Collaborative Development Environments

Grady Booch and Alan W. Brown

Rational Software Corporation

October 28, 2002

Table of Contents

Abstract

A ***collaborative development environment*** (CDE) is a virtual space wherein all the stakeholders of a project – even if distributed by time or distance – may negotiate, brainstorm, discuss, share knowledge, and generally labor together to carry out some task, most often to create an executable deliverable and its supporting artifacts. CDE's are particularly useful as places where engineers may collaborate to solve problems. Here we focus on software developers in their tasks of designing, implementing, deploying, and maintaining high quality software-intensive systems where they are physically separated and make use of the Internet as the basis for their interactions.

In this paper, we examine the points of friction in the software development process and the mechanisms that reduce that friction. We then survey a variety of sites, both inside and outside the software domain, which provide some of these mechanisms. We conclude with observations as to what a CDE is, what it is not, and what it can become.

1. Introduction

A ***collaborative development environment*** (CDE) is a virtual space wherein all the stakeholders of a project – even if distributed by time or distance – may negotiate, brainstorm, discuss, share knowledge, and generally labor together to carry out some task, most often to create an executable deliverable and its supporting artifacts.

Collaboration is essential to every engineering domain. What we are interested in, and what we focus on in this paper, are engineers working to solve problems collaboratively. More specifically, we are focused on software engineers in their tasks of designing, implementing, deploying, and maintaining high quality software-intensive systems where they are physically separated and make use of the Internet as the basis for their interactions. This scenario is commonplace, fueled by outsourcing, integration of third party software, increasing off-shore development, use of home offices, strategic partnerships among companies, etc. The success of distributed teams working together effectively is imperative, and is a distinguishing factor in the success or failure of many modest to large software development organizations.

From its earliest days, collaboration has been an essential part of the fabric of the Internet: email, instant messaging, chat rooms, and discussion groups are common collaborative elements that already exist, and so in one regard, there is nothing new or novel here. Furthermore, collaboration among teams is already common through the use of an increasing number of features embedded into common desktop products such as office suites (e.g., Microsoft Office Suite and Sun StarOffice), and communications packages (e.g., Microsoft Outlook and IBM LotusNotes). In both, there is ample support for shared document reviews, distribution of documents among teams, and some mechanisms for performing common collaborative tasks. In a very practical sense these tools provide the baseline of collaboration functionality for today's software developers, though typically augmented with an assortment of open source, proprietary, and commercial stand-alone point products.

However, what is different about the emergence of CDEs is the collection of these mechanisms into a virtual project space, delivered over the Web and supplemented with tools and creature comforts that are focused on the mission of that particular team, in this case, the creation of software products. Communities of practice are fragile things that can flourish only given the right balance of technology and user experience, and thus creating a CDE that enables efficient, creative teamwork is a hard thing. It is the supportive, integrated nature of a CDE that distinguishes it from the variety of disparate functional products typically in use today.

Ultimately, the purpose of a CDE is to create a frictionless surface for development by eliminating or automating many of the daily, non-creative activities of the team and by providing mechanisms that encourage creative, healthy, and high-bandwidth modes of communication among a project's stakeholders. In this paper, we examine the points of friction in the development process and the mechanisms that reduce that friction. We then survey a variety of sites, both inside and outside the software domain, which provide some of these mechanisms. We conclude with observations as to what a CDE is, what it is not, and what it can become.

2. The Physics of Software

Many aspects of software engineering have changed little in the past twenty years. Re-reading some of the seminal works in software engineering, it is striking just how much of that work remains relevant to today's challenges for developing quality software on time and within budget: abstraction, information hiding, and having a good separation of concerns are all essential principles that apply to every manner of software system; additionally, the requirements management, change control, architecture, iterative development are all well-understood principles of the software development process. However, in other ways the past two decades has witnessed many advances in the technology, the business, and the practice of software engineering.

It is useful to reflect back on the nature of software development some twenty years ago, before the creation of the Unified Modeling Language (UML), the Rational Unified Process (RUP) and the methods that preceded it. This was the time before the Web was a central part of the software development landscape and before the existence of contemporary programming languages such as Java, C++, and C#. Indeed, this was even the time before object-orientation had entered the mainstream of development, although the roots of object-orientation were already present in languages such as Simula and Smalltalk and in abstract data type theory. Integrated development environments (IDEs) were just emerging, with many developers still toiling with command line tools and WYSINWYG (What You See Is Not What You Get) editors. Methodologically, the industry was focused primarily on improving the productivity of the individual developer, although some individuals, most notably Gerald Weinberg, Tom DeMarco, Tim Lister, Larry Constantine, Ed Yourdon, and Fred Brooks were urging us to consider the human side of development.

The world of software development has clearly progressed: the Web and its related technologies are a factor in virtually every project, object-oriented methods and languages are mainstream, IDEs are far more powerful, and methods have begun to address the social dynamics of the development team. Still, as Walker Royce describes it, software development remains an industry that is marked by a diseconomy of scale. In other words, as project functionality, size, and complexity rise, the incremental cost of each new line of code or additional feature tends to increase, not decrease, as we might expect in a classical engineering setting. Furthermore, economic pressures that demand faster time to market often collide with the engineering demands for higher quality.

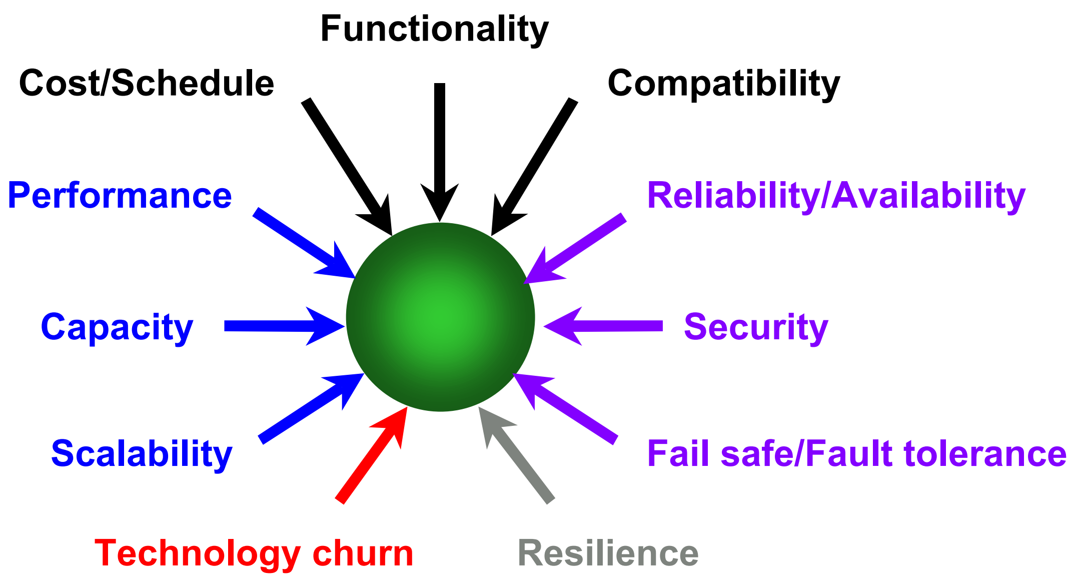


Figure 1: Forces in Software

Figure 1 illustrates a number of pressures that weigh upon and the development team as a whole. Cost, schedule, functionality, and compatibility (with legacy code and with packaged software, including operating systems and middleware, both of which are outside the primary control of the development team) are the most pressing forces. Depending upon the particular nature of the problem domain, performance, capacity, scalability, reliability, availability, security, and fault tolerance weigh in with varying amounts. Technology churn represents the force caused by the rate of change of packaged software, protocols, and hardware, also all outside the control of the development team (except for the choice of what technology to be used). Finally, resilience represents the force caused by the need to construct systems that must accommodate continuous change, with different parts of the system changing at different rates of speed.

For systems of any reasonable complexity, no one person can efficiently counteract these forces. Thus, for most systems under construction, operation, or revision, software development is a team sport. In these circumstances inter- and intra-team interaction, communication, and dynamics play as least as important a role as individual heroics in successful software development. For this reason, understanding more about optimizing software development team performance is a critical task of software engineering.

There are no hard studies that tell us the median size of a contemporary development team, but our experience, across a broad range of domains, is that teams of four to eight people are most common, with teams of one or two being the second-most common. Beyond the range of four to eight members, the existence of larger teams tends to tail off although we also find a peak in the curve of team size/occurrence somewhere around the one to two hundred mark. Some systems, such as telephony, financial, and command and control systems are so complex that they require large teams to complete the work in a timely fashion.

To be honest, these apocryphal figures are not especially useful, because they focus only upon the immediate development team. In practice, most organizations build systems, not just applications. Furthermore, for any interesting system that has deep economic value to the development organization or its users, there are a number of stakeholders beyond the core programming team who have a say in the development process. Web-centric systems are especially sensitive to this factor, for the extended team will typically involve network engineers, security architects, and content creators. If we take these elements into account, our experience is that the median team size for building systems is much larger, somewhere in the range of several dozen to one hundred or more.

Not only must an organization focus upon the efficiency of each individual team, it must also be concerned about the efficiency of its teams of teams. In even a modest size development organization, there might be a hundred or so developers perhaps organized in teams of four to eight, with most focused on point products but a few focused on infrastructure technologies that support all the other teams [1]. In practice, most of these individual teams will themselves be contiguous (that is, their cubicles are physically close to one another), which encourages the jelling of the team through the myriad of informal interactions that occur during the day. However, relative to one another, these teams will typically be physically disconnected from other teams, thus reducing the level of informal contact and in turn reducing the bandwidth and quality of inter-team communication. Indeed, one of the problems any such organization faces is simply keeping these teams of teams on the same page. That requires sharing project status, reducing duplication of work, engineering the architectural seams among groups, and sharing knowledge and experience.

