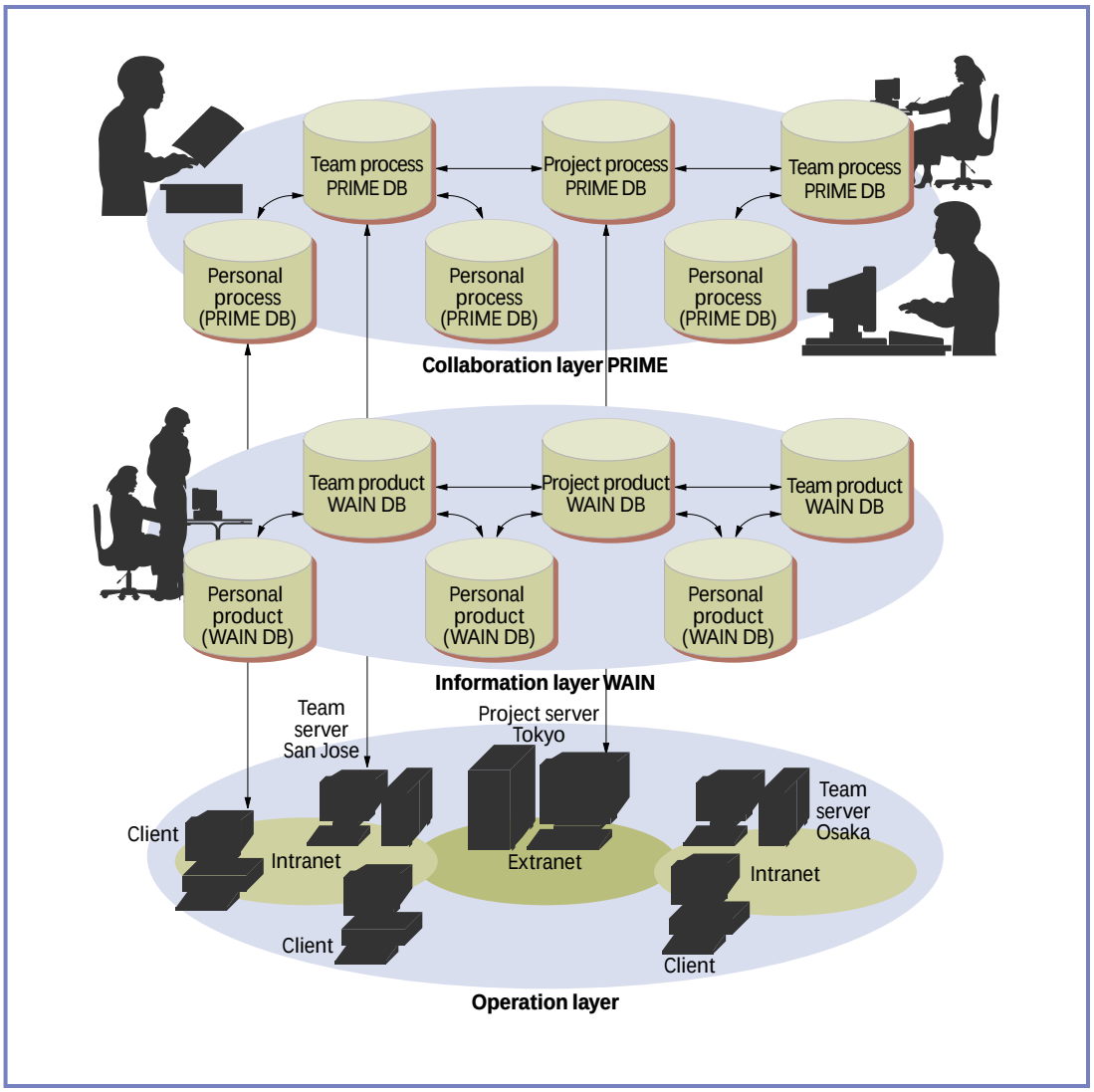


Figure 3. The Agile Software Engineering Environment's framework. Each layer of the framework relates directly to one of the three major issues that this architecture addresses: collaboration, information sharing, and distributed computing.

we use Unix workstations.

