Collaboration in Software Engineering: A Roadmap

Jim Whitehead
Univ. of California, Santa Cruz, USA
ejw@cs.ucsc.edu

Abstract

Software engineering projects are inherently cooperative, requiring many software engineers to coordinate their efforts to produce a large software system. Integral to this effort is developing shared understanding surrounding multiple artifacts, each artifact embodying its own model, over the entire development process. This focus on model-oriented collaboration embedded within a larger process is what distinguishes collaboration research in software engineering from broader collaboration research, which tends to address artifact-neutral coordination technologies and toolkits. This article first presents a list of goals for software engineering collaboration, then surveys existing collaboration support tools in software engineering. The survey covers both tools that focus on a single artifact or stage in the development process (requirements support tools, UML collaboration tools), and tools that support the representation and execution of an entire software process. Important collaboration standards are also described.

Several possible future directions for collaboration in software engineering are presented, including tight integration between web and desktop development environments, broader participation by customers and end users in the entire development process, capturing argumentation surrounding design rationale, and use of massively multiplayer online (MMO) game technology as a collaboration medium. The article concludes by noting a problem in performing research on collaborative systems, that of assessing how well certain artifacts, models, and embedded processes work, and whether they are better than other approaches.

1. Introduction

As humans, we have several limitations that affect our ability to create almost any piece of software. When working at high levels of abstraction—as when writing requirements, designing software, writing code, or creating test cases—we are slow and error-prone. As a consequence, we must work together to complete large projects in reasonable time, and have other people try to catch our mistakes. Once we start working together, we face other problems. The natural language we use to communicate is wonderfully expressive, but frequently ambiguous. Our human memory is good, but not quite deep and precise enough to remember a project's myriad details. We are unable to track what everyone is doing in a large group, and so risk duplicating or clobbering the work of others. Large systems can often be realized in multiple ways, and hence engineers must converge on a single architecture and design.

Collaboration techniques in software engineering have evolved to address our limitations. Software engineering collaboration has multiple goals spanning the entire lifecycle of development:

- *Establish the scope and capabilities of a project.* Engineers must work with the users and funding sources (stakeholders) of a software project to describe what it should do at both a high level, and at the level of detailed requirements. The form of this collaboration can have profound impact on a project, ranging from the up-front negotiation of the waterfall model, to the iterative style of evolutionary prototyping [1].
- *Drive convergence towards a final architecture and design.* System architects and designers must negotiate, create alliances, and engage domain experts to ensure convergence on a single system architecture and design [2].
- *Manage dependencies among activities, artifacts, and organizations.* [3]. This encompasses a wide range of collaborative activities, including typical management tasks of subdividing work into tasks, ordering them, then monitoring, assessing, and controlling the plan of activities. Modularization decisions also affect dependencies.
- *Reduce dependencies among engineers.* An important mechanism for managing dependencies is to reduce them where possible, thereby reducing the need for collaboration. Modularization decisions frequently follow organizational boundaries [4], a mechanism for reducing cross-organization coordination. Software configuration management systems permit developers to work in per-developer workspaces, thereby isolating their changes from others, and reducing the number of change dependencies among developers. With workspaces, developers no longer need to wait for all

developers to finish their current changes before compiling.

- *Identify, record and resolve errors.* Errors and ambiguities are possible in all software artifacts, and many approaches have been developed to find and record their existence. Among the collaborative techniques are inspections and reviews, where many people are brought together so that their multiple perspectives can identify errors, and their questions can surface ambiguities. Testing, where one group creates tests to uncover errors in software developed by others is another collaborative error finding technique. Users of software also collaborate in the identification of errors, whether in explicit beta testing programs, or through normal use, when they submit bug reports. Bug tracking (issue management) systems permit engineers to record problems, as well as manage the process of resolving them.
- *Record organizational memory.* In any long running collaborative project, people may join and leave. Part of the work of collaboration is recording what people know, so that project participants can learn this knowledge now, and in the future [5]. SCM change logs are one form of organizational memory in software projects, as are project repositories of documentation. Process models also record organizational memory, describing best practices for how to develop software.

Software engineers have adopted a wide range of communication and collaboration technologies to assist in the coordination of project work. Every mainstream communication technology has been adopted by software engineers for project use, including telephone, teleconferences, email, voice mail, discussion lists, the Web, instant messaging, voice over IP, and videoconferences. These communication paths are useful at every stage in a project's lifecycle, and support a wide range of unstructured natural language communication. Additionally, software engineers hold meetings in meeting rooms, and conduct informal conversations in hallways, doorways, and offices. While these discussions concern the development of a formal system, a piece of software, the conversations themselves are not formally structured (exceptions being automated email messages generated by SCM systems and bug tracking systems).

In contrast to the unstructured nature of conversations, much collaboration in software engineering is relative to various formal and semi-formal artifacts. Software engineers collaborate on requirements specifications, architecture diagrams, UML diagrams, source code, and bug reports. Each is a different model of the ongoing project. Software engineering collaboration can thus be understood as artifact-based, or model-based collaboration, where the focus of activity is on the production of new models, the creation of shared meaning around the models, and elimination of error and ambiguity within the models. The broad extent of this model-based collaboration is a hallmark of software engineering collaboration. It distinguishes the study of collaboration within software engineering from the more general study of collaboration, which tends to lack this focus on model creation.

This model orientation to software engineering collaboration is important due to its structuring effect. The models provide a shared meaning that engineers use when coordinating their work, as when engineers working together consult a requirements specification to determine how to design a portion of the system. Engineers also use the models to create new shared meaning, as when engineers discuss a UML diagram, and thereby better understand its meaning and implications for ongoing work. The models also surface ambiguity by making it possible for one engineer to clearly describe their understanding of the system; when this is confusing or unclear to others, ambiguity is present. Without the structure and semantics provided by the model, it would be more difficult to recognize differences in understanding among collaborators.

