Computer Supported Cooperative Work (2009) 18:509–558
DOI 10.1007/s10606-009-9104-0


# Using Developer Activity Data to Enhance Awareness during Collaborative Software Development

Inah Omoronyia, John Ferguson, Marc Roper & Murray Wood
*Department of Computer and Information Sciences, University of Strathclyde, Glasgow G1 1XH, UK (Phone: +44-01415483590 E-mail: inah.omoronyia@cis.strath.ac.uk; E-mail: john.ferguson@cis.strath.ac.uk; E-mail: marc.roper@cis.strath.ac.uk; E-mail: murray.wood@cis.strath.ac.uk)*

**Abstract.** Software development is a global activity unconstrained by the bounds of time and space. A major effect of this increasing scale and distribution is that the shared understanding that developers previously acquired by formal and informal face-to-face meetings is difficult to obtain. This paper proposes a shared awareness model that uses information gathered automatically from developer IDE interactions to make explicit orderings of tasks, artefacts and developers that are relevant to particular work contexts in collaborative, and potentially distributed, software development projects. The research findings suggest that such a model can be used to: identify entities (developers, tasks, artefacts) most associated with a particular work context in a software development project; identify relevance relationships amongst tasks, developers and artefacts e.g. which developers and artefacts are currently most relevant to a task or which developers have contributed to a task over time; and, can be used to identify potential bottlenecks in a project through a 'social graph' view. Furthermore, this awareness information is captured and provided as developers work in different locations and at different times.

**Keywords:** context awareness, collaboration, relevance filtering, distributed teamwork, empirical studies, global software development

## 1. Introduction

Software development is a collaborative effort where groups of developers work together within a global time/space matrix. During such collaboration developers need to maintain their awareness of how particular tasks or project artefacts are progressing, what fellow developers are (or have been) doing and the current state of resources associated with the project. In collocated settings the awareness information that concerns developers directly or tangentially is achieved through the use of instant messaging clients, emails, scrum meetings and developers stopping at the desks of co-workers to update them on problems or to see what

Inah Omoronyia moved to the Norwegian University of Science and Technology (NTNU), Trondheim, Norway during the course of this publication.

problems they are facing (Ko et al. 2007). Developing and maintaining such awareness is more difficult in distributed software teams than collocated ones (Cramton 2001). This is because the awareness information required in such settings is tacit, inherent, dynamic and contextual. It is tacit since most of what developers do in collaboration spaces builds from experience, skills, heuristics and interactions that can hardly be documented (Busch and Richards 2001; Hadas and Frank 2001), and inherent since this knowledge is deeply bound to these developers. Its dynamic nature stems from the ever changing state of software projects. Finally, the relevance of such information varies across differing project contexts.

A number of studies have revealed the problems caused by these peculiar attributes of distributed teams. They include poor visibility and control of remote resources; inadequate communication, collaboration and coordination across distributed teams; diminishing trust; and lack of shared contextual awareness (Boland and Fitzgerald 2004; Chisan and Damian 2004; Hargreaves and Damian 2004). An industrial experience report on distributed software teams located over ten sites identified shared contextual awareness of the work carried out by different team members as a major issue (Kommeren and Parviainen 2007). Herbsleb (2007) has suggested that this lack of contextual awareness information makes it difficult to initiate contact and often leads to a misunderstanding of communication content and motivation. The lack of context information limits the potential to track the effect of changes in distributed, collaboration space (Kommeren and Parviainen 2007).

On the other hand, distributed software development offers a number of theoretical benefits, including shortened time-to-market cycles, more rapid response to customer needs and a more effective resource pooling (Kommeren and Parviainen 2007). The goal of the research reported in this paper is to bridge the gap between the reality of distributed software development and these theoretical benefits by developing awareness systems that emulate collocation in distributed settings.

It is proposed that the benefits of collocation in virtual and distributed collaboration spaces can be achieved by capturing the interaction activity trails that occur within these spaces. These trails are built up as developers go about their daily development tasks leaving historical traces behind. An empirical study carried out by Fritz et al. suggests that these developer interactions can be used to build models of awareness about a software code base (Fritz et al. 2007).

This paper starts by discussing related literature on awareness in general settings before focussing on previous research that has aimed to provide support for increased awareness in a collaborative and/or distributed software engineering environment. From this review it becomes clear that awareness information needs, and the mechanisms for dissemination in software development teams, are more easily obtained in collocated than distributed scenarios. To achieve the potential benefits of distributed, collaborative development further work is required that focuses attention on who and what is relevant to particular work contexts within a software project. In contrast to previous work, the approach proposed here aims to build an awareness

model without relying on developers 'tagging' particular artefacts or on the limitations of the underlying configuration management system.

