Collaborative Development Environments

Grady Booch
IBM Rational
December 1, 2006

A **collaborative development environment** (CDE)\(^1\) is a virtual space where the stakeholders of a project – even if separated by time or space – can meet, share, brainstorm, discuss, reason about, negotiate, record, and generally labor together to carry out some task, most often to create some useful artifact and its supporting objects.

Collaboration is an element of every engineering domain. As described by Coplien in *Organizational Patterns of Agile Software Development*, there are some well-defined and well-understood patterns of behavior in hyperproductive teams. There exist opportunities to encourage and amplify these patterns through the use of a CDE. The rise of the Web as a natural extension of an individual’s physical and daily world, the economic pull of outsourcing and off-shore development, the integration of third-party software and in particular the emergence of Web-centric service-oriented architectures, the use of home and remote offices, and the growth of strategic partnerships among organizations and companies have all contributed to the increasing distribution of these teams.

Collaboration has always been an essential part of the fabric of the Internet: email, instant messaging, chat rooms, discussion groups, and Wikis are common collaborative elements that have matured over time, and so in one regard, there is nothing new or novel here. Collaboration among teams is already facilitated through the use of an increasing number of features embedded in standard desktop products such as office suites, where there is ample support for shared document reviews, distribution of documents among teams, and mechanisms for performing common collaborative tasks. In a very practical sense these tools provide the baseline of collaboration functionality for today’s software developers.

However, there are two elements that make CDEs for software development materially different: first, software developers must manipulate semantically-deep artifacts with equally semantically-deep associations among them (this is in contrast to the kinds of semantically-weak textual documents commonly manipulated by business organization); second, the Web is essentially the atmosphere in which virtually all software developers live, and thus it is a very short distance from physically co-located teams to virtually co-located ones, leveraging off the plumbing of the Web and populating this atmosphere with creature comforts that facilitate distributed collaboration.

A CDE is not a very unique or special thing: there are a hundred small things that already exist and that can be combined to form a virtual space. However, a CDE is a particularly fragile thing: because a CDE touches on the social elements of development, it is very sensitive to issues of presentation, simplicity, ease of use, personalization, and culture. A CDE must be lightweight, nimble, and snappy in appearance yet deep in its inner workings; a CDE must never, ever get in the way of work, but rather must disappear in the ether, providing a sanctuary in which the individual

---

\(^1\) CDE is a term that Alan Brown and I first coined in our article with the same title in Zelkowitz’s *Advances in Computers*, published in 2002.
collaborative development environments

The purpose of a CDE is thus to create a frictionless surface for development by eliminating or automating many of the daily, non-creative activities of the individual and the team and by providing mechanisms that encourage creative, healthy, and high-bandwidth modes of communication among a project’s stakeholders.

Every engineering activity involves producing a solution that balances the tangible as well as intangible forces that weigh upon a system. In software, these forces include the usual business ones (cost, schedule, mission), environmental (resources, compatibility, and complexity), developmental (production and resilience), operational (functionality, performance, quality, and dependability) and even issues of value (involving legal, ethical, and moral issues).

For systems of any reasonable complexity, no one person can efficiently counteract these forces. Thus, for most software-intensive systems under construction, operation, or revision, software development is a team sport. In these circumstances inter- and intra-team interaction, communication, and dynamics play at 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 few 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.

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 an even 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. 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 developers.

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.

To paraphrase Abraham Lincoln, software development takes place of the Web, by the Web, and for the Web. There is considerable continuing 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.

The most common IDEs (namely, Microsoft’s Visual Studio and the open source Eclipse) are thick-client centric: catering to the code warrior, these tools serve up a single, complex window whereby the programmer peers into the system under construction. We suggest that 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. Finally, a CDE should be multidimensional in the sense that different stakeholders should be offered different views (some via browsers, some via thick clients), each adapted to that stakeholder’s specific needs.

Collaborative sites both on and off the Web have existed for some time, but we began to see collaborative sites focused solely on software development starting about ten 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 being built by Boeing (and as discussed later, it should come as no surprise that Boeing has continued to innovate mightily in this space). 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.
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.

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. 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. 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).

There exist only a few commercial CDEs focused primarily on the problem of software development over the Web (most notably, SourceForge and Collab.net), but there are many more that have been created 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.

The construction, manufacturing, and electronics industries have been a fruitful place for the evolution of collaboration products. 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. This domain is so well-suited to collaboration products that there even exists a trade organization (Network for Construction Collaboration Technology Providers, http://www.ncctp.net/index.html) who is forming standards for this domain.
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 Gehry Technology’s Digital Project tools. Frank Gehry (http://en.wikipedia.org/wiki/Frank_Gehry) is the architect of dramatic works such as the Guggenheim Museum in Spain. He formed a company (Gehery Technologies) to commercialize the tools that he used to design and construct such buildings. Their Digital Project tools include Designer, Foundation, Structures, MEP (for modeling), Viewer and Project Manager (for project management), and Knowledge Template and Knowledge Adviser (for model checking). The following figure provides an image from their Project Manager.

Moving to the domain of manufacturing, Boeing, together with Dassault, created the Global Collaboration Environment (GCE) to design and build the Boeing 787. This Web-centric system provides modeling, asset management, and product management tools, with an emphasis on traceability and auditability. Dassault has its own specific collaboration tools, most notably ENOVIA (http://www.3ds.com) for collaboration across the manufacturing chain. Their SMARTTEAM Web editor, pictured below, offers tools for the distributed manipulation of 3D models. Dassualt has even created a robust community of practice (http://www.coe.org) for their collaboration products.

As we have already alluded, a CDE is not so much a single killer app as it is a coherent collection of a 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) followed by email are perhaps the most pervasive mechanisms for collaboration. Products such as Microsoft’s NetMeeting are also commonly used by teams for ad-hoc conferencing (with WebEx and its equivalents providing more scalable facilities). 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).

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

- Blogs are useful (as mostly one way) informational sites.
- 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, as well as for group presentations and distributed discussions.
• Portals (such as developerWorks) are somewhat useful meeting places for communities of practice; not only can the primary owner of the domain serve up relevant knowledge, but there are ways that members of that community can share their experiences and solutions.
• Wikis are rather like message boards on steroids, permitting a trusted community to figuratively meet at a water cooler and leave behind the bread crumbs of their conversations.

Most of these infrastructure services are quickly becoming commodities, meaning that they are already available from a variety of sources (and in particular, open-source).

In conclusion, effective teamwork is an essential part of every non-trivial software engineering effort. Collaborative capabilities are essential to support these teams, particularly as individual team sizes get smaller while team interaction becomes more geographically dispersed.