Towards a Model of Awareness Support of Software Development in GSD

James Chisan, Daniela Damian
Department of Computer Science
University of Victoria
PO Box 355, Victoria BC V8W 3P6 Canada
{ chisan, danielad } @cs.uvic.ca

Abstract

Awareness is a powerful concept that can be used to enable developers to quickly and easily grasp the state of the workspace which they operate within. We begin by explaining one approach to how awareness might be used to support software development and to enhance developer cooperation and communication. However, since this approach assumes on-going collaboration it is useful to couch the discussion within a collaborative model. This paper presents a model of how awareness could support software collaboration, by describing typical software collaboration, how it is deficient and how awareness helps ameliorates those deficiencies.

Finally, we discuss a variety of issues that become apparent when considering the approach. It is intended that these issues will stir debate and may help generate insight that ultimately improves global software development via awareness support.

1. Introduction

Software development is essentially a collaborative effort among the stakeholders of a project, especially so for those directly involved in development. Business analysts, system analysts, designers, programmers and testers all work together toward the common goal of producing a software solution. Furthermore, they do this in a common workspace comprised by the intermediate artifacts of development: requirements document, design, test scenarios, code, etc. This paper describes a model for how awareness can support collaboration during software development. In global software development (GSD) environments physical separation impairs communication and infringes on the collaborative freedom collocated developments enjoy. Therefore the aim here is to illustrate our vision of how awareness of the workspace might address deficiencies in software collaboration that are exacerbated during GSD.

This paper follows up on work presented last year at the ICSE 2003 GSD workshop (Damian, Chisan, Allen, and Corrie, 2003), where we described how [small] collocated teams benefit from social mechanisms that naturally facilitate work practices and diminish the apparent need for explicit workspace awareness support. To address this need, we propose that when changes are made within the workspace environment particularly to requirements to which much subsequent development depends, developers should be selectively notified about the nature of the change so that changes in requirements are quickly integrated into their work and the development effort on the whole. Requirements are particularly important since it is here that important decisions which directs all subsequent development is, or should be, recorded.

This work describes the first step to implementing that proposal. It seeks to illustrate a model that shows how awareness can support existing collaboration patterns that are typically exhibited during software development. By using the model, shortcomings in collaboration practices that occur during development can be identified. Then, the means in which awareness can support such practices is demonstrated and analyzed. Ultimately this serves to contribute to the larger, primary research goal to improve software development collaboration.

2. Background

Awareness simply refers to knowledge one has of the changing environment which one operates within, essentially 'knowing what is going on' (Endsley, 1995). In software development, this environment is the workspace environment composed of all the intermediate products of development. In particular, with respect to requirements engineering, document change and contact discovery, both facets of the workspace, have been identified as a source of ineffectiveness, confusion and development paralysis (Herbsleb, Mockus, Finholt, and Grinter, 2000). Furthermore, development is hampered by 'organizational amne-

sia' where issues that have already been discussed and resolved are repeatedly reopened for no other reason than their outcomes have been forgotten (Catledge and Potts, 1996). While the current state of the workspace could be ascertained from the contents of the organizational and project documents, these documents have been shown to be a poor communication media (Curtis, Krasner, and Iscoe, 1988; Al-Rawas and Easterbrook, 1995). Likely, as in the case of requirements, formal mechanisms (documents) are not updated quickly enough and, instead, news is propagated informally (Herbsleb *et al.*, 2000). As developers come to depend on informality, it is no wonder there is little motivation to record progress in formal documents in a timely manner.

While there are a variety of tools that implement awareness as an central feature of the system (Jang, Steinfield, and Pfaff, 2000; Bentley, Horstmann, Sikkel, and Trevor, 1995), these systems are not tailored directly to software development. In some cases, even when such tools are used primarily in development support, awareness refers only to notification of change to authors of the artifact (Brush, Bargeron, Grudin, and Gupta, 2002). Such an approach is not sufficient where developers rely on documents they do not author themselves.