In short, delivering better software faster involves the rhythms of the individual developer, the small team of developer, and – for larger projects – teams of teams and even teams of teams of teams of developers.

Relative to software development two decades ago, we now better understand the best practices of software development that work and those that do not work. However, as Gerald Weinberg has noted, ultimately, programming is a human activity. All meaningful development work derives from activities that beat at different rhythms: the activities of the individual developer, the social dynamics among small teams of developers, and the dynamics among teams of teams. Thus, even if we use the most expressive languages, the most comprehensive packaged software, and the best methods, it is still the manual labor of the development team that yields systems of quality.

Software developers spend a majority of their time on code-centric activities supported by an IDE offering a range of code development and manipulation features. Other aspects of their task involving interaction, communication and coordination within and across teams are generally supported by a discrete combination of capabilities including configuration management systems, issue tracking databases, instant messaging systems, project websites, and so on. Assembled in a coherent fashion, this latter set of capabilities can compose a collaborative development environment (CDE) for software engineers.

Whereas traditional IDEs focus upon improving the efficiencies of the individual developer, CDEs focus upon improving the efficiencies of the development team as a whole. While it is the case that most modern IDEs have some degree of collaborative support (for example, the ability to check in/check out an artifact from a common repository or call out to NetMeeting and whiteboards from a menu), we contend that incrementally adding such collaborative features will not transmogrify an IDE into a CDE. IDEs are essentially developer-centric, meaning that their primary user experience focuses on the individual developer, whereas CDEs are essentially team-centric, meaning that their primary user experience focuses on the needs of the team (but with points of entry for different individuals). Psychologically, this is a subtle yet very important shift of perspective.

3. A Day in the Life of a Developer

In order to best understand the requirements for a CDE, it is reasonable to first understand the social dynamics of the individual, the team and a team of teams. Surprisingly, however, there exist very few empirical studies that highlight what developers really do with their time. There are some soft studies, such as found in the experiences of Gerald Weinberg [2], Tom DeMarco [3], and Larry Constantine [4]. Larry Votta has proposed a framework for such scientific study [5], although no deep studies from that framework have yet been release.

There is one interesting empirical study that can be found, carried out by Joerg Strubing, a sociologist at the University of Berlin. In his study [6], he observed, "Being a sociologist, I have found that designing software is a highly cooperative process." His study consisted of two series of experiments. In the first series, he conducted 10 open-ended interviews and one group discussion; in the second series, he conducted 25 interviews with programmers and two other experts.

Although his sample size was relatively small, Strubing found three activities that developers consistently carried out beyond just coding: organizing their working space and process, representing and communicating design decisions and ideas, and communicating and negotiating with various stakeholders. Thus, beyond coding, Strubing declared programming to be a profession that involved very heterogeneous activities, the management of ambiguity, and a significant degree of negotiativeness.

Because of the scarcity of hard data, Booch conducted an experiment in March 2001 to obtain a snapshot of the daily life of a developer. This experiment was carried out on the Web and involved 50 developers from around the world. Artifacts generated for this experiment included a pre-day survey, timesheets during the event, a digital photo of each participant taken at noon local time on the day of the survey, and one or two snapshots of their desktop. Here we provide a short summary of the main results of that study.

Participants were 81% male and 19% female with most coming from the United States but others from countries including Austria, Germany, Canada, India, Australia, New Zealand, Venezuela, and the Russian Federation. On average, participants had nine years of experience in software development, ranging from one to 43 years. Project domains varied widely, encompassing command and control, financial, communications, and entertainment industries. 46% of the participants worked on Unix, 41% on Windows, 7% on Linux, and 5% on pure Web applications [7]. 37% of the participants worked in Java, 35% in C, C++, or C#, and 28% worked in other languages, most commonly Perl, Visual Basic, Ada, and assembly language.

In analyzing their surveys and time sheets, participants spent about 16% of their day in analysis (ranging from 5% to 40%), 14% of their day designing (ranging from 1% to 40%), 16% of their day coding (ranging from 0% to 60%), and 10% of their day testing (ranging from 0% to 50%).

Beyond these traditional activities, what's particularly interesting is the time each spent on infrastructure tasks. On average, participants spent 3% of their day on the phone and 7% reading (email, snail mail, documents, journals, and magazines). The survey asked participants to distinguish between productive and useless meetings: on average, 10% of their time was spent in productive meetings and a disturbing 7% was spent in useless ones.

This study on day in the life of a developer resonates with Strubing's research: software development is a deeply social process. Whereas IDEs such as Microsoft's VisualStudio, the open source NetBeans and Eclipse, IBM's WebSphere Studio, and Borland's JBuilder are primarily individual productivity tools, there is emerging a genre of development environments we call collaborative, because they address the requirements of this social process.

4. The Emergence of Collaborative Development Environments

To paraphrase Abraham Lincoln, software development takes place of the Web, by the Web, and for the Web. There is considerable development *of* the Web's infrastructure; similarly, a great deal of application software is being developed *for* Web-centric systems. Relative to CDEs, however, development *by* the Web means using the Web to change the nature of software development itself.

If we project out from the most common IDEs such as Microsoft's Visual Studio and IBM's open source Eclipse, we observe that emerging CDEs are and should be Web-based, artifact-centric, and multi-dimensional. The Web is an ideal platform for doing software engineering because the very nature of the Web permits the creation of virtual spaces that transcend the physical boundaries of its participants. CDEs should also be artifact-centric, meaning that they should offer up a user experience that makes work products primary and engages tools only as necessary [8]. Finally, a CDE should be multidimensional in the sense that different stakeholders should be offered different views, each adapted to that stakeholder's specific needs.

Collaborative sites both on and off the Web have existed for some time [9], but we began to see collaborative sites focused solely on software development starting only about five years ago. Most of these sites were neither public nor reusable, but rather were one-off creations for specific projects. One of the earliest such sites we encountered was for a large command and control system. As part of an architectural review, we were placed in front of a homegrown intranet that contained literally every artifact created by the project, from vision documents to models to code to executables. Although this site offered very little in terms of collaborative mechanisms, it did offer up a virtual presence, a veritable electronic meeting place for the project's team members, many of whom were geographically distributed.

Soon after, we saw emerge commercial sites for the construction and CAD industries, both using the Web to provide a virtual project space. Similar sites grew up for the open source software development industry. Indeed, since much open source code is written by individuals who never interact with one another in person but only via email and the Web, it is natural that the Web be used to provide a sense of presence for open source projects.

Recognizing the emergence of the Web as a platform for software development, Frank Maurer, an associate professor at the University of Calgary, began sponsoring a workshop on software engineering over the Internet in conjunction with the International Conference on Software Engineering. His workshops have been conducted since 1998, and have generated several dozen important papers on the theory and practice of CDEs. Many of the mechanisms visible in the sites we survey later in this paper had their roots or were described in these workshops. In addition to Maurer's workshops, A. J. Kim [10] and Adele Goldberg [11] have independently contributed to the practice of building communities on the Web that, relative to CDEs, offer valuable insights into the social dynamics of groups and the elements necessary to make such communities flourish. Finally, the MIT Media Laboratory [12] has conducted basic research regarding collaboration and the Starfire [13] project, led by Bruce Tognazzini while he was at Sun Microsystems, offers a future vision of collaborative development that is worth study.

It can also be noted that collaborative development environments are beginning to get traction in traditional consumer product development [14]. Proctor & Gamble, Johnson & Johnson, Ford, SCI Systems, and The Limited, among many others, are all currently using a variety of collaborative mechanisms to manage their research, design, manufacturing, and shipping needs. Clearly, the artifacts that these companies manipulate are not the same as one would find in a software development organization, but it appears that many of the underlying collaborative mechanisms are identical. This is good news, for it means that there is likely a broad market for general CDEs, which can thus raise the level of practice for software-specific CDEs.

5. Creating a Frictionless Surface

The purpose of a CDE is to create a frictionless surface for development. Our experience suggests that there are a number of points of friction in the daily life of the developer that individually and collectively impact the team's efficiency:

- The cost of start up and on-going working space organization
- Inefficient work product collaboration
- Maintaining effective group communication, including knowledge and experience, project status, and project memory
- Time starvation across multiple tasks
- Stakeholder negotiation
- Stuff that doesn't work

We call these points of friction because energy is lost in their execution which otherwise could be directed to more creative activities that contribute directly to the completion of the project's mission. Addressing these points of friction represents substantial return on investment for many organizations.

The *costs of start up* are related to Strubing's observations concerning organizing the working space. As a team gets started or as new team members join a project, there is always an awkward, disorienting period of time that the member gets settled into his or her space. Finding the right tools, knowing who to talk to, knowing what documents to read first, understanding the current state of the project are activities that all take time, and are especially painful if the project or member is not offered any guidance as to where to begin.

Work product collaboration involves the friction associated with multiple stakeholders creating a shared artifact. Often, one person is given the responsibility for owning a particular artifact and so serves as its primary author. Over time, however, critical documents involve many people in their evolution. Keeping track of changes, knowing who changed things and why, synchronizing simultaneous edits, and in general handling accountability for the life of the artifact are all activities that cost time and that can create inefficiencies if not automated.

Communication is perhaps the largest point of friction. As Strubing noted, "negotiativeness" and the management of ambiguity are both critical non-programming tasks, and both are at the core of sound communication. Typically, the memory of a project, its culture, and its design decisions are locked up in the heads of a few core individuals, and part of a team's activities involve sort of a tribal sharing. Insofar as such knowledge is inaccessible, the depth of information flow will suffer. Furthermore, insofar as the communication paths among team members are noisy – such as it is within teams of teams – communication quality and hence team efficiency suffers.

