Characterization and Classification of Collaborative Tools

Javier Soriano
Universidad Politécnica de Madrid (UPM), Spain

Rafael Fernández
Universidad Politécnica de Madrid (UPM), Spain

Miguel Jiménez
Universidad Politécnica de Madrid (UPM), Spain

INTRODUCTION

Traditionally, collaboration has been a means for organizations to do their work. However, the context in which they do this work is changing, especially in regards to where the work is done, how the work is organized, who does the work, and with this the characteristics of collaboration. Software development is no exception; it is itself a collaborative effort that is likewise affected by these changes. In the context of both open source software development projects and communities and organizations that develop corporate products, more and more developers need to communicate and liaise with colleagues in geographically distant places about the software product they are conceiving, designing, building, testing, debugging, deploying and maintaining. Thus, work teams face sizeable collaborative challenges, for which they have need of tools that they can use to communicate and coordinate their work efficiently.

The response is the collaborative development environment (CDE), a virtual space where all the software project stakeholders, possibly distributed in time and space, can negotiate, brainstorm, discuss, share knowledge and resources and, generally, labor together to carry out some task in the context of a software development process (Booch & Brown, 2003).

The collaborative needs of a team depend largely on factors related to the environment, such as the team’s organizational structure, its geographical and temporal distribution, the target software domain, the software product structure and the actual team members. Each of these factors highlights a different aspect of collaboration. All existing collaborative development tools have been conceived considering different aspects of collaboration, each one meeting particular sets of needs and considering the particular functional, organizational, temporal and spatial characteristics of this collaboration.

As part of Morfeo (2005), we conducted a survey of existing software products and collaborative sites. This survey suggests that the likelihood of being able to develop a CDE meeting all the collaborative needs of the software process is remote, especially taking into account that these needs change depending on contextual factors such as the above. In this respect, the “contextual” approach to collaboration is gaining strength. This approach enables the holistic integration and deployment of collaborative components and services in a CDE suited for a particular context, as opposed to a monolithic conception of such an environment. Therefore it is worth examining existing classification frameworks. There are different classification frameworks that order collaborative tools by the needs they satisfy, each from a different viewpoint. A team that is acquainted with these frameworks can contextualize the range of available collaborative tools, and compare them from different viewpoints and on the basis of assembled criteria sets. This way it can make a grounded decision on what collaborative tools best meet its needs.

This chapter starts with a definition and preliminary characterization of collaborative tools and CDEs. Without claiming to be exhaustive, it goes on to describe some of the most representative frameworks that have been developed to date. The chapter then shows the resulting categorization for each approach and presents some representative tools for each particular category. Finally, it suggests ideas on how to use these frameworks to select the best collaborative tools for a particular work team and development project.
COLLABORATIVE TOOLS AND DEVELOPMENT ENVIRONMENTS

The issue of CDEs was taken up perhaps for the first time back in 1984, when Iren Greif and Paul Cashmand organized a workshop that brought together an influential group of people to examine how to apply technology within a collaborative work environment. This meeting was the source of the “computer-supported cooperative work (CSCW)” concept (Grudin, 1994), which aimed to find an answer to how computer systems can support and coordinate collaborative activities.

A few years later, after further research into the concept of CSCW, Malone and Crowston (1994) introduced coordination theory. This theory was conceived on the basis of research in several different disciplines like computer science, organization theory, management science, economics, linguistics, and psychology. The theory defined coordination as the way of managing dependencies between activities. By characterizing the different types of possible dependencies between task activities, Malone and Crowston were able to identify and, consequently, manage the so-called coordination processes. This investigation identified some of the problems that future CDEs would have to deal with, such as resources allocation, as well as possible solutions.

Years later, when the technology was far enough evolved and after the Internet had materialized, these coordination processes and all the years of CSCW research led to collaborative tools capable of improving not only the development of software applications, but also the networked exchange of information and ideas from different branches of knowledge. Such an exchange often involved users who had possibly never worked together before and did not even know each other based on geographically distant places and even had to overcome time differences. This, in turn, led to the concept of groupware (Baecker, 1993), that is computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment. Groupware came about thanks to the enabling technologies of computer networking, software and services that materialized the ideas that had emerged from CSCW research (Engelbart, 1992).