Based on the management mode and framework required to support the ASP management model, we developed ASEE's Web-based client–server software architecture, shown in Figure 4. The architecture consists of three layers: the client, the team server, and the project server. Individual developers and the team leader can access the client. The team server functions at the project level and can be accessed by the team leader and the project leader. The project server functions at the enhancement level and can be accessed by the project leader and upper-level managers.

ASEE is implemented using a federated multidatabase system in which every client's and server's database is built using the same schema structure. Thus, the collection and distribution of information across clients and servers occurs via the transmission of database replicas. Because all clients and servers share the same schema structure, it's possible to retrieve multiple databases using only SQL operations. Currently, we use Unix machines for our server platforms and Microsoft Windows machines for our client platforms.

Prime for the collaboration layer

Figure 5 shows the three layers of Prime's distributed hierarchical architecture. Prime

- provides just-in-time guidance of the individual developer's software process, as defined in a tree-structured YAC-II Process Description Language chart executed on the developer's workstation;
- supports planning and execution of the software process;
- supports, through its interlinked networks, the collection of statistics at appropriate levels of

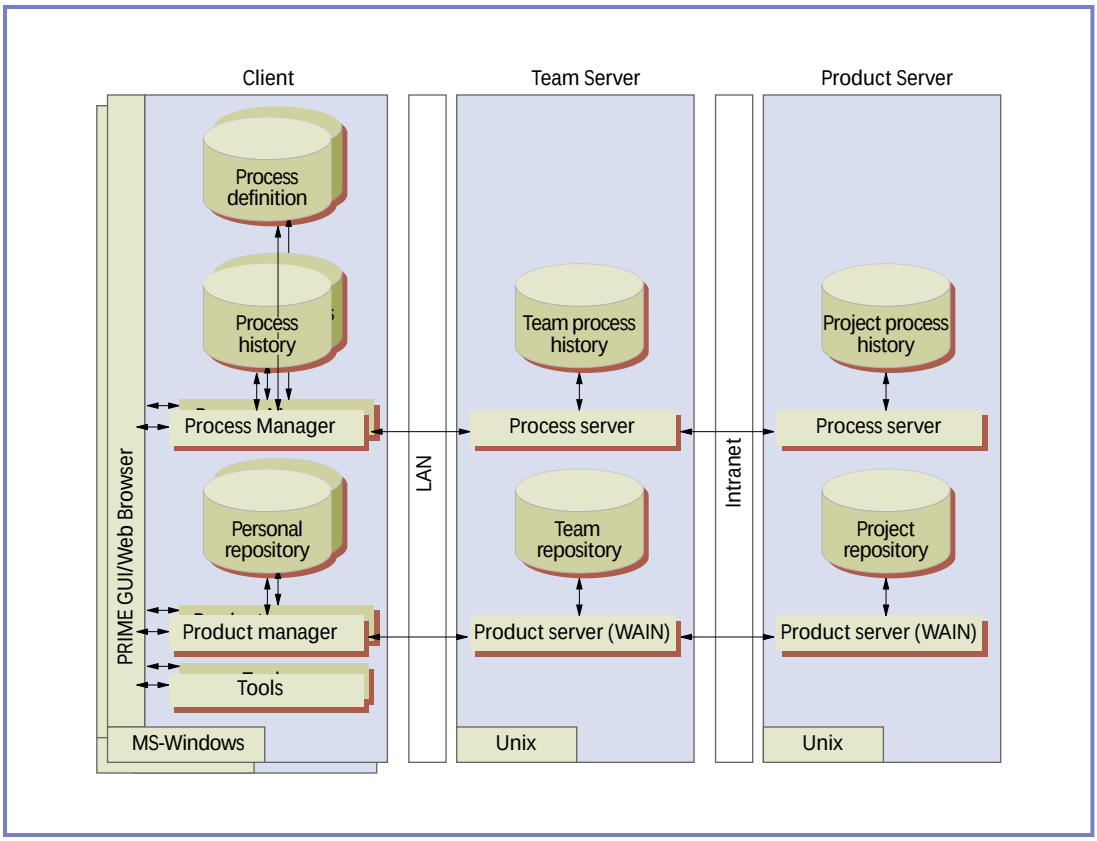


Figure 4. The Agile Software Engineering Environment's architecture.