Time starvation refers to the reality that there is typically never enough time to complete everything on an individual's to do list. Developers are finite in their capacity to work. Although time cannot be expanded, projects on a death march will try to squeeze out every possible cycle by pushing those human limits, typically at great human expense (which is why death marches are not sustainable) [15].

Stakeholder negotiation involves the time necessary to drive different members of the team having different worldviews to some degree of closure so that the team can make progress. Within a project, time will always be spent on explaining to various stakeholders what's being built, the precise desired structure and behavior of the system, and the semantics of design alternatives and ultimately design decision. In short, stakeholder negotiation is the process of driving out ambiguity.

Surprisingly little has been written about the impact of *technology and tools that don't work* upon the efficiency of the development team. Intermittent network outages, operating systems that behave in mysterious ways as if possessed, buggy packaged software, tools that don't work quite as advertised all eat up a teams time. Such interruptions are often small, but any loss will interrupt the concentration of the developer and, ultimately, losses will mount up, one minute at a time.

A CDE can address many of these points of friction. Making a virtual project environment just an URL away can minimize start up costs; being able to self-administer such sites also means that the team can manage its own artifacts rather than require the services of a dedicated support team. The friction associated with work product collaboration can be minimized by offering up artifact storage with integrated change management and the storage of metaknowledge. Communication can be facilitated by the use of mechanisms for discussions, virtual meetings, and project dashboards. Time starvation can be addressed not only by a hundred small creature comforts for the developer, but by making possible virtual agents that act as non-human members of the development team, responsible for carrying out scripted, tedious tasks [16]. Stakeholder negotiation can be facilitated by mechanisms that automate workflow. As for stuff that doesn't work, well, a CDE won't make much different: the best we can suggest is that you should simply refuse to buy or use products of inferior quality [17]. That notwithstanding, if stuff doesn't work for you, then it is likely that there are others in the world who have experienced the same problem and might have solved it or found a work around. In the presence of an extended community of developers, such as might interact with a CDE, mechanisms for sharing experiences can somewhat temper the problems of hard failure (and perhaps even offer a form of collective bargaining to put pressure on the vendor who has delivered up shoddy products).

Ultimately, technology is valuable to an organization insofar as it provides a meaningful return on investment. With CDEs, there exists a potential for both tangible and intangible returns [18]. Tangibly, there exist opportunities for real reduced costs in start up, tool administration, and artifact administration. In difficult economic times, a company's travel budget is invariable under pressure, yet collaborative work must continue. The presence of a CDE can actually reduce these company and human costs by eliminating the need for some travel. Intangibly, a CDE provides a sense of place and identity for the organization's nomadic developers who may be geographically distributed and mobile; such a space helps jell the team. Furthermore, in examining the patterns of communication within teams, it appears that healthy organizations create and collapse small tiger teams all the time for the purpose of wrestling specific problems to the ground. Facilitating the management of such teams permits greater accountability and focus of mission.

6. A Survey of Collaborative Sites

There exist only a few commercial CDEs focused primarily on the problem of software development over the Internet, but there are many more that have been generated for other domains or that address one specific element of the software CDE domain. Studying these different sites can help us understand what a CDE is, what it is not, and what it can be.

We can classify the spectrum of CDEs as follows, and examine exemplars of each in turn:

- Non-software domains
- Asset management
- Information services
- Infrastructure
- Community
- Software development

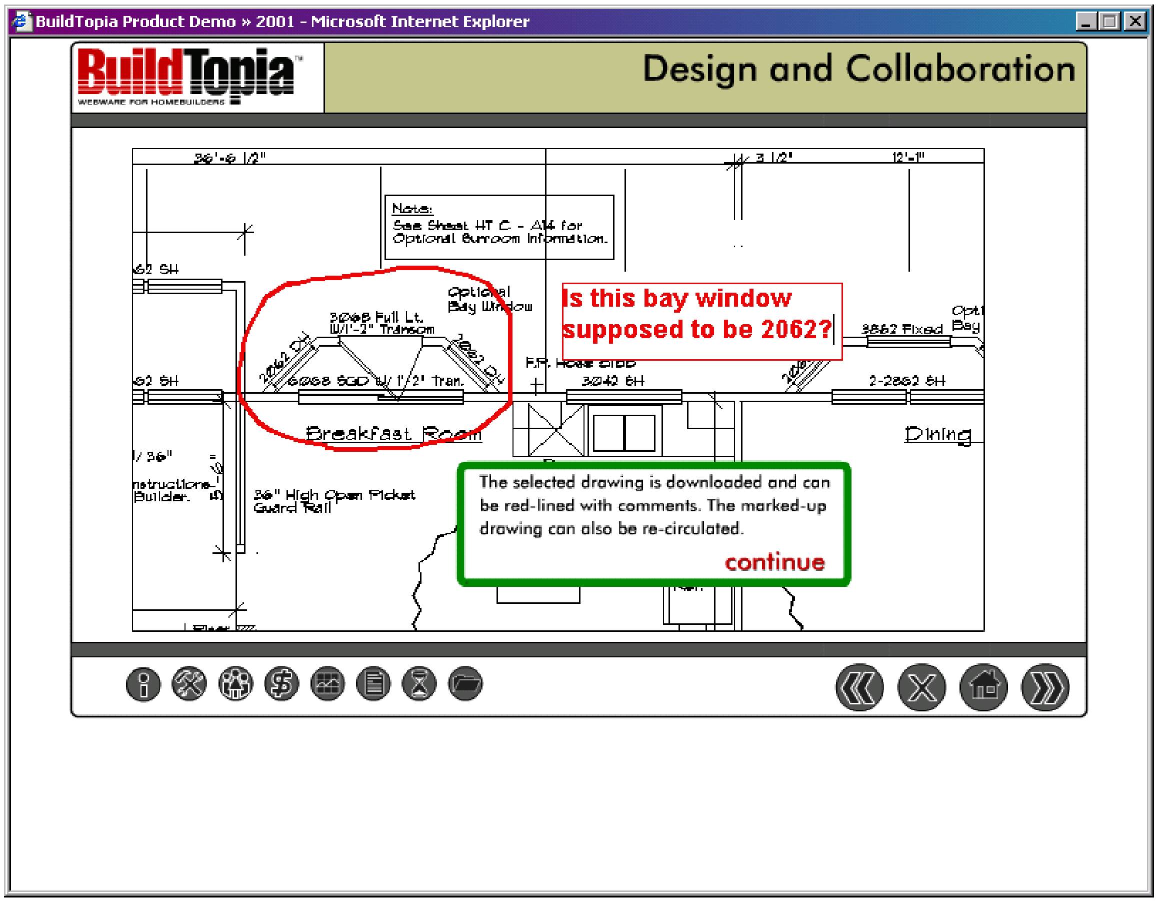
6.1 Non-software domains

The construction, manufacturing and electronics industries have been a fruitful place for the evolution of CDEs. In fact, it is these industries that have been the earliest adopters of collaborative technology, perhaps because many of their points of friction are directly addressed by the features of a CDE. Imagine, for example, a building being erected in Kuala Lumpur. On site, the construction supervisor might encounter a design problem whose resolution would require the interaction of the building's architect, structural engineers, and end users (and, amazingly so, lawyers). If the architect were in London, the structural engineer in New York, and the client in Hong Kong, getting these stakeholders together in real time would be problematic. Instead, by using the Web as a virtual meeting place for the project, resolution can take place in real time.

This kind of collaborative development is what lies behind products such as BuildTopia [19], a Web-centric system for design collaboration, project management, bidding, sales, and customer support for the homebuilding industry. As Figure 2 shows, BuildTopia's product line permits Web-based collaboration on design artifacts such as blueprints.

Within a project site, users can manipulate diagrams in real time as well as manage the workflow and artifacts of a set of building projects.

The manufacturing and electronics industries have similarly pioneered a number of Web-centric collaborative solutions. For example, CoCreate [20], a subsidiary of Hewlett Packard, offers a product called OneSpace, which delivers a virtual conference room. In their words, "through the use of shared views, pointers, and annotations, product development partners can concurrently engineer through co/viewing, co/inspecting, and co/modeling. iCadence [21] is a similar product line directed to the collaborative engineering of electronic devices.


BuildTopia Product Demo » 2001 - Microsoft Internet Explorer
BuildTopia
WEBWARE FOR HOMEBUILDERS
Design and Collaboration
Note: See Sheet HT C - A14 for Optional Sunroom Information.
Optional Bay Window
3068 Full Lt. W/1'-2" Transom
6068 SGD W/ 1'-2' Tran.
Breakfast Room
3042 SH
2-2862 SH
3862 Fixed
Dining
36" High Open Picket Guard Rail

Is this bay window supposed to be 2062?

The selected drawing is downloaded and can be red-lined with comments. The marked-up drawing can also be re-circulated.

