A Need-Based Collaboration Classification Framework

Anita Sarma, André van der Hoek and Li-Te Cheng

Department of Informatics
Donald Bren School of Info & Comp Sciences
University of California Irvine
Irvine, CA 92697-3425 USA
{asarma, andre}@ics.uci.edu

IBM Research
Collaborative User Experience Group
1 Rogers Street
Cambridge, Massachusetts
li-te_cheng@us.ibm.com

Abstract

Research in collaboration has yielded a large number of tools and environments. A number of classification frameworks exist that organize these contributions, but none of them are comprehensive enough; they focus either on a particular aspect of collaboration or on the specific mechanism that the tools follow. We have developed a new framework that is based on the collaboration needs of a developer. Specifically, we have adapted Maslow's hierarchy of needs to create a hierarchy of collaboration needs in the software development world. These collaboration needs can be broadly classified into *basic needs, enhanced needs and comfort needs*, according to which collaborative tools and environments can be categorized. In this paper, we first introduce the framework, and then use it to identify the collaboration needs that Eclipse and its plug-ins satisfy. We also identify further research directions that would enhance Eclipse's ability as a vehicle for collaboration technology.

1 Introduction

Typical software development is a multi-team effort requiring coordination among developers. It has in fact been shown that about 70 percent of a software engineer's time is spent on cooperative activities [1]. Collaboration is thus at the heart of software development.

There is a considerable body of research relating to collaboration in software development. Ethnographic studies investigating how developers coordinate their activities have provided useful insights that have then been employed to create collaborative tools. This, in turn, has resulted in a host of collaborative tools and environments that support collaboration in one way or another.

To get an insight into the capabilities of the collaboration tools and the coordination problems that they solve, we need a comprehensive classification framework. A number of classification frameworks currently exist. Grudin [2] classifies collaboration tools based on time vs. space: whether a tool supports synchronous or asynchronous coordination vs. whether developers need to be collocated or can be distributed. Malone [3] classifies collaboration tools based on the interdependencies between the coordination process that the tools support (managing shared resources, scheduling tasks, decision support etc.). Nutt [4] proposes a model for workflow systems based on: (1) the amount of conformance that is required by the organization for which the process is a model, (2) the level of detail of description, and (3) the operational nature of the model. The formal vs. informal coordination model [5] classifies tools based on the approach to collaboration. On the one hand, formal process-based approaches attempt to break the entire software development effort into discrete steps and force developers to follow these steps so that there is a specific coordination protocol. On the other hand, informal approaches provide coordination by explicitly or implicitly disseminating information (about the artifact and other developers' activities) to the members of the team. It is the responsibility of the members of the team to agree on their social coordination mechanisms.

Each of the above mentioned frameworks of classification either focuses on a particular aspect of collaboration (an area) or is a framework for

classifying a set of tools that belong to a particular area. These classification frameworks do not provide an overview of all existing approaches to collaboration and are inadequate in providing conceptual guidance to help users choose the right kind of tool. The frameworks that do attempt to address more than one research area (e.g., formal vs. informal approaches) are too coarse grained to help users choose the right approach.

Considerable work has been done in the creation of taxonomies of tools in a variety of research areas [6, 7]. While these taxonomies are helpful when a user needs to choose a particular tool, they do not provide any guidance to comparing tools across different research fields. They are therefore limited as a general classification framework.

We have developed a new framework that classifies different collaboration tools and approaches based on the collaboration needs of developers. Specifically, we have adapted Maslow's hierarchy of needs [8] from the business domain to create a hierarchy of needs for collaboration in software development. Our framework complements existing frameworks and, in fact, ties them together with respect to the collaboration needs that each framework has investigated.

Our hierarchy of needs consists of five layers, increasing support from basic needs, through enhanced needs, to comfort needs. These layers are based on the requirements of collaboration, such as task management, communication, access to a common set of artifacts, to name a few. A particular strength of our framework is that we can classify tools from different research areas as our classification is based on the need that the tool satisfies and not on the approach the tool takes.