Software engineers have developed a wide range of model-oriented technologies to support collaborative work on their projects. These technologies span the entire lifecycle, including collaborative requirements tools [6, 7], collaborative UML diagram creation, software configuration management systems and bug tracking systems [8]. Process modeling and enactment systems have been created to help manage the entire lifecycle, supporting managers and developers in assignment of work, monitoring current progress, and improving processes [9, 10]. In the commercial sphere, there are many examples of project management software, including Microsoft Project [11] and Rational Method Composer [12]. Several efforts have created standard interfaces or repositories for software project artifacts, including WebDAV/DeltaV [13, 14] and PCTE [15]. Web-based integrated development environments serve to integrate a range of model-based (SCM, bug tracking systems) and unstructured (discussion list, web pages) collaboration technologies.

The remainder of the paper provides an overview of existing model-based collaboration techniques (Section 2). It then outlines several potential areas for improving the state of collaboration support technologies for software projects (Section 3), and notes the challenges in assessing the impact of collaboration tools (Section 3.5). The paper concludes with a brief summary.

2. Collaboration tools, environments, and infrastructure

Tool support developed specifically to support collaboration in software engineering falls into four broad categories. *Model-based* collaboration tools allow

engineers to collaborate in the context of a specific representation of the software, such as a UML diagram. *Process support* tools represent all or part of a software development process. Systems using explicit process representations permit software process modeling and enactment. In contrast, tools using an implicit representation of software process embed a specific tool-centric work process, such as the checkout, edit, checkin process of most SCM tools. *Awareness tools* do not support a specific task, and instead aim to inform developers about the ongoing work of others, in part to avoid conflicts. *Collaboration infrastructure* has been developed to improve interoperability among collaboration tools, and focuses primarily on their data and control integration. Below, we give a brief overview of previous work in these areas, to provide context for our recommendations for future areas of research on software collaboration technologies.

2.1 Model-based collaboration tools

Software engineering involves the creation of multiple artifacts. These artifacts include the end product, code, but also incorporate requirements specifications, architecture description, design models, testing plans, and so on. Each type of artifact has its own semantics, ranging from free form natural language, to the semi-formal semantics of UML, or the formal semantics of a programming language. Hence, the creation of these artifacts is the creation of models.

Creating each of these artifacts is an inherently collaborative activity. Multiple software engineers contribute to each of these artifacts, working to understand what each other has done, eliminate errors, and add their contributions. Especially with requirements and testing, engineers work with customers to ensure the artifacts accurately reflect their needs. Hence, the collaborative work to create software artifacts is the collaborative work to create models of the software system. Systems designed to support the collaborative creation and editing of specific artifacts are really supporting the creation of specific models, and hence support model-based collaboration. Collaboration tools exist to support the creation of every kind of model found in typical software engineering practice.

Requirements. In the requirements phase, there are many existing commercial tools that support collaborative development of requirements, including Rational RequisitePro [12], Borland CaliberRM [16], and Telelogic DOORS [7] (a more exhaustive list can be found at [17]). These tools allow multiple engineers to describe project use cases and requirements using natural language text, record dependencies among and between requirements and use cases, and change impact analyses. Integration with design and testing tools permits dependencies between requirements, UML models, and test cases to be explicitly represented.

Collaboration features vary across tools. Within RequisitePro, requirements are stored in a per-project requirements database, and can be edited via a Web-based interface, by editing a Word document that interacts with the database via a plugin, or by direct entry using the RequisitePro user interface. Multiple engineers can edit the requirements simultaneously via these interfaces. While cross-organization interaction is possible via the Web-based interface, the tool is primarily designed for within-organization use. RavenFlow [18] supports collaboration via a built-in checkout/checkin process on individual requirements. While most requirements tools are desktop applications, Gatherspace [19] and eRequirements [20] are web-based collaborative requirements tools, with capabilities only accessible via a Web browser.

Research on collaborative requirements tools has focused on supporting negotiation among stakeholders, use of new requirements engineering processes, and exploration of new media and platforms. WinWin was designed to support a requirements engineering process that made negotiation processes explicit in the interface of the tool, with an underlying structure that encouraged resolution of conflicts, creating "win-win" conditions for involved stakeholders [6]. ART-SCENE supports a requirements elicitation approach in which a potentially distributed team writes use cases using a series of structured templates accessible via a Web-based interface. These are then used to automatically generate scenarios that describe normal and alternative situations, which can then be evaluated by requirements analysts [21]. Follow-on work has examined the use of a mobile, PDA-based interface for ART-SCENE, taking advantage of the mobility of the interface to show use cases to customer stakeholders in-situ [22]. The Software Cinema project examined the use of video for recording dialog between engineers and stakeholders, allowing these conversations to be recorded and analyzed in depth [23].

Architecture. Though the creation of a final software architecture for a project is a collaborative and political activity, much of this collaboration takes place outside architecture-focused tools. Rational Software Architect is an UML modeling tool focused on software architecture. Engineers can browse an existing component library and work collaboratively on diagrams with other engineers, with collaboration mediated via the configuration management system. Research systems, such as ArchStudio [24] and ACMEStudio [25] typically support collaborative authoring by versioning architecture description files, allowing a turn-taking authoring model. The MolhadoArch system is more tightly integrated with an underlying fine-grain version control system, and hence affords collaboration at the level of individual model elements [26]. Supporting an explicitly web-based style of

collaboration, [27] describes a web-based tool that supports the ATAM architecture evaluation methodology.

Design. Today, due to the strong adoption of the Unified Modeling Language (UML), mainstream software design tools are synonymous with UML editors, and include Rational Rose [28], ArgoUML [29], Borland Together [30], Telelogic Rhapsody [31], and Altova UModel [32] (a more complete list is at [33]). Collaboration features of UML authoring tools mostly depend on the capabilities of the underlying software configuration management system. For example, ArgoUML provides no built-in collaboration features, instead relying on the user to subdivide their UML models into multiple files, which are then individually managed by the SCM system. Telelogic Rhapsody similarly depends on SCM support, but also provides some model merge capabilities, so that parallel work on the same UML model is possible. The Rosetta UML editor [34]was the first to explore Web-based collaborative editing of UML diagrams, using a Java applet diagram editor. Recently, Gliffy [35] has launched a web-based diagram editor that supports UML diagrams. It uses linear versioning to record document changes, and can inform other collaborators via email when a diagram has changed. SUMLOW supports same-time, same-place collaborative UML diagram creation via a shared electronic whiteboard [36].