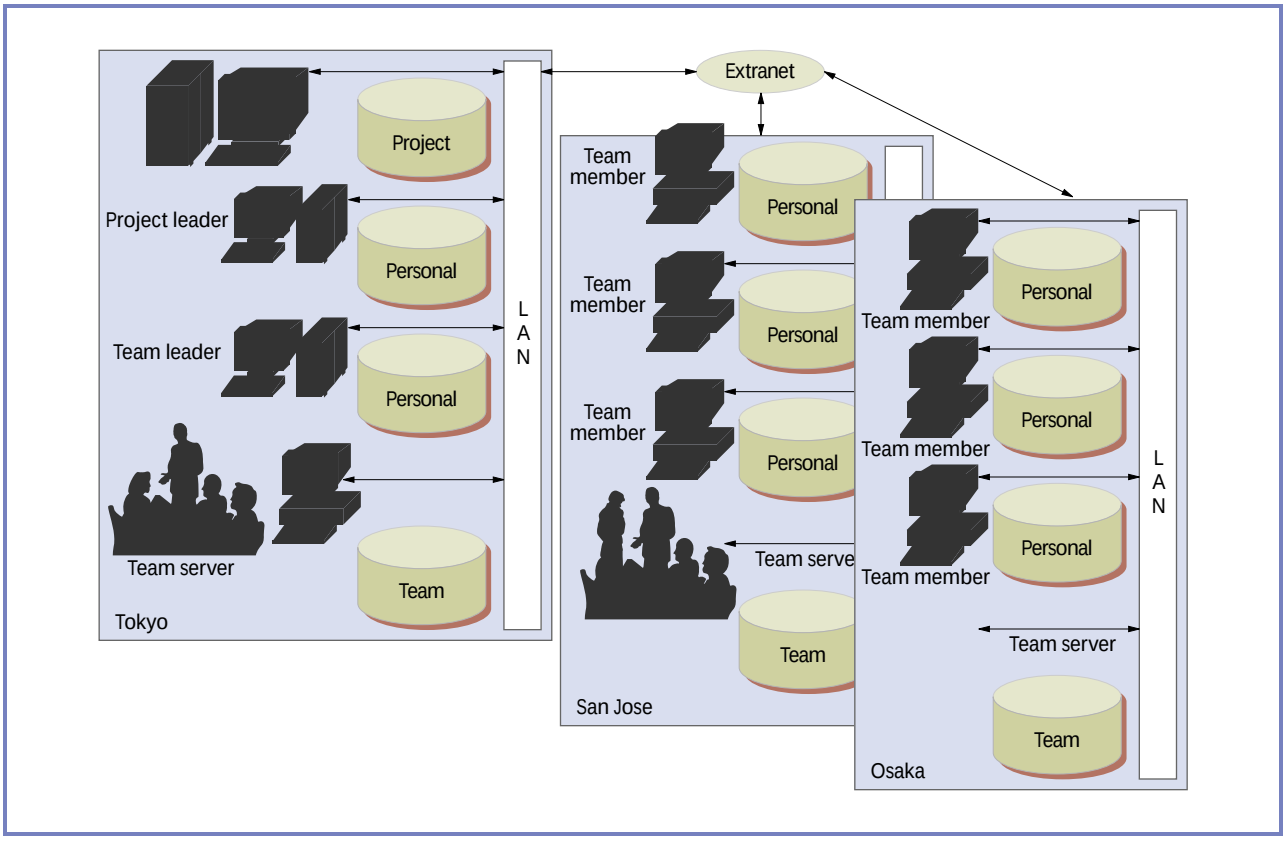


Figure 5. Prime's operational architecture.