To illustrate the use of the framework, we investigate the levels of collaboration needs that Eclipse satisfies via its plugins. Doing so allows us to identify collaboration needs that have not been addressed yet and can be used as guidance for creation of newer collaboration plugins.

The rest of the paper is organized as follows. Section 2 briefly describes our framework. We map Eclipse and its plugins onto the framework in Section 3, and conclude in Section 4.

2 Classification Framework

We have adapted Maslow's hierarchy of needs pyramid [8] to develop our classification framework. Maslow categorized the needs that a person faces in their life into a hierarchy of needs. This hierarchy of needs is in the form of a pyramid and is composed of the following five layers: physiological, safety, love, esteem and self actualization. Each layer represents a set of needs and can be attained only after the needs of the underlying layer are satisfied. For example, only after a person has satisfied their physiological needs, are they able to concentrate on their social needs (the layer above it).

In a similar fashion we categorize the collaboration needs of a developer into a hierarchy and classify the tools based on the collaboration needs that they satisfy. The collaboration needs are broadly classified into *basic needs*, *enhanced needs* and *comfort needs* for collaboration.

Figure 1 shows our classification framework for collaboration tools. The collaboration needs are arranged into a pyramid with five layers (enhanced and comfort needs are split into two layers each). Note that collaboration needs are refined as we progress up the pyramid. As basic needs are satisfied, users require more advanced help from the environment to facilitate collaboration. Note also that each layer in the pyramid is annotated on the right hand side with research areas that have addressed the needs in that layer.

The pyramid consists of three basic strands that make collaborative software development a possibility. These strands are communication, artifact management, and task management. As we progress up the pyramid, the distinction among the strands is blurred, but this is intentional. It represents the insight from the ethnographic studies that users combine different cues and resources from the environment to coordinate their activity. For example, in the higher levels it is possible that the artifacts themselves become the communication medium (e.g., bug reports), or serve as a task management tool (e.g., a requirement specification).

Layer 1 constitutes the *basic needs* of collaboration, without which there can be no cooperative development. This layer specifies that tools need to provide basic facilities such that developers can access a common set of artifacts, communicate with each other, and be able to distribute and recompose tasks.

The *enhanced needs* are split into two layers (layers 2 and 3). Layer 2 denotes needs that are more advanced than the rudimentary collaboration facilities. At this level developers frequently access a common set of artifacts, work in parallel,