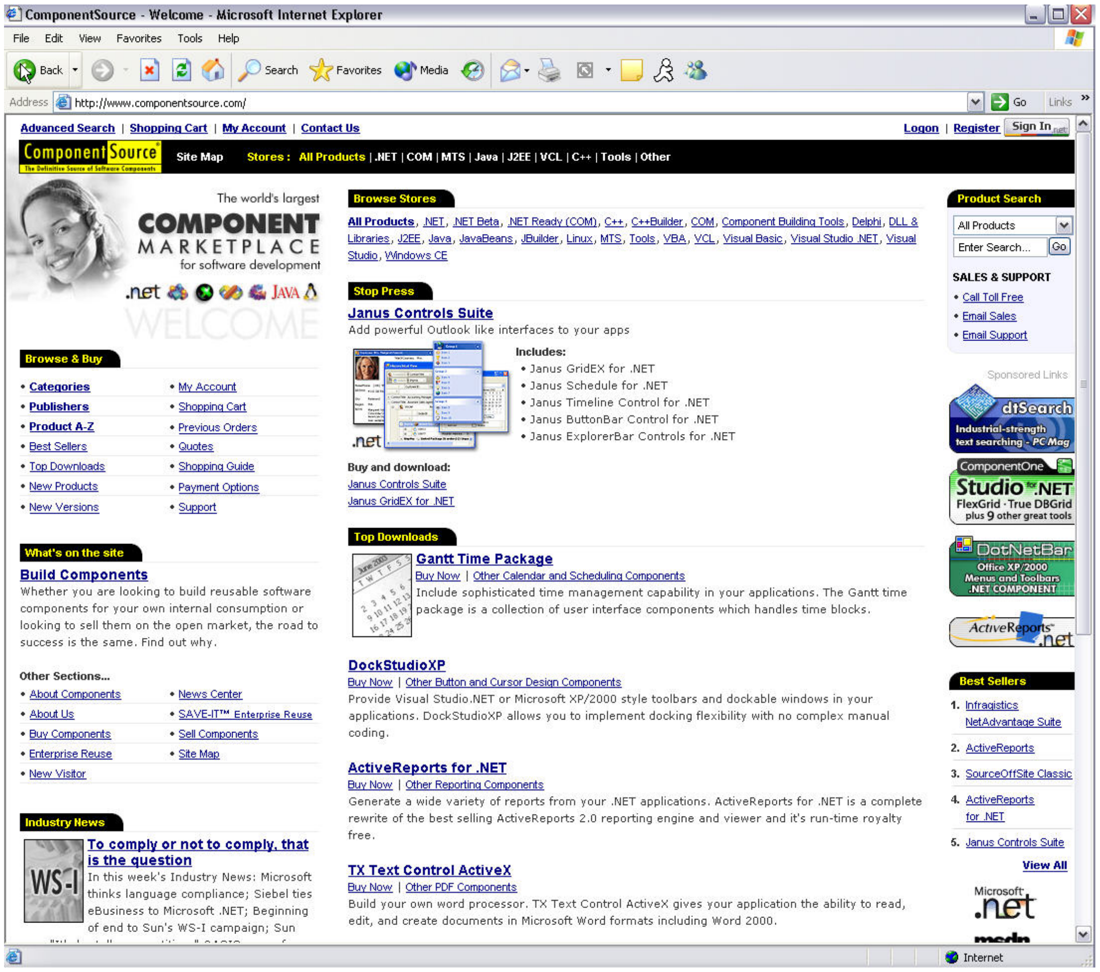
continue

Figure 2: BuildTopia Web-centric Design Collaboration

Sites such as these are most relevant to the problem of software development over the Internet, because they are chasing the same issues, albeit in a different domain. Thus, these CDEs contain many of the same basic elements that a software CDE requires, such as asset storage, basic mechanisms for collaboration, and multiple stakeholder views.

6.2 Asset Management

As Strubing noted, the communication of knowledge is an important task in the daily use case of developers. For software development teams in particular, a key manifestation of that knowledge is found in the project's code and components, which may be shared across releases or even across teams. Using the Web as a repository for such assets is but one feature of a full-blown CDE, but is already commercially available with vendor-neutral component vendors such as ComponentSource [22] (as shown in Figure 3) and Flashline [23] as well as platform-specific sites such as IBM's developerWorks [24] and Microsoft's MSDN [25], both of which deliver component repositories.


ComponentSource - Welcome - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.componentsource.com/
Go Links

Advanced Search | Shopping Cart | My Account | Contact Us | Logon | Register | Sign In

ComponentSource — The Definitive Source of Software Components — Site Map — Stores : All Products | .NET | COM | MTS | Java | J2EE | VCL | C++ | Tools | Other

The world's largest COMPONENT MARKETPLACE for software development

.net JAVA

WELCOME

Browse & Buy

- Categories
- Publishers
- Product A-Z
- Best Sellers
- Top Downloads
- New Products
- New Versions
- My Account
- Shopping Cart
- Previous Orders
- Quotes
- Shopping Guide
- Payment Options
- Support

What's on the site

Build Components
Whether you are looking to build reusable software components for your own internal consumption or looking to sell them on the open market, the road to success is the same. Find out why.

Other Sections...

- About Components
- About Us
- Buy Components
- Enterprise Reuse
- New Visitor
- News Center
- SAVE-IT™ Enterprise Reuse
- Sell Components
- Site Map

Industry News

To comply or not to comply, that is the question
In this week's Industry News: Microsoft thinks language compliance; Siebel ties eBusiness to Microsoft .NET; Beginning of end to Sun's WS-I campaign; Sun

Browse Stores

All Products, .NET, .NET Beta, .NET Ready (COM), C++, C++Builder, COM, Component Building Tools, Delphi, DLL & Libraries, J2EE, Java, JavaBeans, JBuilder, Linux, MTS, Tools, VBA, VCL, Visual Basic, Visual Studio .NET, Visual Studio, Windows CE

Stop Press

Janus Controls Suite
Add powerful Outlook like interfaces to your apps

Includes:
- Janus GridEX for .NET
- Janus Schedule for .NET
- Janus Timeline Control for .NET
- Janus ButtonBar Control for .NET
- Janus ExplorerBar Controls for .NET

Buy and download:
Janus Controls Suite
Janus GridEX for .NET

Top Downloads

Gantt Time Package
Buy Now | Other Calendar and Scheduling Components
Include sophisticated time management capability in your applications. The Gantt time package is a collection of user interface components which handles time blocks.

DockStudioXP
Buy Now | Other Button and Cursor Design Components
Provide Visual Studio.NET or Microsoft XP/2000 style toolbars and dockable windows in your applications. DockStudioXP allows you to implement docking flexibility with no complex manual coding.

ActiveReports for .NET
Buy Now | Other Reporting Components
Generate a wide variety of reports from your .NET applications. ActiveReports for .NET is a complete rewrite of the best selling ActiveReports 2.0 reporting engine and viewer and it's run-time royalty free.

TX Text Control ActiveX
Buy Now | Other PDF Components
Build your own word processor. TX Text Control ActiveX gives your application the ability to read, edit, and create documents in Microsoft Word formats including Word 2000.

Product Search

All Products
Enter Search... Go

SALES & SUPPORT
- Call Toll Free
- Email Sales
- Email Support

Sponsored Links

dtSearch Industrial-strength text searching - PC Mag
ComponentOne Studio for .NET FlexGrid · True DBGrid plus 9 other great tools
DotNetBar Office XP/2000 Menus and Toolbars .NET COMPONENT
ActiveReports .net

Best Sellers

1. Infragistics NetAdvantage Suite
2. ActiveReports
3. SourceOffSite Classic
4. ActiveReports for .NET
5. Janus Controls Suite

View All

Microsoft .net

Internet

Figure 3: ComponentSource Asset Management

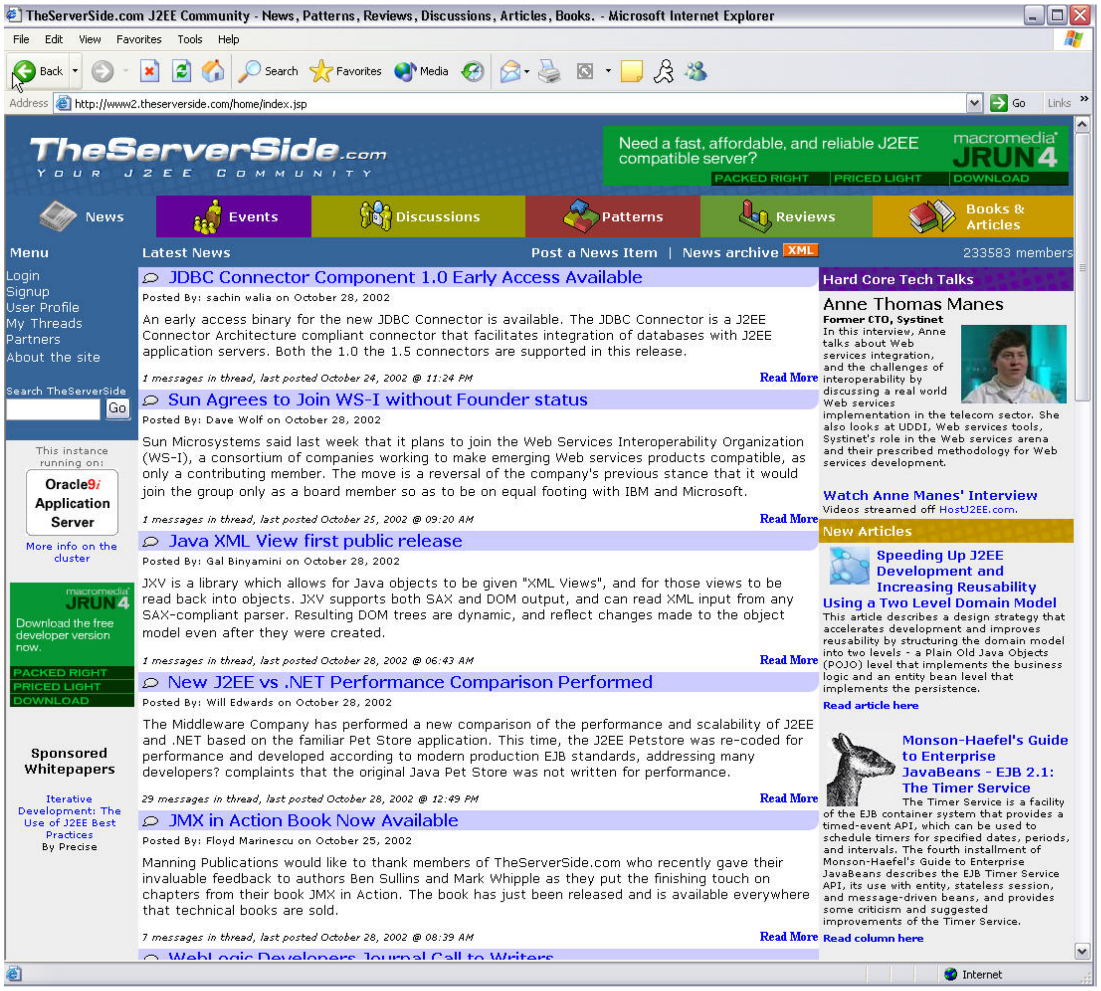
From the perspective of its users, sites such as these essentially push their underlying assets outward via a fairly rigid categorized listing, but with the ability to carry out a general search for specific content. Self-publishing of content is generally not supported directly (because the site owners want to maintain control over the content).