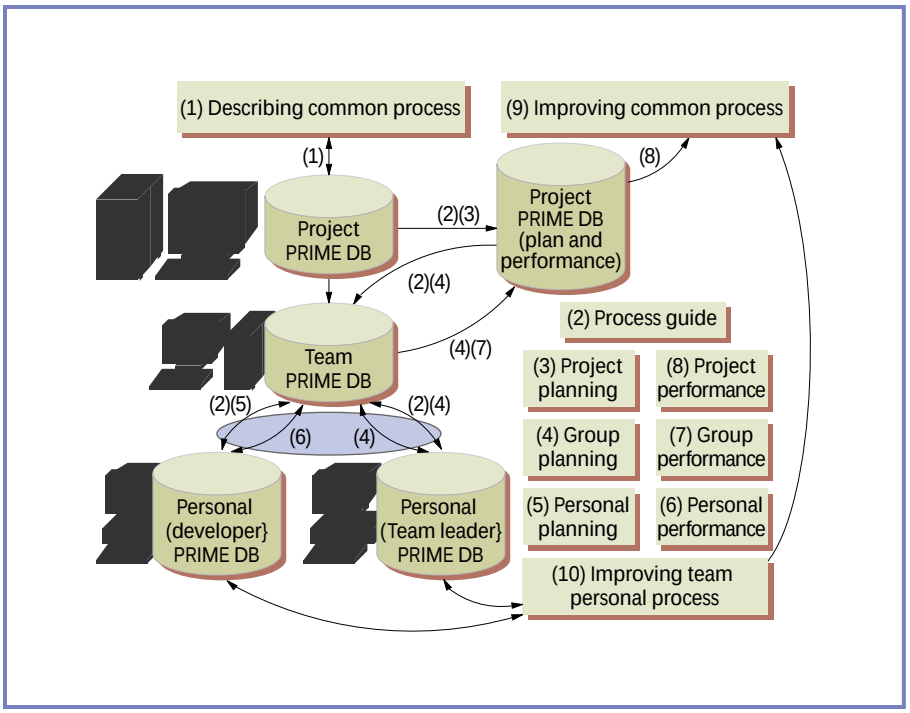


Figure 6. An overview of Prime's workflow.

process and organization;

- enables statistics visualization from multiple viewpoints at appropriate levels of process and organization; and
- controls security.

Figure 6 shows an overview of Prime's workflow; Figure 7 shows a screen view of Prime. The upper-left subwindow shows a list of base processes. The upper-right subwindow shows planning support. The progress of multiple teams and individual developers appear in the screen's left- and right-bottom subwindows, respectively. Prime tracks this data because it's critical to the software process's efficiency to visualize the dynamic behavior of multiple teams at the different levels of the organization's structure: project, teams, and individuals.

As an example of using Prime, Figure 8 shows three Progress and Bug charts from Project X: the large chart on the left provides an overview of the project, while the two smaller charts on the right track the performance of Teams A and B. The Progress and Bug chart, originally proposed by Joichi Abe and colleagues,[4] shows along the vertical axis of a single chart both the number of residual test cases (in a decreasing curve) and the number of bugs identified and fixed (in increasing curves). The vertical axis shows the number of test cases—some 3,000 for Project X—and the number of bugs is normalized to balance the chart. The horizontal axis shows the date. Because the number of bugs identified depends on testing progress, both progress and number of bugs must be viewable in a single chart. Thus, we introduced an extended Progress and Bugs chart into ASEE. Prime automatically collects test results through the extranet from the distributed team servers, then generates the daily Progress and Bugs charts at the team and project levels, as shown in Figure 8. The charts reveal that the progress and quality at the project level appear good, but that Team A is in trouble because the number of bugs it has uncovered exceeds the expected maximum value.

WAIN for the information layer

The middle layer of the ASEE provides a platform for the WAIN, which allows sharing of information over the Internet. From our experience developing CASE environments, we identified two major issues in WAIN's development.

- *Semistructured design information.* Conventional product management systems focus on either source code or well-structured design information such as dataflow diagrams and state transition diagrams. However, a huge volume of design information is not well-structured, but semistructured at best or unstructured at worst.[5] ASEE aims to manage consistently throughout the software process not only well-structured information, but also semistructured and unstructured information. The Web and e-mail systems are particularly suitable for such management.
- *Information sharing over the Internet.* To effectively share design information, we must first classify it, then identify which parts should be shared. Further, because design documents are stored in various formats and structures, we must encapsulate those documents' physical structure to ensure their portability. In addition to functioning as a supernetwork for file sharing across geographically distributed locations, the Web's hypertext structure helps us dynamically represent the relationship among information entities.

WAIN is designed to manage the structured, semistructured, and unstructured design documents posted to the corporate extranet.[5] Figure 9 shows WAIN's architecture. I first implemented WAIN with Gopher, then later ported it to the Web.