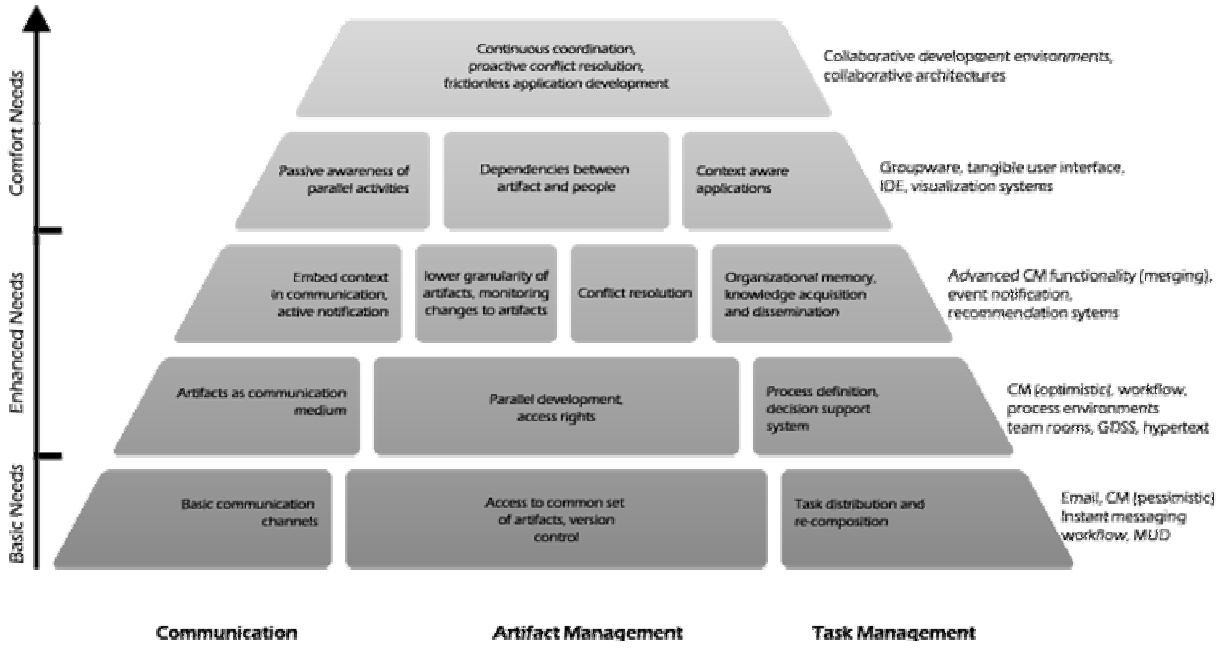

Figure 1. Collaboration Need Hierarchy.

and synchronize their changes, using a predefined coordination protocol.

Tools supporting *enhanced needs* in layer 3 make parallel development easier. Tools can notify developers of parallel activity and ways of resolving conflicting changes. At this level there is refined control over artifacts and context is embedded in communications (e.g., context-specific chats). Developers can use tools that provide organizational help (e.g., expertise locator, recommendation systems) to help them in their development efforts. Most of the functionalities provided by the tools are pull based (developers have to explicitly request information from the tools).

The *comfort needs* of collaboration (layers 4 and 5) address the seamless integration of coordination into software development. Tools in layer 4 allow developers to monitor coordination information without having to switch context from their development environment. It is the responsibility of the tools to present timely and relevant information to the users in a non-obtrusive manner. Developers can thus use this awareness information of parallel changes (and their potential impact) to avoid conflicts, which is a time-consuming and tedious effort.

Tools in layer 5 strive to provide integrated collaboration environments. In essence, collaboration at this level forms a continuum in which coordination information is available to the user at all stages of development, spans across different tool suites, and requires minimal effort from the user. Ultimately, awareness information provided by these kinds of tools and environments must be relevant, peripheral, and concise enough to imitate the way the human brain processes cues from the environment.

Note that, even though our framework classifies collaboration needs into a hierarchy, the framework does not imply that a tool has to build the capabilities of all the underlying layers. A tool can focus on just a particular aspect in the hierarchy and use the underlying infrastructure provided by others. Note also that the top of the pyramid is left open to signify room for future research. We fully expect additional layers to be added as our understanding and available coordination technology matures.

3 Eclipse plugins

To illustrate how our classification framework enables a user to get an insight into the existing tools and environments, we classify the plugins of Eclipse that aid collaboration using this framework. However, classification of all the existing plugins is beyond the scope of this paper,

since there are many hundreds of plugins. Instead, we have chosen a representative set of plugins to demonstrate the usability of our framework.

Plugins that provide email (Nirvana [9]) and chat facilities (Hopy [10], Eclipse Instant messenger [11]) can be categorized in layer 1. Plugins that provide basic Configuration Management facilities (CVS SSL [12], Visual SourceSafe plugins [13]) also fit in this layer, since they are essential for developers to coordinate their development activities.

Plugins for more advanced CM facilities (Spectrum SCM [14], Rational ClearCase [15]) can be classified in layer 2, as they facilitate parallel development. These plugins along with other project management plugins (OpenTime/RC [16], Timer [17]) allow teams to create and track their development process. Bug Tracking systems like CodeBeamer [18] and Jagzilla [19] enable developers to communicate using the bug reports.