Based on the literature review, this research introduces a 'Continuum of Relevance Index' (CRI) model that proposes a new approach to providing relevance awareness information. CRI is derived from developer interactions in a shared collaboration space. The basis of this model is the monitoring of key interactions, such as project views, updates and creates, made by any developer while working in a distributed, collaborative space. The model is used to provide real time relevance rankings that are intended to enhance awareness of the relevance of tasks, developers and artefacts to a selected development work context. The research question that is addressed is: Can a model based on real-time monitoring of IDE interactions, such as creates, edits and views, enhance contextual awareness during distributed, collaborative software development? A qualitative investigation using a prototype implementation of CRI has been carried out using advanced student based collaborative projects. Results of the study demonstrate that the model can provide accurate relevance rankings, and thereby has the potential to increase developer awareness of artefacts, developers and tasks, and their interrelationships, over a range of collaborative project contexts.

## 2. Review of literature and problem formulation

### 2.1. Awareness concepts within the framework of collaborative work

Research literature on the concept of awareness in a general setting suggests that it is both situation and domain dependant with no single meaning. As Schmidt (2002) says, "The very word 'awareness' is one of those highly elastic English words that can be used to mean a host of different things. Depending on the context it may mean anything from consciousness to knowledge to attention or sentience, and from sensitivity or apperception to acquaintance or recollection". Dourish and Bellott (1992) define awareness as understanding the activities of others, providing a context for your own activity, while Gutwin and Greenberg (1998) present awareness as a mechanism for enhancing coordination and efficiency when people work together. Schmidt (2002) defines awareness as an attribute of action, doing it "heedfully, competently, mindfully, accountably". Whilst acknowledging that there are other forms of awareness that exist, the following five awareness types concern the needs of group work dynamics that exist during collaboration:

- Informal awareness is associated with a pervasive sense of who is around, what they are doing, and what they are going to do. People have this kind of knowledge when they work together in the same office. Informal awareness can been used to facilitate casual interactions and initiate appropriate modes of communication (Gross et al. 2005).
- Group-structural awareness constitutes information about group members such as their roles, status and position on certain issues (Gutwin 1997). Group-structural awareness is essential to obtain knowledge of the expertise of other collaborators

based on the roles they assume. This knowledge can prove important in choosing who to initiate an interaction with for mentoring on project activities.

– Social awareness is the type of information collaborators have about each other in a conversational or task context and includes information such as the attention, interest and emotional state of collaborators (Gross et al. 2005). Providing such awareness information helps minimise interruptions and disturbances when engaging in collaborative processes, or rather 'appropriate obtrusiveness', as described by Schmidt (2002).
– Workspace awareness concerns information about the interactions of other collaborators with a shared project workspace and the artefacts it contains. Gutwin et al. (1996) described a set of elements that collaborators may keep track of during a collaborative process in a shared space and the relevant questions associated with these elements—see Table 1.

Context awareness is a more generic concept in which context refers to the set of circumstances or facts that surround a particular event or situation (Webster 2006). From a computer science perspective context awareness was initially perceived as referring to the location of an entity (Dey et al. 2001). The notion has now evolved to not just a location but part of a process with different state transitions (Bolchini et al. 2007).

Each of these five awareness types is associated with the notion of context. For instance, informal awareness provides information on the presence and location of collaborators and is therefore highly dynamic. Similar observations can be made about the changing states of group-structural, social and workspace awareness. Furthermore, since collaborators work on different tasks and different resources and form different perceptions of their workspace, such awareness is highly contextual and therefore cannot be generalised.

*Table 1.* Elements of workspace awareness (Gutwin et al. 1996).

| Element | Relevant questions |
|---|---|
| Identity | Who is participating in the activity? |
| Location | Where are they? |
| Activity level | Are they active in the workspace? |
| Actions | What are they doing? |
| Intentions | What are they going to do? |
| | Where are they going to be? |
| Changes | What changes are they making? |
| | Where are the changes being made? |
| Objects | What objects are they using? |
| Extents | What can they see? |
| Abilities | What can they do? |
| Sphere of Influence | Where can they have effects? |
| Expectations | What do they need me to do? |

In the remainder of this paper, contextual awareness is implied each time the word awareness is used.