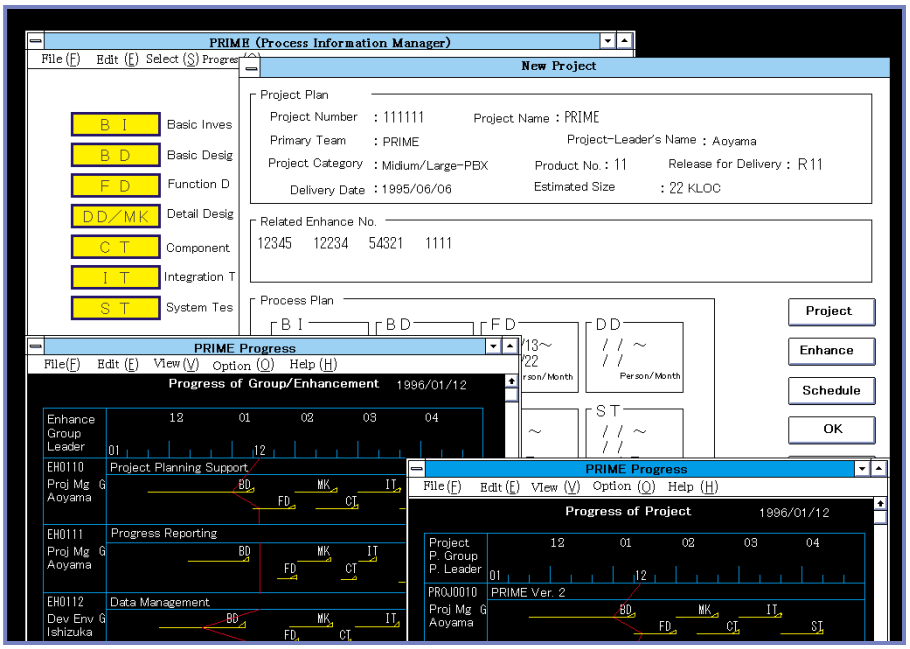


Figure 7. A screen from the Prime process-centered software engineering environment.

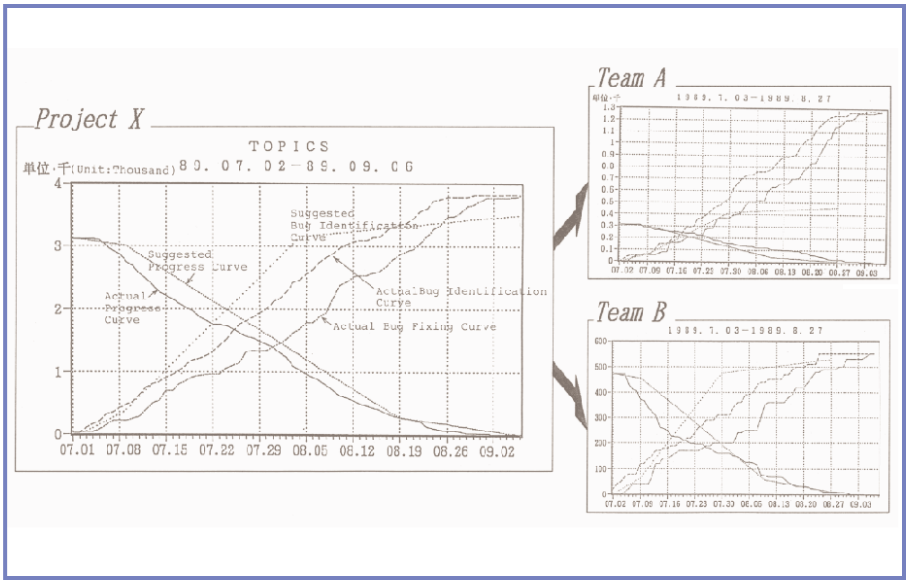


Figure 8. Three Progress and Bugs charts generated using Prime. The left chart presents an overview of the entire project, while the right two charts focus on the performance of Teams A and B.

Working in conjunction with Prime, WAIN

- encapsulates the structure of various design documents and provides uniform access methods to them;
- supports the creation, editing, retrieval, and change control of design documents;
- enables developers to navigate across the distributed repositories; and
- controls security.