By contrast, variations of the commercial sites, as well as homegrown solutions, are often found deployed within an organization to serve as a general repository of non-public components, parts that are either proprietary or of strategic importance to the company. In these cases, some degree of self-publication is typically possible. However, it should be noted that, without some energy applied to the process, such asset repositories could quickly turn into vast, stinking wastelands of decaying code that no one in the organization wants to touch.

Thus, asset management sites support three primary things: a public or private marketplace for assets (a primary feature of ComponentSource); an infrastructure for asset management (a primary strength of Flashline); and a source for community-contributed assets (a primary feature of MSDN and developerWorks).

In addition to these basic features, it is worth noting that most asset management sites have begun to recognize the need for a deeper user experience surrounding their assets. For example, simple features pioneered (and some patented) by Amazon [26], such as “top 10 lists,” targeted, personalized home pages, user feedback, and discussion groups can contribute to this user experience and the creation of a vibrant community of practice surrounding these assets.

6.3 Information Services


TheServerSide.com J2EE Community - News, Patterns, Reviews, Discussions, Articles, Books. - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www2.theserverside.com/home/index.jsp
Go Links
TheServerSide.com
YOUR J2EE COMMUNITY
Need a fast, affordable, and reliable J2EE compatible server?
macromedia JRUN 4
PACKED RIGHT PRICED LIGHT DOWNLOAD
News Events Discussions Patterns Reviews Books & Articles
Menu
Latest News
Post a News Item | News archive XML
233583 members
Login
Signup
User Profile
My Threads
Partners
About the site
Search TheServerSide
Go
This instance running on:
Oracle9i Application Server
More info on the cluster
macromedia JRUN 4
Download the free developer version now.
PACKED RIGHT
PRICED LIGHT
DOWNLOAD
Sponsored Whitepapers
Iterative Development: The Use of J2EE Best Practices
By Precise
Internet

JDBC Connector Component 1.0 Early Access Available

Posted By: sachin walia on October 28, 2002

An early access binary for the new JDBC Connector is available. The JDBC Connector is a J2EE Connector Architecture compliant connector that facilitates integration of databases with J2EE application servers. Both the 1.0 the 1.5 connectors are supported in this release.

1 messages in thread, last posted October 24, 2002 @ 11:24 PM Read More

Sun Agrees to Join WS-I without Founder status

Posted By: Dave Wolf on October 28, 2002

Sun Microsystems said last week that it plans to join the Web Services Interoperability Organization (WS-I), a consortium of companies working to make emerging Web services products compatible, as only a contributing member. The move is a reversal of the company's previous stance that it would join the group only as a board member so as to be on equal footing with IBM and Microsoft.

1 messages in thread, last posted October 25, 2002 @ 09:20 AM Read More

Java XML View first public release

Posted By: Gal Binyamini on October 28, 2002

JXV is a library which allows for Java objects to be given "XML Views", and for those views to be read back into objects. JXV supports both SAX and DOM output, and can read XML input from any SAX-compliant parser. Resulting DOM trees are dynamic, and reflect changes made to the object model even after they were created.

1 messages in thread, last posted October 28, 2002 @ 06:43 AM Read More

New J2EE vs .NET Performance Comparison Performed

Posted By: Will Edwards on October 28, 2002

The Middleware Company has performed a new comparison of the performance and scalability of J2EE and .NET based on the familiar Pet Store application. This time, the J2EE Petstore was re-coded for performance and developed according to modern production EJB standards, addressing many developers? complaints that the original Java Pet Store was not written for performance.

29 messages in thread, last posted October 28, 2002 @ 12:49 PM Read More

JMX in Action Book Now Available

Posted By: Floyd Marinescu on October 25, 2002

Manning Publications would like to thank members of TheServerSide.com who recently gave their invaluable feedback to authors Ben Sullins and Mark Whipple as they put the finishing touch on chapters from their book JMX in Action. The book has just been released and is available everywhere that technical books are sold.

7 messages in thread, last posted October 28, 2002 @ 08:39 AM Read More

WebLogic Developers Journal Call to Writers

Hard Core Tech Talks

Anne Thomas Manes

Former CTO, Systinet

In this interview, Anne talks about Web services integration, and the challenges of interoperability by discussing a real world Web services implementation in the telecom sector. She also looks at UDDI, Web services tools, Systinet's role in the Web services arena and their prescribed methodology for Web services development.

Watch Anne Manes' Interview

Videos streamed off HostJ2EE.com.

New Articles

Speeding Up J2EE Development and Increasing Reusability Using a Two Level Domain Model

This article describes a design strategy that accelerates development and improves reusability by structuring the domain model into two levels - a Plain Old Java Objects (POJO) level that implements the business logic and an entity bean level that implements the persistence.

Read article here

Monson-Haefel's Guide to Enterprise JavaBeans - EJB 2.1: The Timer Service

The Timer Service is a facility of the EJB container system that provides a timed-event API, which can be used to schedule timers for specified dates, periods, and intervals. The fourth installment of Monson-Haefel's Guide to Enterprise JavaBeans describes the EJB Timer Service API, its use with entity, stateless session, and message-driven beans, and provides some criticism and suggested improvements of the Timer Service.

Read column here

Figure 4: TheServerSide Information Services

Whereas asset management sites focus on the delivery of tangible code and components, information service sites focus on pure intellectual property and as such offer guidance on using particular platforms or best practices in software engineering. Information services typically come in three flavors: public sites which are either vendor-sponsored (and therefore focus on one specific platform) or vendor-neutral, and private sites, which tend to be internal project- or organization-specific. In the public categories, there are sites sponsored by the major platform vendors, including Microsoft's MSDN and IBM's developerWorks, as well as other vendor-neutral (but technology-specific) sites, such as TheServerSide (shown in Figure 4) [27], Web Monkey [28], and the Rational Developer Network (RDN) [29]. For a subscription fee, both the IEEE [30] and the ACM [31] offer digital libraries, which provide access to journal articles and conference proceedings.

6.4 Infrastructure

As we will discuss shortly, a CDE is not so much a single killer app as it is a coherent collection of one hundred small things. Although not really CDEs in their own right, there exist a genre of technologies that provide critical collaborative infrastructures for full-blown CDEs. Of these, instant messaging (IM) is perhaps the most pervasive mechanism for collaboration. In our day in the life experiment, a large percentage of participants reported using IM in their desktop snapshots. Microsoft's NetMeeting [32] is perhaps the most commonly used infrastructure for peer-to-peer video conferencing. For web conferencing that scales, there are products such as WebEx [33] and PlaceWare [34]. Although subtly different in their user experience, both services offer Web-centric conferencing with the ability to broadcast documents and slides (for lectures) and to share desktops (offering the moral equivalent of an electronic whiteboard).

Indeed, there really is a spectrum of infrastructure collaborative mechanisms that may be applied to a Web community, each with its own value. Specifically:

- Mailing lists are good for small groups with a common purpose, conversations that wax and wane over time, communities that are just getting started, and newsletters and announcement.
- Message boards are useful for asking and answering questions, encouraging in-depth conversations, managing high-volume conversations, and providing context, history, and a sense of place.
- Chat rooms are good for holding scheduled events, preparing for and debriefing after life events, discussing offline events in real time, and for hanging out (namely, relaxing, flirting, gossiping, and visiting).
- Whiteboards are useful for brainstorming, communicating, and discussing.
- Net meetings are useful for one-on-one discussions.
- WebEx and PlaceWare meetings are useful for group presentations and distributed discussions.

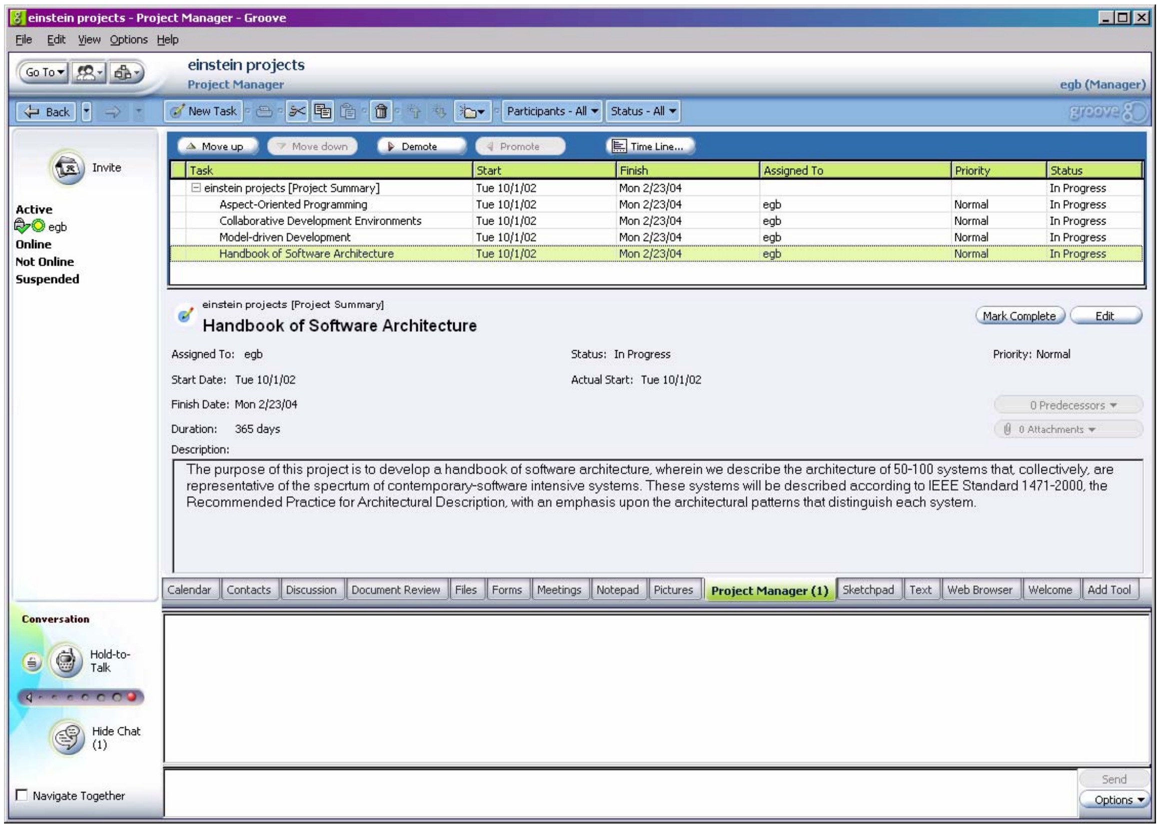
Most of these infrastructure services are quickly becoming commodities, meaning that they are already available from a variety of sources and are ultimately being bundled as a core part of operating systems and main desktop productivity tools, such as Windows XP and Microsoft Office XP.