In contrast, our proposal is to use the artifacts that exist in the workspace and from their contents build, [semi-] automatically, relationships describing document hierarchies and their authors. This establishes dependence between artifacts (ie. design *x* fulfills requirement *y*) and developer-artifact relationships (ie. *Jim* and *Bob* wrote design *x*). Then, when changes in artifacts in the workspace are detected, authors of dependant documents are notified (ie. *Jim* and *Bob* are notified when requirement *y* changes). This method serves to leverage existing document structures and content to selectively deliver awareness to those who need it.

3. Purpose of the Model

Software development is largely a collaborative task, a result of many different stakeholders working closely together to implement a software solution. The purpose of the model presented in this paper, is to show how awareness fits in to the broader system of development collaboration. This is necessary to articulate, in a structured, detailed way how development practices might be improved with awareness support.

To improve development practices, any method must consider the constraints which limit the possible solutions. In large part, the development habits and processes within organizations are a severe constraint that any approach must consider. Thus the model is used first to describe collaboration patterns that capture the nature of current development practices as a means to show potential insufficiencies in these practices. Second, the model serves as a means to structure where faults occur in collaboration and to analyze the nature of awareness support that might address these faults and strengthen the effectiveness of collaboration. By utilizing this model, we establish a formal descriptive, theoretic framework on which to base further research.

4. The Model

To show how failures in software collaboration might be improved with awareness support, we begin by describing an idealistic model of collaboration and how it is weakened in GSD projects. These weaknesses transcend purely GSD origins, such as impaired communication, lack of informal, impromptu discussions; they also relate to other pressures such as time to market, resource constraints or skill shortage.

To accurately describe typical projects that might involve GSD, a development effort of sufficiently large magnitude must be chosen. Therefore, it is assumed that there are a set of unique stakeholders in the project that include: business interests of (1) the development company and (2) the customer company, (3) end users, (4) system analysts, developers including (5) designers, (6) programmers, (7) testers and (8) documenters. For completeness it is useful to briefly illustrate the roles of these stakeholders.

The business interests of the development company may include marketing tactics, product strategy, future vision, stockholder interests, internal efficiency and policy. The customer company is responsible for the contractual and financial obligations of the software purchase, thus their concerns may be primarily more basic focusing on cost, benefit, product adoption and product support. However, the goal of software is to improve the efficiency and effectiveness of the end user whose satisfaction is based on usability, features, quality and reliability. System analysts are tasked with the responsibility of determining the characteristics (requirements) of the software solution, typically from the above mentioned stakeholders, while being constrained by the limitations of the following development stakeholders. Development consists of designers who elaborate on system requirements and produce detailed technical designs to satisfy software solution, programmers use those designs to produce software that complies with their design, meanwhile testers use requirements and designs to develop, and later execute, test cases. Documenters also use requirements and design to publish user documentation.

As figure 1 illustrates the model shows collaboration paths as work on the software development occurs in clusters of activity each centered on task types within the development. While many interactions occur throughout iterations within the activity (circular), equally important inter-