Predictably, this activity yielded the first tangible definition of CDEs as “virtual spaces wherein all the stakeholders of a project—even if distributed by time or distance—can 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” (Booch & Brown, 2003). We can add to this definition by saying that a CDE holistically integrates multiple collaborative tools and resources thanks to which it offers a set of services to aid all the stakeholders in the software development area, including managers, developers, users, commercial software manufacturers and software product support enterprises, to communicate, cooperate and liaise. CDEs consider software development’s social side and assure that the people who design, produce, maintain, commercialize and use software are aware of and communicate about the activities of the others simply, efficiently and effectively, also encouraging creativity and driving innovation.

CLASSIFICATION FRAMEWORKS: A SURVEY

As previously mentioned, the contextual approach to collaboration enables the holistic integration and deployment of collaborative components and services in a CDE suited for a particular context as opposed to a monolithic conception of such an environment. It is, therefore, worth examining the existing classification frameworks that order collaborative tools by the needs they satisfy, each from a different viewpoint. Considering these frameworks in a coordinated fashion, a team can compare the range of collaborative tools available from different viewpoints and on the basis of assembled criteria sets to be able to make a grounded decision on what collaborative tools best meet its needs.

Groupware typologies (Grudin, 1994). The first of the frameworks that we analyze is Grudin’s classification, also known as “groupware typologies.” The term “groupware” refers to the set of CSCW applications used to do collaborative teamwork. From a multidisciplinary viewpoint and based on an earlier framework (DeSanctis & Gallupe, 1987), Grudin elaborates a groupware typology based on a space-time categorization. To implement this classification, Grudin considers that the activities can be performed at a single site, at different sites, all of which are known to participants, via e-mailing or at different sites, not all of which are known to participants, as in messaging through newsgroups. Additionally, an activity, for example a...
Characterization and Classification of Collaboration Tools

**Figure 1. Grudin’s classification**

<table>
<thead>
<tr>
<th>Time</th>
<th>Place</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same</td>
<td>Same</td>
</tr>
<tr>
<td>Different but predictable</td>
<td>Different but predictable</td>
</tr>
<tr>
<td>Different and unpredictable</td>
<td>Different and unpredictable</td>
</tr>
</tbody>
</table>

Meeting, can be performed in real time, but also at different times, albeit within a more or less scheduled or expected time interval, such as sending an e-mail and getting the response within a day. Finally, an activity can be carried out at different and unpredictable times, as in open-ended collaborative writing projects. This approach is outlined as a 3x3 grid in Figure 1.

This is a very easy-to-use framework and is therefore widely employed. It facilitates communication, especially by groupware developers, but not without risk: it obscures an organizational perspective. Most real work does not fall into any one category, because face-to-face meetings, for example, often take place at the same time as distributed and asynchronous communications as a task progresses.

**Malone and Crowston (1994)** research an area called coordination theory, which focuses on the interdisciplinary study of coordination. Research in this area uses and extends ideas about coordination from disciplines such as computer science, organization theory, operations research, economics, linguistics, and psychology.

They define coordination as the process of managing dependencies among activities. They then use that concept to shape a framework to characterize different kinds of dependencies (such as managing shared resources, managing producer/consumer relationships, and so on) and identify the coordination processes that can be used to manage them (i.e., priority order, notifications, tracking, goal selections, task decompositions). They give us examples of how different disciplines have analyzed coordination processes, identifying similarities among concepts that suggest how ideas can be transported back and forth across disciplinary boundaries.

In that way, they proclaim the use of coordination concepts from other disciplines (philosophy, artificial intelligence, etc.) to suggest design ideas for cooperative work tools. The framework they suggest provides a natural way of classifying existing cooperative work systems according to the coordination processes they support.