Testing and Inspections. Like requirements, testing often involves substantial collaboration between an engineering team and customers. Testing interactions vary substantially across projects and organizations. Application software developers often make use of public beta tests in which potential users gain advance access to software, and report bugs back to the development team. As well, best practices for usability testing involves multiple people performing specific tasks under observation, another form of testing based collaboration. Adversarial interactions are also possible, as is the case with a formal acceptance test, where the customer is actively looking for lack of conformance to a requirements specification.

Within an engineering organization, testing typically involves collaboration between a testing group and a development team. The key collaborative tool used to manage the interface between testers (including public beta testers) and developers is the bug tracking (or issue management) tool [37]. Long a staple of software development projects, bug tracking tools permit the recording of an initial error report, prioritization, addition of follow-on comments and error data, linking together similar reports, and assignment to a developer who will repair the software. Once a bug has been fixed, this can be recorded in the bug tracking system. Search facilities permit a wide range of error reporting. A comparison of multiple issue tracking and bug tracking systems can be found at [38].

Software inspections involve multiple engineers reviewing a specific software artifact. As a result, software inspection tools have a long history of being collaborative. Hedberg [39] divides this history into early tools, distributed tools, asynchronous tools, and web-based tools. Early tools (circa 1990) were designed to support engineers holding a face-to-face meeting, while distributed tools (1992-93) permitted remote engineers to participate in an inspection meeting. Asynchronous tools (1994-97) relaxed the requirement for the inspection participants to all meet at the same time, and Web-based tools supported inspection processes on the Web (1997-onwards). MacDonald and Miller [40] also survey software inspection support systems as of 1999.

Traceability and consistency. While ensuring traceability from requirements to code and tests is not inherently a collaborative activity, once a project has multiple engineers, creating traceability links and ensuring their consistency is a major task. XLinkit performs automated consistency checks across a project [41], while [42] describes an approach for automatically inferring documentation to source code links using information retrieval techniques. Inconsistencies identified by these approaches can then form the starting point for examining whether there are mismatches between the artifacts created by different collaborators.

2.2 Process centered collaboration

Engineers working together to develop a large software project can benefit from having a predefined structure for the sequence of steps to be performed, the roles engineers must fulfill, and the artifacts that must be created. This predefined structure takes the form of a software process model, and serves to reduce the amount of coordination required to initiate a project. By having the typical sequence of steps, roles, and artifacts defined, engineers can more quickly tackle the project at hand, rather than renegotiating the entire project structure. Over time, engineers within an organization develop experience with a specific process structure. The net effect is to reduce the amount of coordination work required within a project by regularizing points of collaboration, as well as to increase predictability of future activity.

To the extent that software processes are predictable, software environments can mediate the collaborative work within a project. Process centered software development environments have facilities for writing software process models in a process modeling language (see [43] for a retrospective on this literature), then executing these models in the context of the environment. While a process model lies at the core of process centered environments, this process guides the collaborative activity of engineers working on other artifacts, and is not the focus of their collaboration. Hence, for example, the environment can manage the assignment of tasks to engineers, monitor their

completion, and automatically invoke appropriate tools. A far-from-exhaustive list of such systems includes Arcadia [44], Oz [45], Marvel [46] , ConversationBuilder [47], and Endeavors [9]. One challenge faced by such systems is the need to handle exceptions to an ongoing process, an issue addressed by [48].

2.3 Collaboration awareness

Software configuration management systems are the primary technology coordinating file-based collaboration among software engineers. The primary collaborative mechanism supported by SCM systems is the workspace. Typically each developer has their own workspace, and uses a checkout, edit, checkin cycle to modify a project artifact. Workspaces provide isolation from the work of other developers, and hence while an artifact is checked out, no other engineer can see its current state. Many SCM system permit parallel work on artifacts, in which multiple engineers edit the same artifact at the same time, using merge tools to resolve inconsistencies [49]. Workspaces allow engineers to work more efficiently by reduce the coordination burden among engineers, and avoiding turn-taking for editing artifacts. They raise several issues, however, including the inability to know which developers are working on a specific artifact. Palantir addresses this problem by providing engineers with workspace awareness, information about the current activities of other engineers [50]. By increasing awareness of the activities of other engineers, they are able to perform coordination activities sooner, and potentially avoid conflicts. Augur is another example of an awareness tool [51]. It provides a visualization of several aspects of the development history of a project, extracted from an SCM repository, thereby allowing members of a distributed project to be more aware of ongoing and historical activity.

2.4 Collaboration infrastructure

Various infrastructure technologies make it possible for engineers to work collaboratively. Software tool integration technologies make it possible for software tools (and the engineers operating them) to coordinate their work. Major forms of tool integration include data integration, ensuring that tools can exchange data, and control integration, ensuring that tools are aware of the activities of other tools, and can take action based on that knowledge. For example, in the Marvel environment, once an engineer finished editing their source code, it was stored in a central repository (data integration), and then a compiler was automatically called by Marvel (control integration) [46].

The Portable Common Tool Environment (PCTE) was developed from 1983-89 to create a broad range of interoperability standards for tool integration spanning data, control, and user interface integration [15]. Its greatest success was in defining a data model and interface for data integration. The WebDAV effort (1996-2006) aimed to give the Web have open interfaces for writing content, thereby affording data integration among software engineering tools, as well as a range of other content authoring tools [13, 14]. Today, the data integration needs of software environments are predominantly met by SCM systems managing files via isolated workspaces. However, the world of data integration standards and SCM meet in tools like Subversion [52] that use WebDAV as the data integration technology in their implementation.