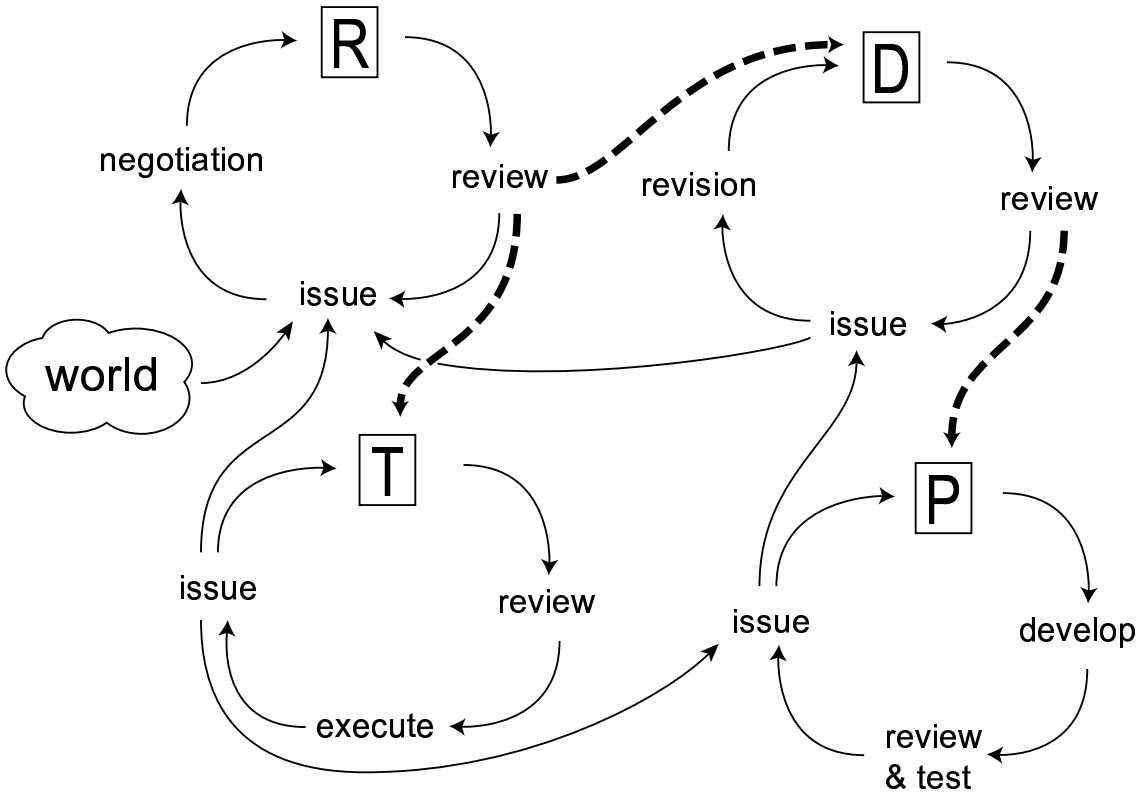

Figure 1. Model of collaboration in software development. Bold, dashed arrows show collaboration paths between development tasks that can be supported with our awareness approach

actions occur between activities. System analysts develop requirements ("R") then review requirements and field concerns about requirements from development and incorporate issues raised by non-development stakeholders, such as customers, users, management or marketing. The requirements activity is concentrated around the task of developing concise, complete requirement documents. Designers take requirements and develop technical designs ("D") then they refine their designs by raising concerns about ambiguous, conflicting or impossible requirements, and by fielding concerns of programmers. The design activity is concentrated around the task of developing complete, logical designs for programmers. Tests also take requirements (and may also use designs) to develop test cases and test scenarios to validate the software ("T") then they refine their tests by raising concerns about questionable requirements and by coordinating with programmers over the execution of their testing procedures. The testing activity is concentrated around the task of developing tests to validate that the software product works and fulfills the software requirements. Programmers take designs and develop the code that becomes the software product ("P") then they progressively refine their code by raising concerns about the design to designers. The programming activity is concentrated around the task of developing functional software that satisfies their designs. In ideal circumstances once the requirements activity has concluded no further events would affect the nature of the requirements, design and test could begin, and then at the conclusion of detailed design, programming could begin and finally the software would be validated with a minimum of activity iteration.

Unfortunately, the model fails to capture the difficulties encountered during real-life software development. In all development the constraint of time bears down most heavily on the life-cycle of the development process. Development that occurs in GSD environments is further handicapped by a lack of timely, lucid communication between developer activities.

When time is constrained, development time is compressed by encouraging each activity phase to begin as soon as possible. In many cases this means that requirements, design, coding and test all start simultaneously. However, the consequence of this philosophy is that interactivity communication is increases, both in frequency and importance.