Table 1 shows some of the typical coordination processes that collaborative tools support.

**Nutt (1996).** There are several approaches to how computer technology should assist collaboration across the network. They range from approaches where the coordination of work is uniquely human-controlled, referred to as situated work, to workflow-based approaches where the computer is involved in scheduling the group’s work. Workflow technology has evolved from a modeling focus to flexible model-based systems to support collaborative work.

A groupwork procedure is defined in a workflow system by a workflow model composed of a set of discrete work steps with explicit specifications of how a unit of work flows through the different steps. Workflow models have also been used to describe how the procedure performs the work and to analyze how

**Table 1. Coordination processes typically supported by collaborative tools**

- Managing shared resources (task assignment and prioritization)
- Managing producer/consumer relationships (sequencing prerequisites)
- Managing simultaneity constraints (synchronizing)
- Managing task/subtask relationship (goal decomposition)
- Group decision making
- Communication
a procedure behaves. Workflow languages contain
constructs to define a set of steps to represent a unit of
work, a sequential computational language to provide
an interpretation for each step, and a coordination lan-
guage to define how the work flows among the steps.
The fundamental purpose of a computational representa-
tion language is to enable one person to describe how
a step is accomplished for the benefit of other human
readers. The coordination language specifies how a
workcase passes from step to step; this is a distinctive
feature of workflow languages.

Within this workflow system framework, Nutt
defines a 3D domain space based on the underlying
workflow model and more specifically on the mode in
which the workflow model represents a work proce-
dure. Therefore, a workflow model can represent parts
of the process that must be performed for the process
to be acceptable, and parts that can be performed by a
computer or human. Models should represent proce-
dures according to the way the model is to be used, as
defined by three different criteria: conformance level
that is required by the organization for which the process
is a model, the detail level of the description and the
operativity of the model. Consequently, the resulting
framework can classify the models that represent just
structured or explicit work, models conceived to deal
with unstructured work, descriptive and analytical work-
flow models and conventional workflow models.

Conradi and Westfechtel (1998) developed a frame-
work to classify collaborative tools based on their expe-
rience in software configuration management (SCM).
This framework is based on a taxonomy used originally
to classify SMC systems. The framework classifies
SMC systems according to a three-layer hierarchy. The
first level defines categories (e.g., “general,” “product
space,” etc.). Each category includes multiple features,
where each category is a dimension of the classifica-
tion scheme (“environment,” “object management,”
etc.). A feature can have values based on some sort of
equation (“language-based,” “structure-oriented,”
etc.), and be single- or multi-valued.

Continuous coordination (Van der Hoek et al.,
2004, 2006) introduces and explores an alternative
approach to computer-supported cooperative work called
continuous coordination. It blends the best aspects of
the more formal, process-oriented approach with those
of the more informal, awareness-based approach. The
classification framework they propose organizes col-
aborative tools into tools that fit the formal pattern,
tools that fit the informal pattern and tools that combine
both approaches.

The tools that follow the formal, process-oriented ap-
proach define process models that implicitly or explicitly
divide work into periodically resynchronized multiple
independent tasks according to the specific tool. In this
approach, the tool is responsible for the coordination
protocols that developers are to follow. This approach
can be characterized as inherently group-centric: it
makes the group as a whole the important entity by
providing a scalable, predictable, and dependable solu-
tion that promotes tightly controlled coordination and
insulates different activities from each other. Its draw-
back is that the insulation provided by the workspaces
quickly turns into isolation, as developers are not aware
of the activities of others that may affect their work
(Sarma, 2005). Nevertheless, this is the only drawback,
as any formal process is inevitably surrounded by a set
of informal practices according to which the formal
conditions are negotiated and evaluated.

The informal approach can be characterized as inher-
ently user-centric, and the term awareness is defined
as an informal understanding of the activity of others
that provides a context for monitoring and assessing
group and individual activity. The tools that follow
this approach should provide coordination by means of
explicit or implicit dissemination of information to
the other members of the group. It is the responsibil-
ity of the team members to interpret this information
and proactively know how to go about coordination. This
approach also has a drawback: if they are continuously
receiving a lot of information, users are likely to ignore
a lot of it.