6.5 Community

Moving closer to complete CDEs are sites that exist to build Web communities. Basically, a Web community is a collection of individuals with a shared interest and a shared identity with that group. A multitude of small things go in to growing and sustaining a vibrant community, but in turn only one or two few small things gone bad can quickly destroy that same community. Usenet newsgroups are perhaps the most elementary Internet communities, but most are generally not satisfying because their signal to noise ratio is extremely low. Lotus Notes [35], developed by Ray Ozzie, has proven effective in building business communities and as such serves as an electronic bulletin board for an organization. Public community services, such as the Ward Cunningham's WikiWeb [36] and Yahoo's groups [37] both offer virtual meeting spaces with the ability to share artifacts and self-administer a community's members and content.

A very vibrant community of a different sort may be found at Slashdot [38]. Although short on tangible artifacts but long on collaborative content, Slashdot is a Web community in a very real sense, for its participants are typically quite passionate and involved contributors to a multitude of threads of discussion. Slashdot is a good example of a jelled community; small things, such as the personal recognition given to individuals, the high rate of change of content, and the information-rich but easily navigated pages all contribute to Slashdot being a genuinely fun place to hang out.

Groove [39] is Ray Ozzie's take on building scalable communities by using peer-to-peer technology. Groove is a for-fee Web-centric service, and is perhaps the best example of a general, domain-independent collaborative site. As Figure 5 illustrates, the Groove user experience provides a virtual project space with the ability to share assets, conduct discussions, and manage tasks and schedules. Groove features include desktop integration with tools such as Microsoft Office, an open framework for integrating other tools, text and voice chat, and dynamic awareness of users and content.



Task	Start	Finish	Assigned To	Priority	Status
einstein projects [Project Summary]	Tue 10/1/02	Mon 2/23/04			In Progress
Aspect-Oriented Programming	Tue 10/1/02	Mon 2/23/04	egb	Normal	In Progress
Collaborative Development Environments	Tue 10/1/02	Mon 2/23/04	egb	Normal	In Progress
Model-driven Development	Tue 10/1/02	Mon 2/23/04	egb	Normal	In Progress
Handbook of Software Architecture	Tue 10/1/02	Mon 2/23/04	egb	Normal	In Progress

einstein projects [Project Summary]
Handbook of Software Architecture

Assigned To: egb
Status: In Progress
Priority: Normal
Start Date: Tue 10/1/02
Actual Start: Tue 10/1/02
Finish Date: Mon 2/23/04
Duration: 365 days
Description:
The purpose of this project is to develop a handbook of software architecture, wherein we describe the architecture of 50-100 systems that, collectively, are representative of the spectrum of contemporary-software intensive systems. These systems will be described according to IEEE Standard 1471-2000, the Recommended Practice for Architectural Description, with an emphasis upon the architectural patterns that distinguish each system.

Figure 5: A Groove Workspace

Similar to Groove but using a very different underlying technology is Microsoft's SharePoint [40]. SharePoint comes in two flavors: SharePoint Team services (for ad hoc, small teams) and SharePoint Portal (for enterprise intranets). In Microsoft's words:

- Small or ad hoc workgroups need informal means to work together on group deliverables, share documents, and communicate status with one another. These groups need to share information easily and effortlessly and SharePoint Team Services–based Web sites allow them to do that.
- Large workgroups with structured processes need greater management over their information. They require features like formal publishing processes and the ability to search for and aggregate content from multiple data stores and file formats. For this scenario, SharePoint Portal Server 2001 is recommended.

6.6 Software Development

Currently, there exist a handful of research CDEs directed to the problem of software development, and an even smaller number of commercial sites.

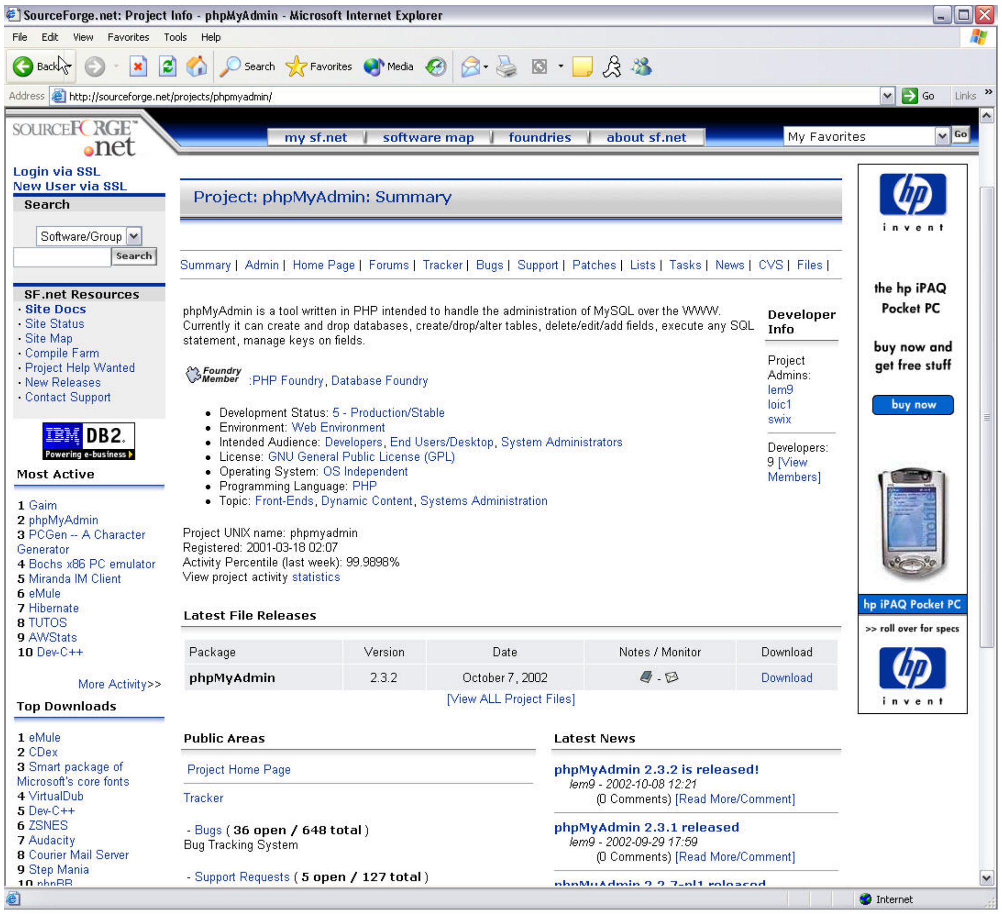
Perhaps the most interesting research CDE is MILOS (Minimally Invasive Long-term Organizational Support), an effort being carried out by the Software Process Support Group at the University of Calgary and the Artificial Intelligence Group at the University of Kaiserslautern [41]. MILOS is an open source effort (under a GNU public license) and focuses primarily on software process workflow automation.

Commercially, DevX [42] and collab.net's [43] products fit the category of software CDEs, with DevX offering only minimal features and collab.net offering many more creature comforts. DevX is more so an information services site, although it does offer some basic tools for bug tracking.

Collab.net has both a public and a private face. Its public face is SourceForge [44], an open source CDE focused on, not surprisingly, the development of open source software. SourceForge is host to projects ranging from the generally useful MySQL to the more sinister BackOrfice 2000. Its private face is SourceCast [45], in effect a private label SourceForge (but with a number of other important features, such as greater security, something that is largely a non-issue for open source development).

As Figure 6 illustrates, the SourceForge user experience is both project- and artifact-centric. A user may enter the site via a specific project or via a personalized home page. Within a project, there are facilities for artifact storage, simple configuration management (via the open source CVS), simple bug tracking (also via an open source tool), task management, and discussions. SourceForge offers a number of creature comforts, such as the ability o self-publish, track changes, and manage project membership.

We cannot overemphasize the contribution made by the open source community to the creation of CDEs, recognizing that this is an emergent benefit from the open source movement. Beyond the creation of some genuinely useful software (i.e. Linux and Apache), the community has demonstrated that complex software of scale can be created by large, dispersed teams over the Internet using an interactive, fast-turnaround lifecycle, but with a core team and collaborative mechanisms that permit continuous integration and the creation of stable, intermediate releases.