Layer 3 of the framework is comprised of plugins like CVSUpdateCheck [20] and Insectivore [21], which notify developers of changes in tasks and projects. Plugins like JReflex [22] and Hipikat [23] can be classified in this category as well, since they provide a form of organizational memory. Specifically, JReflex creates a set of heuristics for understanding at a high-level the nature of the collaboration among the members of the development team and their roles, and Hipikat recommends relevant software development artifacts based on the context in which a developer requests help. Plugins like BoneClipse [24], BranchView [25] provide additional information of the artifacts and their history in the CM repository as graphical displays.

Plugins in layer 4 provide enhanced collaboration features like screen and application sharing (Sangam [26], Collaboration Development Tool Subproject [27]). Plugins like Composonent [28], JAZZ [29] and Palantír [30] provide passive awareness of parallel activities enabling developers to better coordinate their efforts. JAZZ and Palantír further warn developers of potential conflicts that might occur due to conflicting changes in remote workspaces. Developers can thus avoid these potential conflicts by proactively coordinating themselves.

There are currently no existing plugins that can be placed in layer 5. We anticipate there to be plugins that fit into this layer in the future, but a significant hurdle exists. At this layer, much diverging functionality comes together in ways that are unpredictable. Simply integrating previous efforts does not do the trick; as the usage models underneath different approaches vary wildly. Rather, it is necessary to explore different combinations, study usage of these combinations, and slowly but surely come to an understanding of which practices work and which do not. A critical factor is that the three strands of communication, artifact management, and task management must be tightly integrated in a seamless manner, without requiring much user intervention.

In classifying the plugins in the framework, we note that plugin development has followed the hierarchy. Basic features were created first, followed by enhanced and eventually comfort features. This reflects itself in the number of plugins available. The lower layers in the framework contain many more plugins than the upper layers.

Another interesting observation is that new plugins continue to be developed at the lower layers; these tend to provide functionality similar to existing plugins, as most of them are plugins for different but equivalent commercial or open source tools. On the one hand, these plugins are useful such that developers can use their favorite tools within Eclipse. On the other hand, though, this does not advance the research agenda much.

Where research may thrive is at the upper layers, since the research community has not yet focused on the functionality needed there. Some research projects have begun to emerge that attempt to focus on collaboration needs at the top layers, but unfortunately most of these tools tend to build all the functionalities from scratch instead of using the infrastructure already provided by other plugins. Hence, they tend to get stuck at the lower layers.

4 Conclusions

Research in collaboration has resulted in a host of tools and environments. Each tool has typically been built from scratch. We are now at a point in time in which this is no longer possible or needed; we must begin to perform incremental research and use the computing infrastructure already provided by other tools. Eclipse, with its powerful plugin architecture, provides the perfect infrastructure for this.

Before embarking on creating a new collaborative tool, a developer first should: (1) distill the collaboration need they wish to research, (2) investigate other tools in the same category, and (3)

investigate the infrastructure that is already available to them. A comprehensive classification framework based on the collaboration needs that a tool satisfies is indispensable in this process, as the mapping of tools to collaboration needs helps in each of these steps.

Our classification framework is an example of such a need-based framework. Other existing classification frameworks either focus on a particular aspect of collaboration, or on the method that a set of tools follow. These frameworks thus fail to provide an overall picture of the state of research in collaborative development.

With our need-based framework we break the tunnel effect that results from a typical focus on one research area to provide a unified but limited way of looking at coordination at large. We believe our framework can better serve as a guidebook for future collaborative tool development. It is multi-dimensional, examines generic properties, and covers a comprehensive set of needs.

To illustrate our framework we have chosen a representative set Eclipse collaboration plugins and classified them using our framework. While doing this, we realized that there are a lot more plugins that address the needs at the lower level than at the higher levels. Clearly there is much research to be performed at the higher levels, research that must be addressed for collaboration to become an everyday normal activity.