As already mentioned, these two approaches have
strengths and weaknesses. This is why the continuous
coordination approach is proposed. The aim of con-
tinuous coordination is to support collaborative work
by combining the strengths of the formal and informal
approaches. The tools that adopt this approach should
be highly flexible instruments capable of adapting the
support for coordination they provide depending on
the needs of the task to be performed.

Booch and Brown (2003) define one of the simplest
and clearest frameworks for classifying collaborative
development environments. CDEs are composed of a
number of tools that really define the features the CDE
supports. These characteristics are, for example, instant
messaging, centralized information management, proj-
et self-administration, and so forth. What the proposed
framework does is to divide these features into three categories of functionalities needed for any CDE. These functions are informally known as the “three Cs,” as they are based on the CDE’s coordination, collaboration, and community building nature. Note that none of these features are on their own particularly complex or difficult to implement, and hence a rich CDE is the emergent creature that rises from a hundred small things, using the Web as the centre of the user experience. Collectively, however, this set of features is complicated to integrate in one and the same environment. Therefore collaborative environment users get the satisfaction of completing a task thanks to the joint use of all these features in the environment.

Sarma (2004; 2005) provides a novel criterion for classifying existing development environments. This is to categorize these environments according to user effort. Effort is defined as the time it takes to configure, monitor and interpret the information the tool offers. Sarma’s framework is based on two key features of collaborative tools: (a) the level of support for coordination offered to users, and (b) the tool’s approach to one of the three key components of collaboration: communication, artifact management, and task management. Combining these two features, the framework uses a pyramid structure to classify CDEs. The level of support for coordination takes up five vertical levels in the pyramid: (1) functional, (2) defined, (3) proactive, (4) passive, and (5) seamless. All the possible tool approaches are arranged as three horizontal levels: (1) communication among team members, (2) artifact management, and (3) task management. Figure 2 illustrates the use of this framework.

From the viewpoint of the layers, the level of support provided by collaborative tools increases as we go up in the pyramid, whereas the user effort it takes to enable collaboration decreases. Tools towards the top of the pyramid provide advanced automated support to users, can handle large and complex team structures, and reduce action and information overload on users, as compared with tools at the lower layers, which allow cooperative development with little technical support.

Development teams at the functional level are small and work with a bare minimum of tool support. Developers can manage to collaborate, but it still takes a lot of manual effort. Tools at this level allow different developers to access the same set of artefacts or communicate via e-mail, but developers at this level are chiefly responsible for the actual coordination activity of determining who changed which artefacts and when. If much more coordination support is needed because there are a great many developers working at the same time on a complex project, we have to move up the pyramid to get more powerful tools that provide full-featured, seamless coordination support that places a minimal burden on the user.

From the viewpoint of the possible tool approaches, those classed into the communication level will help teams to keep each other up to date with the tasks that have been completed, to communicate changes in schedules, to query or provide solutions to problems, to schedule meetings, and fulfill several other purposes. If a CDE does not have good communication features, this often leads to considerable project delays. Tools classed at the artefact management level will be responsible for managing the changes to software artefacts to ensure the
correct program behavior. Artefacts in software systems are highly interdependent on each other. Developers therefore depend on tool support to version their code, coordinate parallel development, resolve conflicts, and integrate code, among other things. Finally, tools within the task management level will help teams to decompose the projects into smaller units, identifying developers with expertise, assigning tasks to developers, and creating a development schedule. In most projects, task management is time-consuming and difficult, and should therefore be highly automated.

ON CLASSIFYING COLLABORATIVE TOOLS: A CASE STUDY

So far we have given an overview of what collaborative tools and development environments are and how they are used. We also illustrated a survey of the most important classification frameworks developed to date. However, this section has a more practical goal. It aims to develop a short case study showing how the four most representative classification frameworks classify different collaborative work tools.