To manage all three kinds of data in WAIN, I developed the categorization criteria shown in Figure 10. Data falls into one of three categories according to a document's logical and temporal characteristics: structured, semistructured, or unstructured. For each document a given project generates, we define a mapping from the data category to the storage system. Figure 11 shows a WAIN screen.

Lessons Learned

Since 1993, the Agile Software Engineering Environment has been used in a large-scale communications software project at Fujitsu. Prior to implementing the ASEE, we used a mainframe-based software engineering environment. However, in conjunction with the development of ASEE, we moved to a client–server environment. The ASEE itself has evolved since then. Because we set six-month intervals for our development cycle times, and three-month intervals for our delivery cycle times, we have delivered ASEE releases every three months for more than four years. In that time, we have learned some major lessons from ASEE's application.

- *Moving from a volume-based to a time-based perspective.* ASEE supports time-based management. Thus, time is a new and critical parameter in ASEE-based development. For example, previously, we needed a few hours to collect data from the distributed teams. Further, each team needed time simply to input the data. This delay meant that the data collected did not represent the team's current status and, because it was gathered asynchronously, sometimes lacked integrity. Such data flaws can cause critical problems in tasks such as the daily build,[6] in which staff must have access to accurate bug information each day. Thus, for ASEE to function properly and avoid these problems, we had to make it a real-time system by re-architecting the

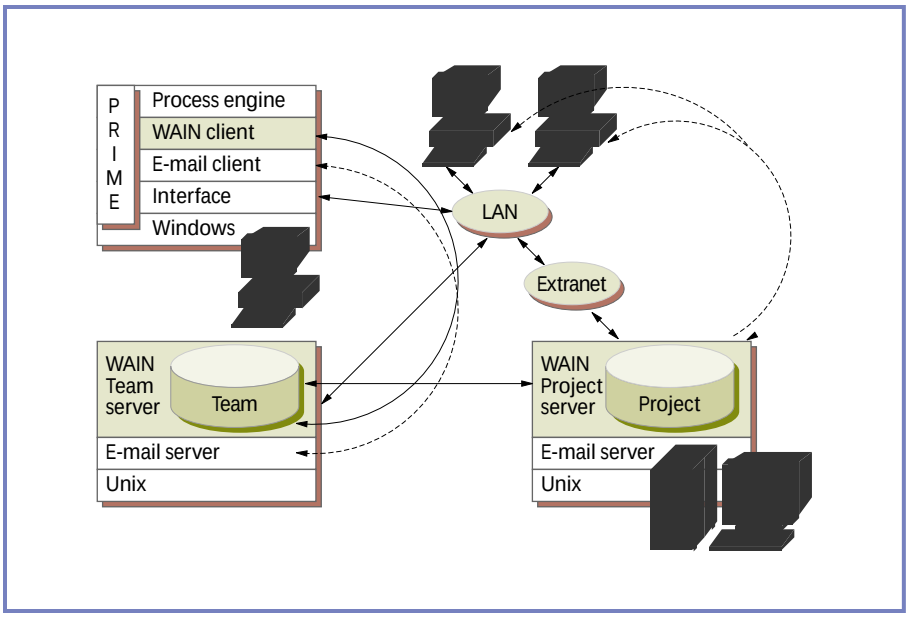


Figure 9. WAIN's operational architecture.