As requirements become available it is crucial that they are communicated quickly to development and test to minimize the efforts they spend using out-of-date requirements information. (see Figure 1, bold arrows) Likewise, designs need to be propagated quickly to programmers so that they minimize wasted effort. Conversely, issues about requirements need to be recognized quickly by design so that systems analysts can resolve conflicts and ambiguities. During GSD, this scenario is further hindered by slow, sometimes asynchronous communication that is unclear and ambiguous.

Software development failures have often been attributed to requirements error. In particular, captured changes in requirements are not broadcasted to appropriate developers who would otherwise adjust their efforts to reflect such change. In part, this may be attributed to requirements that are only informally captured and are not formally articulated within requirements documents, which become, and remain, habitually out-of-date - making them a redundant waste of effort (Herbsleb *et al.*, 2000).

For example, consider the following frustrating scenario that many industrial practitioners could probably identify with: An unavoidable technical constraint discovered by a programmer causes design to be reworked, designers negotiate an adjustment in requirements with system analysts and modify their designs appropriately. Subsequently, this technical constraint is raised repeatedly by other programmers who were not informed of the design changes (or are using designs that may have been unrelated to the adjusted design, but related to the affected requirement). This chaotic disruption wastes time and effort and causes unnecessary aggravation to the development team.

Clearly this scenario could benefit directly from an awareness of changes in the development workspace that reflect project decisions made by developers. By effectively employing awareness for development artifacts, it is possible to automate some of the communication that occurs between development activities. (see Figure 1, bold arrows). By detecting changes within requirements and automatically notifying relevant designers, testers and programmers, these developer stakeholders can be kept aware of the requirements on which they rely. Furthermore, this promotes development artifacts as first class documents in which to record and organize information. Developers are not interested in wasting their time polling documents for change, but if notified of changes, they are (further) motivated to refer to those documents to determine the nature of those changes. Authors, system analysts in the case of requirements, are motivated to articulate their refinements in the document. Thus, the document itself becomes a medium of communication and developers can begin to rely on their timeliness and currency.

Although this approach does not address the effectiveness of any particular activity, it does suggest that improvements to interdepartmental communication can be realized. For example, awareness does not help the system analyst to capture shifts in markets or abrupt changes business strategy. In contrast, awareness helps the system analyst to communicate these changes (once identified) via the requirements document in a reliable timely fashion to relevant developers. Only then can development be made dynamic and responsive to emergent requirements inherent during iterative, time-constrained development.

5. Issues of the Proposed Solution

In using the model to develop possible awareness solutions to improve collaboration and development during software development a variety of issues become apparent. The issues described below are presented for discussion during the workshop in which this paper is submitted, in the hope that further insight can be exchanged by attendees.

5.1. Extent of Information Dissemination

Of central concern to providing awareness to participants of the development process is striking a balance between providing information germane to their current work responsibilities and limiting extraneous, redundant information that overwhelms developers or desensitizes them to awareness mechanisms. Rather than a scattershot broadcast approach, the analysis of person-artefact relationships makes it possible to provide information to developers in context to their responsibilities and contributions to the workspace.

To maximize the accuracy of notification, person-artifact relationships can be, in many cases, extracted from existing information in the workspace - for example authors of a design could be extracted from the design itself. While this may establish sufficient relationship between developer and artifact additional questions still remain about the extent of awareness required to keep developers up to date.

5.2. Privacy

Software organizations interact in increasingly complex arrangements of acquired, partner and/or outsourcing companies. Data that is [automatically] collected from intermediate artifacts created during GSD can span physical and organizational boundaries. If this information is used for awareness purposes, then this represents a potential for information flow across boundaries, in such cases questions of privacy may arise. For example, an outsource company may want to limit what information is be collected and disseminated to its client (an intimate stakeholder in the project). Even within a single company, some developers may oppose the collection of information that could

reflect on their progress and productivity. This issue can be most closely related to that of personal privacy problems that have been considered with respect to video conferencing (Boyle and Greenberg, 2000) and presence awareness (Godefroid, Herbsleb, Jagadeesany, and Li, 2000), where control over data fidelity is used to maintain presence awareness while preserving personal privacy. This issue differs because it transcends personal privacy to include organizational privacy, and is with respect to the virtual workspace rather than a mere reflection of physical space.