Figure 1 shows the findings of having classed instant messaging, calendar and scheduling, online polling, e-mail, and messaging board according to a number of classification frameworks. Specifically, we have used Grudin’s, Booch’s and Sarma’s frameworks, and continuous coordination.

As we can see, the results of classifying a collaborative tool differ from one framework to another. This suggests that these frameworks should be used jointly to discriminate tools better. Tools that are classed in the same group according to Grudin’s framework, that is the calendar and scheduling and the online polling tools, are clearly differentiated in Booch’s framework. Calendar and scheduling is classed as a coordination tool, whereas online polling is classed as a collaborative tool. Using the continuous coordination framework, the calendar and scheduling tool would fit both formal and informal approaches (i.e., following the continuous coordination model). On the other hand, the online polling tool will be classed as an informal approach. Finally,

Figure 3. Example of coordinated use of a number of frameworks
Characterization and Classification of Collaboration Tools

if we compared both tools using Sarma's framework, they would be classed at the task management level. We should stress that the calendar and scheduling tools would be classed as a basic functional-level tool, except for tools that support event notification services. These tools would be classed several levels higher, as passive level tools. Note, therefore, that the same tool can be classed in more than one specific place within the same framework at a time.

CONCLUSION

The collaborative needs of a team depend largely on environment-related factors, such as the team's organizational structure, its geographical and temporal distribution, the target software domain, the software product structure and the actual team members. Each of these factors highlights a different aspect of collaboration. All existing collaborative development tools have been conceived considering different aspects of collaboration. Each tool meets a particular set of needs and considers the specific functional, organizational, temporal and spatial characteristics of this collaboration. Bearing this in mind, this paper has stressed, based on a survey of existing software products and collaborative sites, that it is unlikely to be possible to build a CDE that meets every possible collaborative need of the software development process, especially since these needs change depending on contextual factors such as the above. A team should know and holistically consider the existing classification frameworks, the most representative ones have been outlined in this chapter, to be able to contextualize the range of available collaborative tools and compare them from different viewpoints. On the basis of the assembled criteria sets, the team will be able to make a grounded decision on what collaborative tools best meet its needs. In this sense, the article has shown the resulting categorization for some of the approaches and has suggested ideas on how to use these frameworks as a holistic means to select the best suite of collaborative tools for a particular work team and development project. We view this approach as a highly influential and important line of future research.

REFERENCES


Characterization and Classification of Collaboration Tools


KEY TERMS

Collaboration: Refers to the different processes wherein people, from small groups to larger collectives and societies, work together, possibly in ubiquitous environments like Internet. A number of useful and effective collaborative environments and methods have emerged from the study of such processes and their distinctive properties.

Collaborative Development Environment: A virtual space wherein all the stakeholders of a project, even if separated by time or distance, may negotiate, communicate, coordinate, brainstorm, discuss, share knowledge, and liaise to carry out some task, most often to create an executable deliverable and its supporting artifacts, holistically integrating multiple collaborative tools and resources.

Computer-Supported Cooperative Work: A field of study addressing the way collaborative activities and their coordination can be supported by means of software and computer systems commonly referred to as groupware, as well as their psychological, social, and organizational effects.

Collaborative Tool: A software module conceived to assure that the people who design, produce, maintain, commercialize and use software are aware of and communicate about the activities of the others simply, efficiently and effectively, also encouraging creativity, driving innovation, and considering software development's social nature.

Coordination: The management of dependencies between activities (generally representing independent subtasks as a result of the division of a cooperative task) and the support of (inter) dependencies among actors involved in carrying them out.

Groupware: Computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment, thanks to the enabling technologies of computer networking, software and services.

Open Source Community: A loosely organized, ad-hoc community of contributors from all over the world who share an interest in meeting a common need, ranging from minor projects to huge developments, which they carry out using a high-performance collaborative development environment (CDE). The concept represents one of the most successful examples of high-performance collaboration and community-building on the Internet.

Software Configuration Management: The discipline of managing the evolution of large and complex software systems.