For control integration there are two main approaches, direct tool invocation, and event notification services. In direct tool integration, a primary tool in an environment (e.g., an integrated development environment, like Eclipse) directly calls another tool to perform some work. When multiple tools need to be coordinated, a message passing approach works better. In this case, tools exchange event notification messages via some form of event transport. The Field environment introduced the notion of a message bus (an event notification middleware service) in development environments [53], with the Sienna system exemplifying more recent work in this space [54].

3. Future directions in software engineering collaboration

In the gaps between existing collaboration efforts are several areas for improving collaboration in software engineering. Desktop-based IDEs can enhance project collaboration if they are better integrated with web-based IDEs, a task that requires new interoperability standards. Support for multi-project and multi-organization collaboration has not been significantly addressed in the software engineering community, yet is an emerging concern in increasingly large systems-of-systems. Tools for capturing project-specific design tradeoff argumentation can help capture project rationale that is not explicitly represented in requirements, designs, or code. Software engineers have consistently appropriated new general-purpose communication technologies for project collaboration, often with some adaptation to the needs of projects. This may very well be the case with networked 3D game-like environments.

The sections below provide some ideas on improving collaboration within software projects, thereby providing a glimpse into the future of collaboration in software engineering. Mindful of novelist William Gibson's quote, "the future is here, it's just not evenly distributed yet," many of the trends below are already present, either in kernel form that can be extrapolated, or are in widespread use in contexts other than software engineering.

3.1 Integrating web and desktop environments

One clear trend in the overview of collaboration tools given in the previous section is the existence of web-based tools in every phase of software development. This mirrors the broader trend of moving applications to the web, afforded

by the greater interactivity of AJAX (asynchronous JavaScript and XML), more uniformity in JavaScript capabilities across browsers, and increasing processing power available in the browser. Web-based applications have the benefit of centralized tool administration, and straightforward deployment of new system capabilities. Traditionally, the most significant drawback to web-based applications has been the lack of user interface interactivity, and so graphics or editing intensive applications were traditionally not viewed as being suitable for the web. In the realm of software engineering, this meant that UML diagram editing and source code editing were relegated to desktop only applications.

Google Maps smashed the low interactivity stereotype in early 2005, and is now viewed as the vanguard of the loosely defined "Web 2.0" movement that began in 2004. Web 2.0 applications tend to have desktop-like user interface interactivity within a web browser, as well as facilities for other sites to integrate their data into the application, or integrate the site's data into another application. Recent web applications like Gliffy that support browser-based UML diagram editing can thus be viewed as part of the broader Web 2.0 movement.

Even though there is a trend towards creating web-based software engineering tools, there is also longstanding practice surrounding the use of desktop integrated development environments such as Visual Studio, Eclipse, and JBuilder. Each of these tools is a platform in its own right, with substantial ecosystems of third party extensions and substantial developer investment in work practices surrounding their use. Clearly, desktop IDEs are not going to be displaced by completely web-based environments. Instead, future projects are likely to adopt a mixture of web-based and desktop tools. Requirements and bug tracking tools are two areas where web-based tools already have widespread use, and where using the Web permits easier cross-organization collaboration, either to gather feedback from stakeholders about requirements, or to gather bug information from users. Code editing, and the edit-compile-debug cycle seems destined to remain on the desktop for now, since the desktop is more interactive, and the need for cross-organization collaboration in the coding phase is currently well handled by SCM systems.

With future environments composed of a mix of web-based and desktop systems, improvements in project collaboration can come from creating a series of interface standards by which desktop IDEs can access the information and capabilities of Web-based services. For example, it would be beneficial for developers to have rich access to bug tracking data within their desktop IDEs. This would permit better access to existing capabilities, including improved linking between bug reports and code modifications, and submission of bug reports from within the IDE. It would open up advanced capabilities as well, such as automatically searching the bug database for bug reports related to the code currently open in the code editor, and display of prior bug report information in currently edited code, giving developers improved rationale information for the code they see in front of them. In a similar vein, standard interfaces to requirements management software would permit better traceability between source code and requirements, as well as the ability to comment on requirements based on insights developed while coding. Ultimately, the existence of open standards for integrating desktop IDEs and web-based environments would permit more seamless interaction with the complex information space created by each software project. This would permit delivery of rich assistance services, as described in [63].

Open research questions revolve around the type and characteristics of the interface standards. While creating interfaces to access the capabilities of a bug tracking system or a requirements management system appear to be useful, the exact nature of the capabilities supported by these interfaces is somewhat unclear. For example, in the case of bug tracking software, there are many different use models and data models supported by these systems. This raises the traditional issues faced by interoperability standards: should the standard aim for a union of all available features with each system supporting only a subset, or should it support only a limited set of features that are frequently used? This, in turn raises questions about exactly which features are most useful, and how an engineer might incorporate these features into their work practices.

3.2 Broader participation in design

Many forms of software have high costs for acquiring and learning the software, leading to lock-in for its users. This is especially true for enterprise software applications, where there can be substantial customization of the software for each location. This leads to customer organizations having a need to deeply understand product architecture and design, and to have some influence over specific aspects of software evolution to accommodate their evolving needs. In current practice, customers are consulted about requirements needs, which are then integrated into a final set of requirements that drive the development of the next version of the software. Customers are also usually participants in the testing process via the preliminary use and examination of various beta releases. In the current model, customers are engaged during requirements elicitation, but then become disengaged for the requirements analysis, design, and coding phases, only to reconnect again for the final phase of testing.

Broadened participation by customers in the requirements, design, coding and early testing phases would keep customers engaged during these middle stages, allowing them to more actively ensure their direct needs are met. While open source software development can be viewed as

an extreme of what is being suggested here, in many contexts broadening participation need not mean going all the way to open source. Development organizations can have proprietary closed-source models in which they still have substantial fine-grain engagement with customers in which customers are directly engaged in the requirements, design, coding, and testing process. Additionally, broadening participation does not necessarily mean that customers would be given access to all source code, or input on all decisions. Nevertheless, by increasing the participation of the direct end users of software in its development, software engineers can reduce the risk that the final software does not meet the needs of customer organizations. As in open source software, a more broadly participative model can allow customers to fix those bugs that mostly directly affect them, even if, from a global perspective, they are of low priority, and hence unlikely to be fixed in traditional development. A participatory development model could also permits customers to add new features, thereby better tailoring the software to their needs.