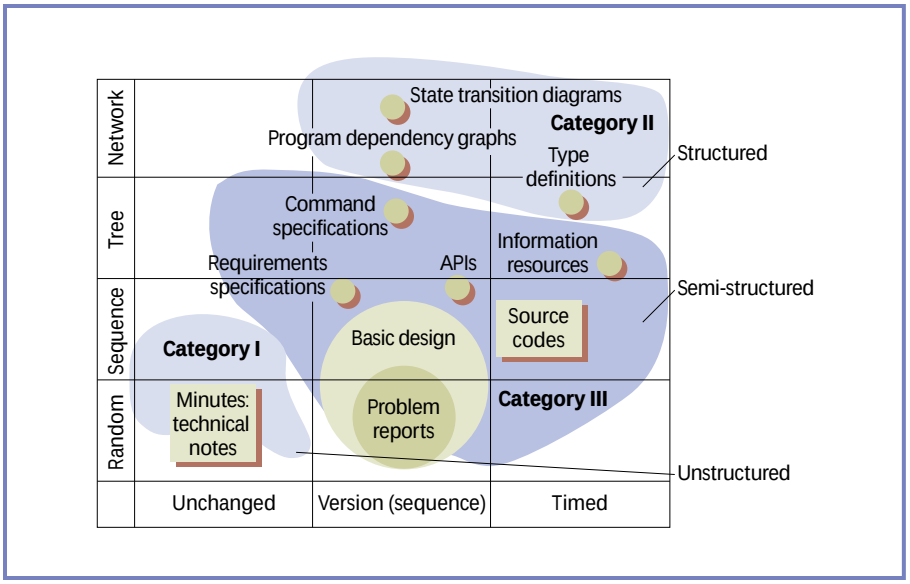


Figure 10. Categorization of design information.

software engineering environment from the time viewpoint.

♦ *Moving from enforcement to empowerment.* I believe that the mission of a process-centered software engineering environment is not the enforcement of methods but the empowerment of individuals. Many PSEEs and groupware packages force people to follow a specific order of processes or tasks.[7-9] We found, however, that software developers work back and forth across multiple processes and tasks.[10] Rigorous ordering at the detailed process level interferes with this pattern, causing inflexibility and loss of productivity. We observed that to be optimally effective, developers needed to iterate certain micro-processes under a specific level. Thus, we set up the management levels so that developers can avoid inflexible process enactment at the level of individual processes. The level structure also separates the concerns of the different management levels as they follow their preferred development methods. Drawing on an analogy from mechanical engineering, we call this the backlash mechanism because it allows for a limited freedom that avoids unnecessary friction and constraints. ASEE users adopted this concept readily.

♦ *Moving from point-contact to phase-contact.* Although an integrated, conventional software engineering environment supports only specific points of development work, a PSEE must support the everyday life of software development. Thus, usability is a critical issue in PSEE design. Significantly, the largest number of change requests submitted by ASEE users concerned usability issues.

♦ *Shifting from the manager's to the developer's viewpoint.* People often regard PSEE as a management tool. However, using the PSEE as a management tool sometimes runs afoul of the for-the-sake-of-management syndrome, which causes developers to view it as an unwelcome burden that interferes with their "real" work. To avoid this reaction, we designed ASEE to provide support for the developer's viewpoint as well, so that it helps both developers and managers.

♦ *Enhancing security.* ASEE has been used by multiple companies over the extranet. Because some information is sensitive, we provided a high-security version of ASEE that protects some information.

♦ *Moving to the Web.* WAIN's initial implementation was Gopher-based. However, after Web technology became popular we integrated ASEE into the Web, which is particularly suited to managing design documents because they are not well structured. However, our work in this regard is not finished: we have yet to add various support features specific to

software engineering documents, such as configuration management and tool integration.

The ASEE, which supports the ASP over the Internet, provides just-in-time management of the software development process by providing the right process and product information to the right people at the right time. Because time-to-market offers a competitive edge, I believe that agility will become an increasingly important aspect of software engineering environments. Thus, we must re-architect our software engineering environments from a time-sensitive viewpoint. Further, ASEE is network-centric and promotes a virtual software factory of multiple teams geographically distributed over the Internet.

Since 1996, I have been involved with the Software Continuous Acquisition and Life-Cycle Support consortium sponsored by the Information Technology Promotion Agency in Japan. Unlike the CALS projects in other countries, Japan's CALS is a purely commercial initiative. Its fundamental philosophy is open and global. It strives to integrate best practices and tools over the Internet so that people can collaborate to quickly deliver products across companies and across countries.