Acknowledgements

Effort funded by the National Science Foundation, grant numbers CCR-0093489 and IIS-0205724, and an Eclipse Innovation Grant (2004).

About the Authors

André van der Hoek is an assistant professor at UC Irvine, Anita Sarma is a Ph.D. student under his supervision. Li-Te Cheng is a researcher in the CUE group at IBM Research, Cambridge.

References

[1] Vessey, I. and A.P. Sravanapudi, *Tools as Collaborative Support Technologies*, in *Communications of the ACM*. 1995. p. 83-95.

[2] Grudin, J., *CSCW: History and Focus.* IEEE Computer, 1994. **27**(5): p. 19-27.

[3] Malone, T.W. and K. Crowston, *The interdisciplinary study of coordination.* ACM Computing Surveys (CSUR), 1994. **26**(1): p. 87-119.

[4] Nutt, G. *The Evolution Towards Flexible Workflow Systems*. in *Dist. Systems Eng.* Dec 1996.

[5] van der Hoek, A.a.e.a. *Continuous Coordination: A New Paradigm for Collaborative Software Engineering Tools*. in *Proceedings of Workshop on WoDISEE*. 2004. Scotland.

[6] Roth, J. *A taxonomy for synchronous groupware architectures*. in *Workshop on Software Architectures for Cooperative Systems*. Dec. 2000. Philadelphia, PA.

[7] Conradi, R. and B. Westfechtel, *Version Models for Software Configuration Management.* ACM Computing Surveys, 1998. **30**(2): p. 232-282.

[8] Maslow, A., *Motivation and Personality,*, H. Row, Editor. 1970.

[9] Nirvana, *http://nirvana.sourceforge.net/*.

[10] Hopy, *http://sourceforge.net/projects/hopy/*.

[11] Messenger, E.I., *http://eimp.sourceforge.net.*

[12] CVS_SSL, *http://home.arcor.de/rolf_wilms/cvsssl-_help.html.*

[13] VSS_Plugin, *http://sourceforge.net/projects/vss-plugin.*

[14] SpectrumSCM, *http://www.spectrumscm.com.*

[15] Clearcase, *http://sourceforge.net/projects/eclipse-ccase.*

[16] OpenTime/RC, *http://www.opnworks.com/-opentime/.*

[17] Plugin, T., *http://trz.turbino.net/.*

[18] CodeBeamer, *http://www.intland.com/products/-codebeamer.htm*.

[19] Jagzilla, *http://jagzilla.sourceforge.net.*

[20] CVSUpdateCheck, *http://cvsupdatecheck.-sourceforge.net/.*

[21] Insectivore, *http://www.zclipse.org/projects/insectivore/.*

[22] Jreflex, *http://www.cs.ualberta.ca/~stroulia/-JRefleX/environment.html.*

[23] Cubranic, Murphy, and Booth. *Hipikat: A Developer's Recommender*. in *OOPSLA*. 2002.

[24] Boneclipse, *http://www.bonevich.com/boneclipse-master/boneclipse-logging/index.html.*

[25] BranchView, *http://andrei.gmxhome.de/perforce-/BranchViewWrapper.html*.

[26] Sangam, *http://sourceforge.net/projects/sangam.*

[27] subproject, C.D., *http://www.scs.carleton.ca/-~deugo/projects/eclipse/.*

[28] Composent, *http://www.eclipseplugincentra-l.com/Web_Links+index-req-viewlink-cid-44.html*.

[29] Cheng, L.-T., et al. *Jazzing up Eclipse with Collaborative Tools*. in *OOPSLA / Eclipse Technology Exchange Workshop*. 2003. Anaheim, CA.

[30] Sarma, A., Z. Noroozi, and A. van der Hoek. *Palantír: Raising Awareness among Configuration Management Workspaces*. in *Twentyfifth International Conference on Software Engineering*. 2003. Portland, Oregon, USA.