The trend toward providing support for distributed development teams in a wide range of development tools makes a broader engagement possible. Open source SCM tools such as Subversion, as well as web-based requirements tools and problem tracking tools make it possible to coordinate globally distributed teams. To date, it has primarily been open source software projects that take full advantage of tool distribution to have broadly participative development teams drawing from a broad array of organizations and individuals. Commercial development has made use of this improved distribution support as well, but generally to support globally distributed teams in the same organization, or contractor/subcontractor relationships. Commercial projects currently do not leverage existing tools' support for distributed teams to incorporate greater customer participation.

3.3 Capturing rationale argumentation

An important part of a software project's documentation is a record of the rationale behind major decisions concerning its architecture and design. As new team members join a project over its multi-year evolution, an understanding of project rationale makes it less likely that design assumptions and choices will be accidentally violated. This, in turn, should result in less code decay. A recent study [55] shows that engineers recognize the utility of documenting design rationale, but that better tool support is needed to capture design choices and the reasons for making them.

Technical design choices are often portrayed as being the outcome of a rational decision making process in which an engineer carefully teases out the variables of interest, gathers information, and then makes a reasoned tradeoff. What this model does not reflect is the potential for disagreement among many experienced software engineers on how to assess the importance of factors affecting a given design. One of the strongest design criteria used in software engineering is design for change, which inherently involves making predictions about the future. Clearly we do not yet have a perfect crystal ball for peering into the future, and hence experienced engineers naturally have differing opinions on which changes are likely to occur, and how to accommodate them. As well, architectural choices often involve decisions concerning which technical platform to choose (e.g., J2EE, Ruby on Rails, PHP, etc.), requiring assessments about their present and future qualities. As a result, the design process is not just an engineer making rational decisions from a set of facts, but instead is a predictive process in which multiple engineers argue over current facts and future potentials. Architecture and design are *argumentative* processes in which engineers resolve differences of prediction and interpretation to develop models of the software system's structure. Since only one vision of a system's structure will prevail, the process of architecture and design is simultaneously cooperative and competitive.

Effective recording of a project's rationale requires capturing the argumentation structure used by engineers in their debates concerning the final system structure. Outside of software engineering, there is growing interest in visual languages and software systems that model the structure of arguments [56]. While models vary, argumentation support systems generally record the question or point that is being contested (argued about), statements that support or contest the main point, as well as evidence that substantiates a particular statement. Argumentation structures are generally hierarchical, permitting pro and con arguments to be made about individual supporting statements under the main point. For example, a "con" argument concerning the use of solar panels as the energy source for a project might state that solar electric power is currently not competitive with existing coal-fired power plants. A counter to that argument might state that while this is true of wholesale costs, solar energy is competitive with peak retail electric costs in many markets.

Providing collaborative tools to support software engineers in the recording and visualization of architecture and design argumentation structures would do a better job of capturing the nuances and tradeoffs involved in creating large systems. They would also better convey the assumptions that went into a particular decision, making it easier for succeeding engineers to know when they can safely change a system's design.

There are several open research issues in collaborative argumentation support for software engineering. It is currently unclear what argumentation structure would best support design rationale capture for software projects. Existing argumentation systems often start their

argumentation structure with a single driving question, such as "can computers think?" as exemplified by Robert Horn's set of seven poster-sized argumentation maps (http://www.macrovu.com/CCTGeneralInfo.html).
Software feature variability analysis research ([57] is one example) generally represents software as having a number of features, each of which is hierarchically broken down into multiple sub-features and choices. This suggests that argumentation structures for software would have multiple starting points (a forest instead of a tree), and should also be integrated with its variability structure. Some existing approaches for modeling design rationale can be found in [58] and [59].

Since argumentation involves multiple actors, argumentation support systems need to be collaborative, allowing many people to modify the evolving argument. A wiki-like system with a built-in notion of argumentation might be a useful way to collect and structure software system design rationale. Recent work on tool support for capturing argumentation surrounding design rationale is [60] which describes the Compendium project.

3.4 Using novel communication and presence technologies

Software engineers have a long track record of integrating new communication technologies into their development processes. Email, instant messaging, and web-based applications are very commonly used in today's projects to coordinate work and be aware of whether other developers are currently active (present). As a result, engineers would be expected to adopt emerging communication and presence technologies if they offer advantages over current tools.

Networked collaborative 3D game worlds are one such emerging technology. The past few years have witnessed the emergence of massively-multiplayer online (MMO) games, the most popular being World of Warcraft (WoW). These games support thousands of simultaneous players who interact in a shared virtual world. Each player controls an avatar, a graphic representation of the player in the world. Communication features supported by games include instant messaging, email-like message services, and presence information (seeing another active player's avatar). Many players use a third-party voice communication service to coordinate groups of player engaged in joint combat during quests.

Steve Dossick's PhD dissertation [61] describes early work on the use of 3D game environments to create a "Software Immersion Environment" in which project artifacts are arranged in a physical 3D space, a form of virtual memory palace. Only recently have MMOs like Second Life emerged that are not explicitly role-playing game worlds, and hence are framed in a way that makes them potentially usable for professional work. While Second Life's focus on leisure activities makes it unpalatable for all but the most adventurous of early adopters, these environments still hint at their potential for engineering collaboration.

One problem in both distributed collaboration between multiple organizations and telecommuting workers is the need for improved awareness of the presence and activity of co-workers. While instant messaging software provides presence information, it is limited in its ability to provide awareness. For example, it cannot show that people have gathered for a meeting. A 3D virtual environment could potentially let other engineers know about each others activities, and be able to have meetings where distributed project participants can be physically proximate, at least in the virtual world. The 3D environment could also be used to provide a physical topology to the structure of information in a large software project, which might permit more rapid browsing and access of project data.