5.3. Delivery

Ideally awareness of the workspace should be as natural as awareness of night and day. The challenge is to minimize the effort and distraction required of developers to keep up-to-date with the goings-on within the workspace. In other words, to provide awareness information as tacitly as possible. Delivery of awareness can vary widely although a few obvious choices include the provision of awareness information in a textual or semi-textual/visual manner via email, the web, instant messenger, or with the use of some specialized application. Management overhead, more stuff to read

5.4. Visualization of Artifact Relationships

Providing awareness requires the establishment of relationships among artifacts and between artifacts and the stakeholders involved in the development. Once this information has been collected it may have significant value on its own as a resource for developers and analysts. Furthermore, these relationships may dramatically enrich awareness events delivered to developers, providing context for the event. The inherent value of this information suggests that it needs to be intuitively accessible to developers. Naturally, we wish to consider how these relationships could be visualized by determining what information developers need to extract and what sorts of questions they may find themselves asking about relationships among artifacts.

6. Future Work

The model presented above is only an intermediate step on the way to achieving the larger research goal of improving software development by providing better support to collaboration in global software teams. This model is a first version based primarily from reports reviewed in available literature, however validation of this model is required. To validate the model we intend to present the model to an industrial partner and survey a sample breadth of stakeholders to critique the model based on their experience. With this input the model will be adjusted to capture those experiences and through this validation process the development of the model will continue to evolve, becoming more refined and more accurately capturing the true nature of collaboration during software development. In the longer run, the model will be used to develop technological or process-based solutions that are specifically designed to deliver awareness to enhance collaboration in GSD. Shortcomings of that solution will also serve to reflect insufficiencies of the model so that the model can be improved.

7. Conclusion

Awareness is a collaborative response to the problem of improving software development practices in GSD. The model described herein serves to show how awareness could be used to improve the naturally collaborative process of software development. Not only could awareness help ameliorate GSD-specific issues, but such solutions also promise to be highly beneficial to co-located development too.

References

A. Al-Rawas and S. Easterbrook. "Communication problems in requirements engineering: A field study." In *First Westminster Conference on Professional Awareness in Software Engineering*. London, 1995.

R. Bentley, T. Horstmann, K. Sikkel, and J. Trevor. "Supporting collaborative information sharing with the WWW: The BSCW shared workspace system." In *Proceedings of the Fourth International WWW Conference*. Boston, MA, 1995.

M. Boyle and S. Greenberg. "Balancing awareness and privacy in a video media space using distortion filtation." In *Proceedings of the Western Computer Graphics Sumposium*. 2000.

A. J. B. Brush, D. Bargeron, J. Grudin, and A. Gupta. "Notification for shared annotation of digital documents." In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press, 2002.

L. Catledge and C. Potts. "Collaboration during conceptual design." In *Proceedings of the 2nd International Conference on Requirements Engineering*. 1996.

B. Curtis, H. Krasner, and N. Iscoe. "A field study of the software design process for large systems." *Communications of the ACM*, 31(11) 1988, pp. 1268–1287.

D. Damian, J. Chisan, P. Allen, and B. Corrie. "Awareness meets requirements management: awareness needs

in global software development." In *International Workshop on Global Software Development (colocated with ICSE '03)*. Portland, OR, 2003.

M. Endsley. "Toward a theory of situation awareness in dynamic systems." *Human Factors*, 37(1) 1995, pp. 65–84.

P. Godefroid, J. D. Herbsleb, L. J. Jagadeesany, and D. Li. "Ensuring privacy in presence awareness: an automated verification approach." In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM Press, 2000.

J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter. "Distance, dependencies, and delay in a global collaboration." In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM Press, 2000.

C. Y. Jang, C. Steinfield, and B. Pfaff. "Supporting awareness among virtual teams in a web-based collaborative system: the teamscope system." *SIGGROUP Bulletin*, 21(3) 2000, pp. 28–34.