Software organizations have discovered that speed is a strong weapon. To remain competitive, developers must work in real time; their processes must be agile. The Internet has proven the perfect environment for such development, allowing teams to collaborate around the clock, around the world. Thus, Web-based agile software engineering has become an emerging discipline.[11] ❖

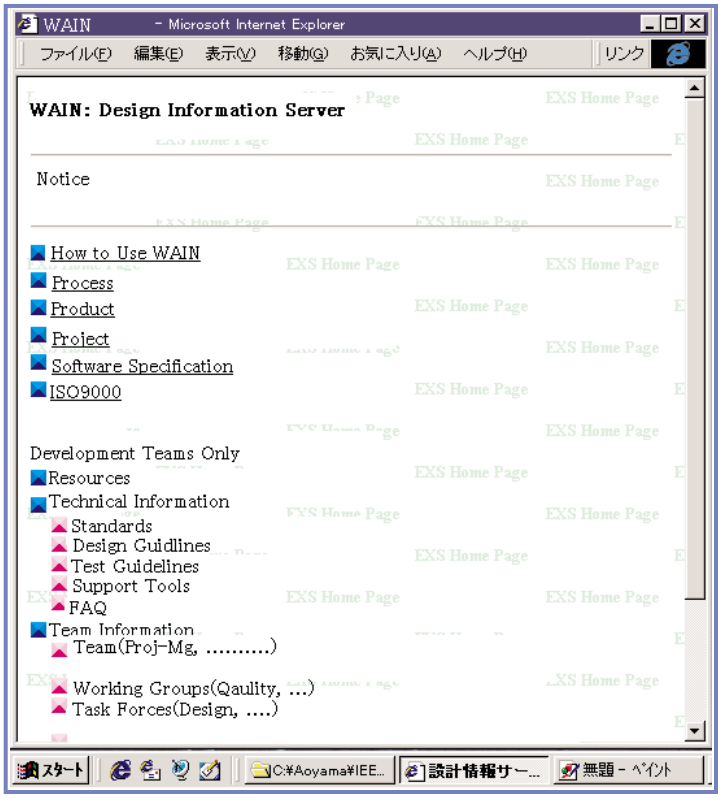


Figure 11. Sample screen from WAIN, the wide area information network.

Acknowledgments

I thank Kiyoo Nakamura and Carl K. Chang for their continuous encouragement. Thanks also to many of my colleagues at Fujitsu for their participation in the development of ASEE.

References

1. M. Aoyama, "Concurrent-Development Process Model," *IEEE Software*, July 1993, pp. 46-55.
2. M. Aoyama, "Agile Software Process and Its Experience," *Proc. 20th ICSE*, IEEE Computer Soc. Press, Los Alamitos, Calif., Apr. 1998, pp. 3-12.
3. W.S. Humphrey, *Introduction to the Personal Software Process*, Addison Wesley Longman, Reading, Mass., 1997.
4. M. Aoyama, "Sharing and Managing the Design Information in a Distributed Concurrent Development of Large-Scale Software Systems," *Proc. IEEE COMPSAC '96*, IEEE Computer Soc. Press, Los Alamitos, Calif., Aug. 1996, pp. 168-175.
5. J. Abe et al., "An Investigation of the Nature of the PB Curves of Software Projects," *Trans. Information Processing Soc. of Japan*, Vol. 20, No. 4, July 1979, pp. 306-313.
6. M.A. Cusumano and R.W. Selby, *Microsoft Secrets*, The Free Press, New York, 1995.
7. *Software Process Modelling and Technology*, A. Finkelstein et al. (eds.), John Wiley and Sons, New York, 1994.
8. P.K. Garg and M. Jazayeri, *Process-Centered Software Engineering Environments*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1995.
9. L.J. Osterweil, "Software Processes Are Software Too," *Proc. 9th ICSE*, IEEE Computer Soc. Press, Los Alamitos, Calif., Mar. 1987, pp. 2-13.
10. D.E. Perry et al., "People, Organizations, and Process Improvement," *IEEE Software*, July 1994, pp. 36-45.
11. *Proc. Workshop on Software Eng. over the Internet*, F. Maurer (ed.), http://sern.cpsc.ucalgary.ca/~maurer/ICSE98WS/ICSE98WS.html.

About the Author

Mikio Aoyama is a professor at the Department of Information and Electronics Engineering, Niigata Institute of Technology. Previously, he worked at Fujitsu from 1980 to 1995. His current research interests include component-based software engineering, OO methodology, and process engineering.

Aoyama received a BS and an MS in electronics from Okayama University He is a member of the IEEE Computer Society, ACM, Information Processing Society ofJapan, and the Institute of Electronics, Information, and Communications Engineers, Japan.

Address questions about this article to Aoyama at mikio@iee.niit.ac.jp.