Very speculatively, it might also be possible for projects to graft the narrative and reward structure of MMO role-playing games onto traditional engineering project work. In a game like WoW, in order to advance, players go on quests, a goal-oriented activity in the game world (go to a dungeon, kill all monsters, retrieve valuable artifact, return for reward). As players perform quests, their abilities increase, which is reflected in their character "leveling up." Many players find the game setting (typically from fantasy or science fiction) combined with the quest narrative and leveling reward structure to be very motivational, and for some addictive. It would be intriguing for a project to map development activities onto this style of gameplay. One could imagine engineer experience and capabilities represented in the form of levels, with project subgoals broken down into quest-like units. If this could tap into the motivational aspects of MMO style gameplay, it might increase team productivity by providing a range of incentive structures in addition to the traditional ones of salary, promotion, and satisfaction at completing a project.

There is a range of research issues inherent in the use of 3D virtual environments as a collaboration infrastructure. One issue is how to synchronize physical and virtual worlds. If a number of workers are in the office, and some are not, how should the behavior of the office workers in the physical office be reflected in the virtual environment? In reverse, should physical office workers be made aware of the presence of virtual workers, perhaps by the use of screens placed in the office, or a form of ambient awareness such as a desk light that goes on when virtual workers are present? The architecture of virtual project spaces is also unclear. One possibility is to have the virtual space represent the organization of the various software project artifacts including requirements, designs, code, test cases, and so on. Alternately, the virtual space could be a form of idealized work environment, where everyone has a nice, large office with window. Combinations of the two are also possible, given the lack of real-world constraints. Finally, the utility of adopting a 3D virtual world needs careful examination,

as the benefits of the technology need to clearly exceed the costs. It is currently very unclear that this is true.

3.5 Improved assessment of collaboration technology

Adoption of collaboration technologies in a software development organization involves the injection of new technology and associated (evolving) work practices surrounding its use, into the highly complex and variable activity of writing software. Since software effort estimation remains a difficult task even for experienced engineers, the productivity and overall outcome of software projects is highly variable. As a result, assessing the impact of the introduction of new technology into a project is difficult, and usually subjective. This creates substantial difficulty for the objective assessment of collaboration technology. Without the ability to objectively assess the pros and cons of specific collaboration tools, forward progress in the field of software collaboration support tools is hard to measure. This, in turn, yields the potential for churn as it is difficult to assess whether a new idea is truly beneficial, or, if a slight tweak on some old idea is an improvement that would lead to adoption this time around.

It is clear that the past 20 years have brought tremendous advances in collaboration tools for software engineers, in the form of Internet-aware SCM tools, broad adoption of email, web-based bug tracking systems, instant messaging, and so on. Just as clearly, there are no studies that quantify the benefits received from using these collaboration tools. Developing improved methods for assessing the impact of collaboration tools would boost research in this area by increasing confidence in positive results, and making it easier to convince teams to adopt new technologies. Work by de Souza and Redmiles on how activity theory could inform research on collaborative software engineering tools offers one possible framework for structuring the assessment of collaboration tools [62].

4. Summary

Software engineering project work is a highly cooperative activity, and promises to continue in this way. To the extent that advances in software engineering team collaboration can reduce accidental difficulties inherent in the coordination of large teams of people, and can better leverage the unique talents and capabilities of each team member, new work in this area will improve the productivity and quality of software projects. This paper has presented an overview of the goals of collaboration in software engineering, and a brief survey of existing collaboration tools. The distinguishing quality of software engineering collaboration tools as being model-based helps to focus this survey. An important trend uncovered by this survey is the movement towards web-based tools in all phases of software development. However, at present there is no integrated web-based environment that covers the entire software development lifecycle, with existing tools typically covering a single phase, such as requirements, or UML diagramming.

A series of potential future directions for collaboration research in software engineering were presented. These include better integration of desktop and web-based development environments, broadening participation in software projects, capturing design rationale in the form of an argumentation structure, and the use of 3D virtual game environments as a presence, communication, and potentially motivational infrastructure. Finally, the challenges inherent in assessing collaboration technologies leads to a plea for improvement in how such systems are assessed.

Predicting the future is notoriously difficult. So, while the specific future directions outlined herein may not necessarily come to pass, it is clear that our improving understanding of software engineering as a collaborative endeavor, combined with the rapidly declining cost of communications and the rapidly increasing capabilities of our computational and communication platforms, will lead to improvements in how engineers collaborate to create large software artifacts.

Acknowledgements

Discussions with Rohit Khare contributed the idea of broadening participation in design. I am grateful to Guozheng Ge for assistance in preparing the manuscript.

References

[1] S. McConnell, "Lifecycle Planning," in *Rapid Development: Taming Wild Software Schedules* Redmond, WA: Microsoft Press, 1996.

[2] R. Grinter, "Systems Architecture: Product Designing and Social Engineering," in *ACM Conference on Work Activities Coordination and Collaboration (WACC'99)*, San Francisco, California, 1999, pp. 11-18.

[3] T. W. Malone and K. Crowston, "The Interdisciplinary Study of Coordination," in ACM Computing Surveys (CSUR), vol 26, no 1, pp. 87-119, 1994.

[4] C. R. B. d. Souza, D. F. Redmiles, L.-T. Cheng, D. R. Millen, and J. Patterson, "How a Good Software Practice Thwarts Collaboration - The Mulitple Roles of APIs in Software Development," in *Proc. Foundations of Software Engineering (FSE 2004)*, Newport Beach, CA, 2004, pp. 221-230.

[5] M. S. Ackerman and D. W. McDonald, "Collaborative Support for Informal Information in Collective Memory Systems," in *Information Systems Frontiers,* vol. 2, no. 3/4, pp. 333-347, 2000.

[6] B. Boehm and A. Egyed, "Software Requirements Negotiation: Some Lessons Learned," in *the 20th International Conference on Software Engineering (ICSE'98)*, Kyoto, Japan, 1998, pp. 503-507.