SourceForge.net: Project Info - phpMyAdmin - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://sourceforge.net/projects/phpmyadmin/
Go Links
SOURCEFORGE .net
my sf.net | software map | foundries | about sf.net
My Favorites Go
Login via SSL
New User via SSL
Search
Software/Group
Search
hp invent
the hp iPAQ Pocket PC
buy now and get free stuff
buy now
hp iPAQ Pocket PC
>> roll over for specs
IBM DB2. Powering e-business
Internet

Project: phpMyAdmin: Summary

Summary | Admin | Home Page | Forums | Tracker | Bugs | Support | Patches | Lists | Tasks | News | CVS | Files |

SF.net Resources
- Site Docs
- Site Status
- Site Map
- Compile Farm
- Project Help Wanted
- New Releases
- Contact Support

phpMyAdmin is a tool written in PHP intended to handle the administration of MySQL over the WWW. Currently it can create and drop databases, create/drop/alter tables, delete/edit/add fields, execute any SQL statement, manage keys on fields.

Foundry Member :PHP Foundry, Database Foundry

- Development Status: 5 - Production/Stable
- Environment: Web Environment
- Intended Audience: Developers, End Users/Desktop, System Administrators
- License: GNU General Public License (GPL)
- Operating System: OS Independent
- Programming Language: PHP
- Topic: Front-Ends, Dynamic Content, Systems Administration

Project UNIX name: phpmyadmin
Registered: 2001-03-18 02:07
Activity Percentile (last week): 99.9898%
View project activity statistics

Developer Info

Project Admins:
lem9
loic1
swix

Developers:
9 [View Members]

Most Active

1 Gaim
2 phpMyAdmin
3 PCGen -- A Character Generator
4 Bochs x86 PC emulator
5 Miranda IM Client
6 eMule
7 Hibernate
8 TUTOS
9 AWStats
10 Dev-C++

More Activity>>

Top Downloads

1 eMule
2 CDex
3 Smart package of Microsoft's core fonts
4 VirtualDub
5 Dev-C++
6 ZSNES
7 Audacity
8 Courier Mail Server
9 Step Mania
10 phpBB

Latest File Releases

Package	Version	Date	Notes / Monitor	Download
phpMyAdmin	2.3.2	October 7, 2002	-	Download

[View ALL Project Files]

Public Areas

Project Home Page

Tracker

- Bugs (**36 open / 648 total**)
Bug Tracking System

- Support Requests (**5 open / 127 total**)

Latest News

phpMyAdmin 2.3.2 is released!
lem9 - 2002-10-08 12:21
(0 Comments) [Read More/Comment]

phpMyAdmin 2.3.1 released
lem9 - 2002-09-29 17:59
(0 Comments) [Read More/Comment]

phpMyAdmin 2.2.7-pl1 released

Figure 6: SourceForge CDE

7. Collaborative Development Environment Features

Our survey suggests that there does not yet exist the quintessential software-specific CDE although the prognosis for such an entity is promising, since this broad spectrum of sites and technologies offer the underlying features necessary for such a CDE. After looking at history, performing our own study of developer practices, and examining the key technical trends in the current state of the practice with collaborative solutions, we are able to synthesize the primary features of a software CDE.

The value of a CDE comes primarily in addressing the points of friction in software development. We observe that this manifests itself in features that support keeping track of project information and resources, manage tools and artifacts responsible for generating those resources, and enable communication among teams.

More specifically, Roger Fournier suggests that the following elements are essential in the creation of a virtual meeting space [46]:

- Instant messaging
- Virtual meeting room

- Application sharing
- Centralized information management
- Searching and indexing
- Configuration control of shared artifacts
- Co-browsing
- Electronic document routing and workflow
- Calendaring and scheduling
- Online event notification
- Project resource profiling
- Whiteboards
- Online voting and polling

We would add to this list the following features:

- Tools for both connected and disconnected use
- Threaded discussion forum
- Project dashboards and metrics
- Self-publication of content
- Self-administration of projects
- Multiple levels of information visibility
- Personalization of content
- Virtual agents that can be scripted to manage daily project hygiene and other repetitive tasks [47]

As illustrated in Figure 7, we can organize these features into three categories of capabilities necessary for any CDE. Informally known as the "Three Cs", these categories are based on the coordination, collaboration, and community building nature of a CDE.

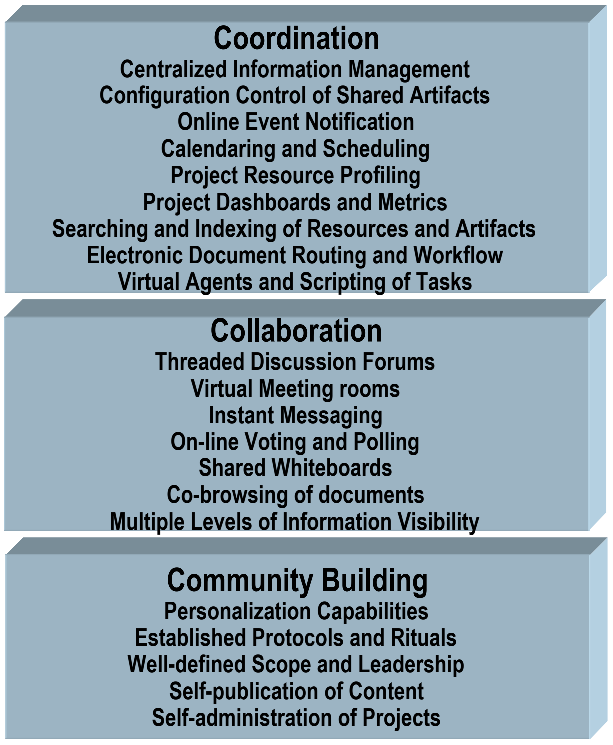

Coordination
Centralized Information Management
Configuration Control of Shared Artifacts
Online Event Notification
Calendaring and Scheduling
Project Resource Profiling
Project Dashboards and Metrics
Searching and Indexing of Resources and Artifacts
Electronic Document Routing and Workflow
Virtual Agents and Scripting of Tasks
Collaboration
Threaded Discussion Forums
Virtual Meeting rooms
Instant Messaging
On-line Voting and Polling
Shared Whiteboards
Co-browsing of documents
Multiple Levels of Information Visibility
Community Building
Personalization Capabilities
Established Protocols and Rituals
Well-defined Scope and Leadership
Self-publication of Content
Self-administration of Projects

Figure 7. A Categorization of CDE Features.

None of these features are, by themselves, particularly complex or difficult to implement [48]. For this reason, we observe that a rich CDE is the emergent creature that rises from a hundred small things, using the Web as the center of the user experience. Collectively, however, this set of capabilities offers significant challenges in the technical integration of these features, and the creation of a satisfying user experience for teams and individuals routinely employing many of these features to complete a shared task. Much of the diversity seen the examples referenced in this paper are a result of different approaches taken toward addressing these challenges.

Conceptually, we may render a software development-specific CDE as shown in Figure 8 [49]. Orbiting the CDE are team members who may or may not be geographically dispersed. The CDE itself is bounded by a secure physical and software boundary that guards the sanctity of the CDE's content and validates users (who likely have different rights that yield different views and permissible operations upon a project's artifacts). The CDE is further decomposed into three layers:

- Project workspaces, which provide mechanisms for team management and collaboration
- Team tools, which provide infrastructure support for requirements management, change management/version control, and document management
- Development resources, which provide artifact-specific tools, Web-centric training, asset management, and information services

There are some important variations upon this model that must be considered for the general case.

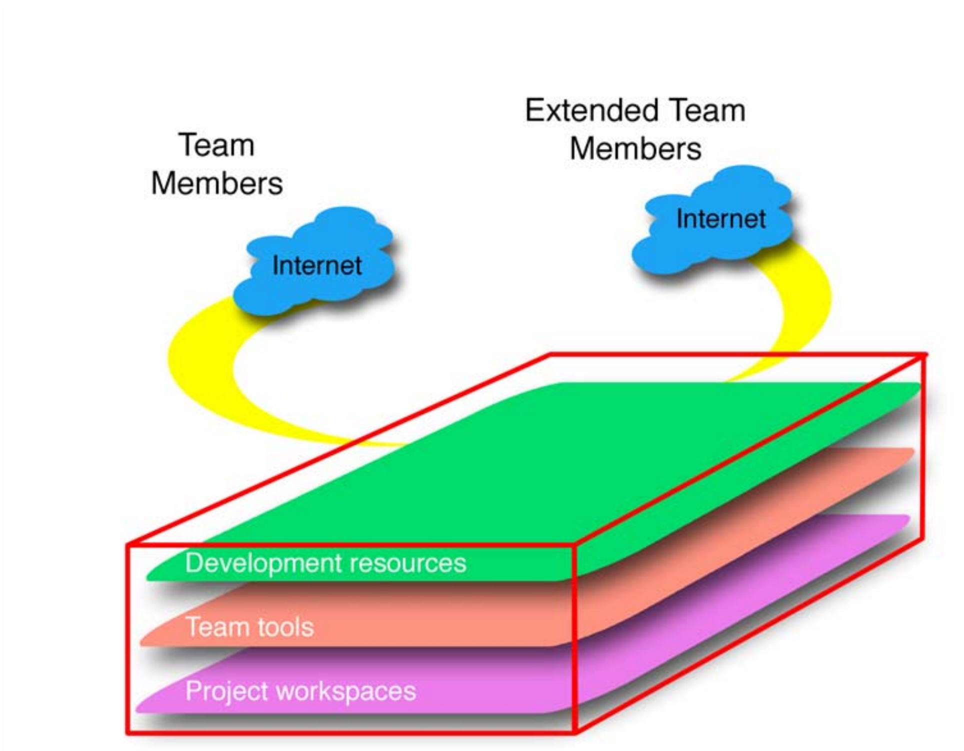


Figure 8: Conceptual Model of a Collaborative Development Environment

First, are tools hosted or non-hosted? If fully hosted, this leads to an ASP (application service provider) model with all of its benefits (ease of tool distribution and administration) as well as its challenges (delivering a meaningful user experience for highly interactive tools such as visual modeling tools). If fully not hosted, the user experience is somewhat complicated because each user's desktop must host the appropriate tools. In practice, our experience is that a mix of hosting and non-hosting is best: host core infrastructure tools, such as change management, that must be used by every stakeholder, and support desktop integration to other, more semantically rich tools.