[7] Telelogic, "Getting Started with DOORS (DOORS 7.1)," Feb 5, 2004, http://endymion.ugent.be/~stijn/doors/doors_getting_started.pdf.
[8] The Bugzilla Team, "The Bugzilla Guide - 2.23.3 Development Release," October 15, 2006, http://www.bugzilla.org/docs/tip/html/.
[9] G. A. Bolcer and R. N. Taylor, "Endeavors: a Process System Integration Infrastructure," in *4th International Conference on the Software Process (ICSP'96)*, Brighton, UK, 1996, pp. 76-89.
[10] B. S. Lerner, L. J. Osterweil, Stanley M. Sutton Jr., and A. Wise, "Programming Process Coordination in Little-JIL Toward the Harmonious Functioning of Parts for Effective Results," in *European Workshop on Software Process Technology*, 1998.
[11] Microsoft Corportation, "Microsoft Office Project Standard 2007 Product Guide," April 2006, http://office.microsoft.com/en-us/project/HA101680121033.aspx.
[12] Rational Software Corporation, "Rational RequisitePro User's Guide," June 2003, http://www.se.fh-heilbronn.de/usefulstuff/Rational%20Rose%202003%20Documentation/reqpro_user.pdf.
[13] L. Dusseault, *WebDAV: Next-Generation Collaborative Web Authoring*, Prentice Hall PTR, 2003.
[14] E. J. Whitehead, Jr. and Y. Y. Goland, "WebDAV: A Network Protocol for Remote Collaborative Authoring on the Web," in *6th European Conference on Computer Supported Cooperative Work (ECSCW'99)*, Copenhagen, Denmark, 1999, pp. 291-310.
[15] L. Wakeman and J. Jowett, *PCTE: The Standard for Open Repositories*: Prentice Hall, 1993.
[16] Borland Software Corporation, "CaliberRM 2006 User Tutorial," Nov 2006, http://info.borland.com/techpubs/caliber_rm/2006/EN/CaliberRM%20Tutorial.pdf.
[17] Ludwig Consulting Services, "Requirement Management Tools," 2006, http://www.jiludwig.com/Requirements_Management_Tools.html.
[18] Ravenflow, "RAVEN: Requirements Authoring and Validation Environment," 2007, http://www.ravenflow.com/products/index.php.
[19] GATHERSPACE.COM, "Gatherspace: Requirements Management and Use Case Software," 2007, http://www.gatherspace.com/static/product2.html.
[20] eRequirements, "eRequirements Product Tour," 2007, http://www.erequirements.com/app?service=page/ProductTour.
[21] N. Maiden, "Discovering Requirements with Scenarios: The ART-SCENE Solution," in *ERCIM News,* vol. 58, July 2004.
[22] N. Maiden, N. Seyff, P. Grunbacher, O. Otojare, and K. Mitteregger, "Making Mobile Requirements Engineering Tools Usable and Useful," in *14th Int'l Requirements Engineering Conference (RE'06)*, 2006, pp. 26-35.
[23] O. Creighton, M. Ott, and B. Bruegge, "Software Cinema-Video-based Requirements Engineering," in *14th Int'l Requirements Engineering Conference (RE'06)*, 2006, pp. 106-115.
[24] UCI Software Architecture Group, "ArchStudio 4: Software and Systems Architecture Development Environment," 2007, http://www.isr.uci.edu/projects/archstudio/.
[25] A. Kompanek, "Modeling a System with Acme," 1998, http://www.cs.cmu.edu/~acme/html/WORKING-%20Modeling%20a%20System%20with%20Acme.html.
[26] T. N. Nguyen and E. V. Munson, "Object-oriented Configuration Management Technology can Improve Software Architectural Traceability," in *3rd ACIS International Conference on Software Engineering Research, Management and Applications (SERA'05)*, Mount Pleasant, Michigan, USA, 2005, pp. 86-93.
[27] P. Maheshwari and A. Teoh, "Supporting ATAM with a Collaborative Web-based Software Architecture Evaluation Tool," in *Science of Computer Programming,* vol. 57, no. 1, pp. 109-128, 2005.
[28] IBM, "Rational Rose Product Overview," 2007, http://www-306.ibm.com/software/awdtools/developer/rose/index.html.
[29] A. Ramirez, P. Vanpeperstraete, A. Rueckert, K. Odutola, J. Bennett, L. Tolke, and M. v. d. Wulp, "ArgoUML User Manual (v0.22)," 2006, http://argouml-stats.tigris.org/documentation/manual-0.22/.
[30] Borland Software Corporation, "Borland Together Product Overview," 2007, http://www.borland.com/us/products/together/index.html.
[31] Telelogic, "Rhapsody: Model-Driven Development with UML 2.0, SysML and Beyond," 2007, http://www.ilogix.com/sublevel.aspx?id=53.
[32] Altova, "UModel Product Overview," 2007, http://www.altova.com/products/umodel/uml_tool.html.
[33] Wikipedia.org, "List of UML Tools," 2007, http://en.wikipedia.org/wiki/List_of_UML_tools.
[34] T. C. N. Graham, A. G. Ryman, and R. Rasouli, "A World-Wide-Web Architecture for Collaborative Software Design," in *Software Technology and Engineering Practice (STEP'99)*, Pittsburgh, PA, 1999, pp. 22-29.
[35] gliffy.com, "Gliffy Product Information," 2007, http://gliffy.com/features.shtml.
[36] Q. Chen, J. Grundy, and J. Hosking, "An e-whiteboard Application to Support Early Design-stage Sketching of UML Diagrams," in *IEEE Symposium on Human Centric Computing Languages and Environments*, Auckland, New Zealand, 2003, pp. 219-226.
[37] S. V. Shukla and D. F. Redmiles, "Collaborative Learning in a Software Bug-Tracking Scenario," in *Workshop on Approaches for Distributed Learning through Computer Supported Collaborative Learning*, Boston, MA, 1996.
[38] Wikipedia.org, "Comparison of Iusse Tracking Systems," http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems.
[39] H. Hedberg, "Introducing the Next Generation of Software Inspection Tools," in *Product Focused Software Process Improvement (LNCS 3009)*, 2004, pp. 234-247.
[40] F. Macdonald and J. Miller, "A Comparison of Computer Support Systems for Software Inspection," *Automated Software Engineering,* vol. 6, no. 3, pp. 291-313, 1999.
[41] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein, "xlinkit: A Consistency Checking and Smart Link Generation Service," *ACM Transactions on Internet Technology (TOIT),* vol 2, no 2, pp. 151-185, May 2002.
[42] A. Marcus and J. I. Maletic, "Recovering Documentation-to-source-code Traceability Links using Latent Semantic Indexing," in *Proc. 25th Int'l Conf. on Software Engineering (ICSE'03)*, Portland, Oregon, 2003, pp. 125-135.

[43] L. Osterweil, "Software Processes are Software too," in *9th International Conference on Software Engineering*, Monterey, CA, 1987, pp. 2-13.

[44] R. Kadia, "Issues Encountered in Building a Flexible Software Development Environment," in *ACM SIGSOFT 92: 5th Symposium on Software Development Environments*, Tyson's Corner, Virginia, 1992, pp. 169-180.

[45] I. Z. Ben-Shaul, "Oz: A Decentralized Process Centered Environment (PhD Thesis)," in *Department of Computer Science*: Columbia University, Dec 1994.

[46] I. Z. Ben-Shaul, G. E. Kaiser, and G. T. Heineman, "An Architecture for Multi-user Software Development Environments," in *ACM SIGSOFT 92: 5th Symposium on Software Development Environments*, Tyson's Corner, Virginia, 1992, pp. 149-158.

[47] S. M. Kaplan, W. J. Tolone, A. M. Carroll, D. P. Bogia, and C. Bignoli, "Supporting Collaborative Software Development with ConversationBuilder," in *ACM SIGSOFT 92: 5th Symposium on Software Development Environments*, Tyson's Corner, Virginia, 1992, pp. 11-20.

[48] P. J. Kammer, G. A. Bolcer, R. N. Taylor, A. S. Hitomi, and M. Bergman, "Techniques for Supporting Dynamic and Adaptive Workflow," in *Computer Supported Cooperative Work (CSCW),* vol. 9, no. 3/4, pp. 269-292, 2000.

[49] T. Mens, "A State-of-the-Art Survey on Software Merging," *IEEE Transactions on Software Engineering,* vol 28, no 5, pp. 449-462, 2002.

[50] A. Sarma, Z. Noroozi, and A. v. d. Hoek, "Palantír: Raising Awareness among Configuration Management Workspaces," in *25th International Conference on Software Engineering*, Portland, Oregon, May 2003, pp. 444-454.

[51] J. Froehlich and P. Dourish, "Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams," in *26th Int'l Conference on Software Engineering (ICSE'04)*, Edinburgh, Scotland, UK, 2004, pp. 387-396.

[52] Tigris.org, "Tigris.org: Open Source Software Engineering Tools," 2007, http://www.tigris.org/.

[53] S. P. Reiss, *The Field Programming Environment: A Friendly Integrated Environment for Learning and Development*. Norwell, MA: Kluwer, 1995.

[54] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service," in ACM Transactions on Computer Systems, vol. 19, no. 3, pp. 332-383, Aug 2001.

[55] A. Tang, M. A. Babar, I. Gorton, and J. Han, "A Survey of the Use and Documentation of Architecture Design Rationale," in *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, Pittsburgh, PA, 2005.

[56] *Visualizing Argumentation: Software Tools for Collaborative and Educational Sense-Making*, P. A. Kirschner, S. Buckingham-Shum, and C. S. Carr, Eds. London: Springer-Verlag, 2003.

[57] W. Zhang, H. Mei, and H. Zhao, "Feature-driven Requirement Dependency Analysis and High-level Software Design," in *Requirements Engineering,* vol 11, no 3, pp. 205-220, 2006.

[58] *Design Rationale: Concepts, Techniques, and Use*, T. P. Morgan and J. M. Carroll, Eds.: Lawrence Erlbaum Associates, 1996.

[59] *Rationale Management in Software Engineering*, A. H. Dutoit, B. Paech, R. Mccall, and I. Mistrik, Eds.: Springer-Verlag, New York, 2006.

[60] S. J. B. Shum, A. M. Selvin, M. Sierhuis, J. Conklin, C. B. Haley, and B. Nuseibeh, "Hypermedia Support for Argumentation-Based Rationale: 15 Years on from gIBIS and QOC " in *Rationale Management in Software Engineering*, A. H. Dutoit, B. Paech, R. Mccall, and I. Mistrik, Eds., 2005.

[61] S. E. Dossick, "A Virtual Environment Framework for Software Engineering (PhD Thesis)," Dept. of Computer Science, Columbia University, 2000.

[62] C. R. B. d. Souza and D. F. Redmiles, "Opportunities for Extending Activity Theory for Studying Collaborative Software Development," in *Workshop on Applying Activity Theory to CSCW Research and Practice (with the 8th European Conference of Computer-Supported Cooperative Work, ECSCW'03)*, Helsinki, Finland, 2003.

[63] A. Zeller, "The Future of Programming Environments: Integration, Synergy, and Assistance," in *Future of Software Engineering 2007*, L. Briand and A. Wolf (eds.), IEEE-CS Press, 2007.

Collaboration in Software Engineering: A Roadmap

Jim Whitehead

Jim Whitehead is an Associate Professor of Computer Science at the University of California, Santa Cruz. Jim received the Bachelor of Science in Electrical Engineering from the Rensselaer Polytechnic Institute in 1989, and a PhD in Information and Computer Science from the University of California, Irvine in 2000, under his advisor, Richard N. Taylor. From 1996-2004, Jim created and led the Internet Engineering Task Force Working Group on Web Distributed Authoring and Versioning (WebDAV), and participated in the creation of the DeltaV follow-on standard for versioning and configuration management. In 2005-06, Jim led efforts to create a new major at UC Santa Cruz, the Bachelor of Science in Computer Science: Computer Game Design.