Second, does the CDE reside in front of or behind an organization's firewall? If the CDE lives behind the organization's firewall and is therefore a captive resource, physical and soft security issues are somewhat simplified, or rather, delegated to the company's existing information technology security policies and mechanisms. If the CDE lives on the open Web, in front of a company's firewall, security issues are more complicated, although little more so than required by any public enterprise site. Indeed, once such security mechanisms are in place, the barriers are mainly emotional, not real.

Third, does the CDE support disconnected use? If its stakeholders are all directly connected to the CDE when working with its content, the management of project state is simplified. However, in practice, developers engage in project work while disconnected: long airplane flights are still a fine place to code, as is a home office. In such cases, the

CDE must take into account the synchronization of changes and the cloning of a project's state. Our experience is that the presence of a solid change management system in the CDE's infrastructure can generally deal with the disconnected use case fairly well.

8. The Evolution of Collaborative Development Environments

Given this survey of the current state of the practice in collaborative solutions on the Web, and the complete set of CDE features that may be provided in any solution, we recognize that there are a number of substantial barriers to successful adoption of a CDE. These include technological, emotional, and business barriers.

First, there remain complex technical barriers. Building a highly secure, scalable, high-performance Web-centric system, no matter the problem domain, is still a challenging engineering problem. In fact, many question the whole basis of using a public (and inherently insecure), uncontrolled infrastructure such as the Internet as the basis for many kinds of collaborative tasks involving key pieces of a corporations intellectual property. Second, there are emotional barriers: leaving a project's key assets on servers out of the immediate control of the team is potentially frightening. Additionally, communities are fragile things, easily ruined by small, meddlesome features; as such, the importance of a simple, friendly, and complete user experience cannot be discounted. Furthermore, the presence of a CDE encourages the sharing of information, and in some larger organizations, that's viewed as a political threat. Third, there are business issues surrounding the nature of a successful business model for a CDE. This includes fundamental issues such as subscription versus pay-per-use versus single fee charging, charging for tools versus charging for services versus charging for information, and loosely-coupled best-of-breed solutions versus tightly-coupled custom developed solutions.

As for the technical issues, the good news is that time is on our side, for the general trend is that mechanisms and best practices for Web-centric development are maturing. Emotional barriers are harder to overcome, although it should be pointed out that, for most modest to large organizations, project assets are already outside the direct control of each team (and instead in the hands of the company's IT department). As for business barriers, we expect that the market place will, over time, center upon the right business models for CDEs.

9. Summary

Effective teamwork is an essential part of every non-trivial software engineering effort. Collaborative capabilities are essential to support these teams, particularly as team sizes get smaller, and team interaction becomes more geographically dispersed. This paper has examined the current features required for a fully functioning CDE by surveying the current state of the practice as represented by a range of different solutions. The paper then synthesized a complete feature list from these examples, and categorized those features into three main types: coordination, collaboration, and community building mechanisms.

However, as we have already stated, no CDE currently supports all of the features we have discussed. Furthermore, few organizations find themselves in a position to readily adopt such a CDE should it be available. Thus, having read this paper many individuals and project managers are left asking what positive steps they can now take to prepare their teams for effective adoption of CDE technology. Simply stated, we believe there are 5 key steps to prepare an organization for successful CDE adoption:

1. Get some good team processes in place for coordination, collaboration, and community building within and across your organization. Document those processes, and understand some of the key inefficiencies, or points of friction, that currently exist.
    - To assist with this you can learn from, and adopt, existing process frameworks such as the Rational Unified Process (RUP), and agree on communication via standard notations for describing artifacts and processes such as the Unified Modeling Language (UML).

2. If you have existing team infrastructure technology, evaluate it against the features described in this paper. If you don't already have that technology, examine what is available and invest in some key areas.
    - A number of products are available offering many essential infrastructure features, e.g., requirements management, visual modeling, test management.

3. Standardize on a common IP approach across the teams and projects.
    - This may involve use of information portals offered by tool vendors, together with locally maintained project specific information repositories.

4. Examine current reuse practices within and across the teams. Look for opportunities to increase collaboration and sharing of IP, and provide practical guidance that facilitates this.
    - Common standards for documentation and discovery of project assets may be helpful. Examine work such as the Reusable Asset Specification (RAS) [50] for ideas on how this can practically be achieved.

5. Monitor, optimize, and iterate these 5 steps.

To conclude, we observe that CDEs may be classified in one of several stages of maturity, each of which builds upon the previous stage:

- Stage 1: Support for simple artifact storage
- Stage 2: Availability of basic mechanisms for collaboration
- Stage 3: Infrastructure tools for advanced artifact management
- Stage 4: Creature comforts for team management, advanced mechanisms for collaboration, and availability of artifact-specific tools

- Stage 5: Asset and information services that encourage a vibrant community of practice

With regard to the state of the practice in CDEs, the current median is hovering somewhere around Stage 1 and 2, but with point solutions, as our survey indicates, for elements at advanced stages.

Indeed, the proliferation of these point solutions suggests that the prognosis for the future of CDEs is encouraging, since vibrant CDEs are primarily the sound collection of these hundred things. Even if a project's understanding of a problem were perfect, even if things always worked, even if all stakeholders were in agreement, a software development project has inherent friction. As we have examined, the core value of a vibrant CDE is to provide a frictionless surface for the development team, thereby minimizing the many daily irritants of collaboration and letting the team focus on its primary task, namely, the creation of useful software that works.

10. Notes and References

1. For example, it is common to have an architecture team that builds mechanisms upon commercial middleware such as Microsoft's .NET or IBM's WebSphere. Creating such mechanisms typically requires special knowledge, which is more efficiently managed by a centralized team than by every individual developer most of whom are domain experts, not technology experts.
2. Weinberg, G., 1989, *The Psychology of Computer Programming*, Dorset House Publishing.
3. DeMarco, T. and Lister, T. 1999, *Peopleware: Productive Projects and Teams*, Dorset House Publishing.
4. Constantine, L., 2001, *Peopleware Papers: Notes on the Human Side of Computing*, Prentice Hall.
5. Votta, L., 1994, *By the Way, Has Anyone Studied Real Programmers Yet?* 9th International Software Process Workshop, Reston, Virginia.
6. Strubing, J., 1994, "Designing the Working Process: What Programmers Do Besides Programming," *User-Centered Requirements for Software Engineering Environments*, Springer, Berlin, Germany.
7. These numbers add up to more than 100%, because several participants worked on multiple platforms.
8. This is a subtle but psychologically important shift: rather than saying "I shall open this editor so that I can cut some code" one would say "here's an aspect of code that I need to work on (and hand me the right tools to do the job)."
9. In particular, the Well is perhaps the quintessential collaborative site.
10. In particular, see Kim, A. J., 2000, *Community Building on the Web: Secret Strategies for Successful Online Communities*, Peachpit Press.
11. Goldberg, A., 2000, "Collaborative Software Engineering," Net Object Days, Erfurt, Germany.
12. www.media.mit.edu
13. www.asktog.com/starfire/starfireHome.html
14. Ante, S., "Simultaneous Software," BusinessWeek, August 27, 2001.
15. Yourdon, E., 1997, *Death March*, Prentice-Hall.

16. As the classic joke (from www.cartoonbank.com) goes, "On the Internet, no one knows you are a dog." In a virtual project space, whether or not a team member is human is sometimes irrelevant.

17. Of course, sometimes that's not so easy an edict to follow, especially if the market place offers limited choices.

18. We say *potential* simply because there have been no hard economic studies for the ROI of CDEs in practice. However, the business value of a CDE comes from the observation that this technology can be both an aspirin as well as a vitamin: it alleviates some of the pain of development and promotes healthy project behavior.

19. www.buildtopia.com

20. www.cocreate.com

21. www.cadence.com

22. www.componentsource.com

23. www.flashline.com

24. www.ibm.com/developerworks

25. msdn.microsoft.com

26. www.amazon.com

27. www.theserverside.com

28. www.hotwired.lycos.com/webmonkey

29. www.Rational.net

30. www.computer.org/publications/dlib

31. www.acm.org/dl

32. www.microsoft.com/windows/NetMeeting

33. www.webex.com

34. www.placeware.com

35. www.lotus.com

36. www.wikiweb.com

37. groups.yahoo.com

38. www.slashdot.com

39. www.groove.com

40. www.microsoft.com/sharepoint

41. Holtz, H., Konnecker, A., Maurer, F., "Task-Specific Knowledge Management in a Process Centred SEE", Proceedings of the Workshop on Learning Software Organizations, LSO-2001, Springer 2001.

42. www.devx.com

43. www.collab.net

44. sourceforge.net

45. www.collab.net/products/sourcecast

46. Fournier, R., *Infoworld*, March 5, 2001

47. It's not a great leap to imagine anthropomorphized agents such as found at www.annanova.com serving as extended team members. Such virtual agents could be tasked with regular tedious

administrative activities, such as providing meeting reminders, driving common workflows such as a document review cycle, or nagging developers to assemble components for daily releases.

48. Except for the hosting of tools and the provision of strong security, both of which are inherently difficult issues.

49. Adapted from a model first drawn by Dan Wedge and Rich Hillebrecht of Rational Software Corporation.

50. www.rational.com/ras