### 2.2. Disseminating awareness information in computer mediated collaboration

Insight on how awareness is supported in electronic and virtual shared workspaces is increasingly important, especially in scenarios where time and space are parameters for defining a collaboration process. Researchers have pursued a variety of strategies to keep collaborators aware of important information. Cadiz et al. (2001) suggested that these strategies generally fall into one of three categories: polling, alerts and peripheral awareness.

Polling involves making information accessible and allows collaborators to repeatedly check, or 'poll', the information. This is advantageous when an individual has knowledge of where to repeatedly check for updates on information. It is also an appropriate mechanism to disseminate awareness when such information is required on an 'as needed' basis.

Alerts involve intentionally interrupting an individual to provide awareness information and overcome the drawback associated with polling of only finding information when they poll the information source. Alerts can be delivered via audio or visual cues and can range from highly to minimally intrusive, utilising intelligent algorithms to determine if the cost of interruption is worth the benefit (Horvitz et al. 1999). The main disadvantage of this approach is that alerts often do disrupt users from their primary task (Cutrell et al. 2001; McFarlane 1999).

Peripheral awareness works by filling a user's peripheral attention with information such that it envelops them without distracting them. The goal is to present the information so that it works its way into a user's mind without intentional interruption. Peripheral forms of disseminating awareness have been provided using peripheral audio (Alexanderson 2004; Pacey and MacGregor 2001), peripheral vision or a mix of peripheral visual and audio cues (Cadiz et al. 2001; Heiner et al. 1999; Weiser and Brown 1996). The disadvantage of peripheral awareness is that it is possible to ignore important information that appears only at the periphery. Furthermore, the challenge is to figure out what should be presented peripherally, and to strike a balance between too much and too little (Pedersen 1998).

Kantor and Redmiles (2001) argue that there is no optimal strategy to provide awareness information in collaboration spaces because of the associated advantages and disadvantages. For example, if one author is changing a document, a co-author and an end user of the document are likely to benefit from different awareness strategies.

### 2.3. Awareness information needs and dissemination in software development teams

The awareness requirements amongst collaborating software developers generally focus on the need for information on people, project resources and development

tasks. As such, they have much in common with the general awareness information needs over shared workspaces identified by Gutwin et al. (1996) as shown in Table 1. A study conducted by Ko et al. (2007) showed that the most frequently sought information during software development included awareness about tasks, artefacts and co-workers. The study also suggested that developers frequently sought information about how the resources they depended on changed; what their fellow team-mates had been doing; and what information was relevant to their task. These awareness information requirements focus on two broad perspectives. The *historical perspective* is where information is sought about the overall impact of project entities on a selected work process (e.g. developers seeking information on which artefacts are relevant to accomplish a selected task). The *recent perspective* is where information is sought about the current state of entities in the workspace (e.g. what team-mates are doing, or recent changes on artefacts that developers depend on).

In collocated teams such awareness information has been disseminated via polling and alert mechanisms such as email and instant messaging clients. Development teams also use frequent, brief meetings throughout the day to stay aware of work effort and problems. Also, developers will stop by a co-worker's office, chat in the hallway or over coffee to update them on problems or see what problems they are facing (Curtis et al. 1988). Research studies demonstrate that the social nature of software development work is also driven by awareness information. For instance, Perry et al. (1994) reported that over half of developers' time was spent interacting with team members, of which much of the communication was to maintain awareness. Ko et al. (2007) also highlighted that co-workers were the most frequent resource when seeking information. Furthermore, simply watching another developer carry out a task (Segal 1995), and observing changes to project artefacts (Dix et al. 2004) have also been used as a source of awareness in collocated software development.

Awareness information needs in *distributed* software development teams are not significantly different from collocated needs. In distributed teams developers also seek to maintain awareness of other developers activities including the code artefacts and the tasks they are working on. Gutwin et al. (2004) found that distributed developers sought to maintain a broad awareness of who were the main people working on their project and what their expertise was.

For distributed teams, however, obtaining and disseminating awareness information is more challenging and mostly dependent on electronic means. Studies carried out on open source projects showed that awareness was mainly maintained using text-based communication such as mailing lists and instant messaging clients (Gutwin et al. 2004). The main advantage of text-based channels stems from their simplicity of use, but they depend on commitment from developers to read the shared text and making their project communications public (Biehl et al. 2007).

Version check-in logs in configuration management systems have also been used to obtain awareness of work in both distributed and collocated software

development. de Souza et al. (2003) showed that collocated developers often gauge expertise by inspecting check-in logs. Similarly, Gutwin et al. (2004) reported that distributed developers obtained awareness through check-in logs, with changes being automatically sent to a mailing list of subscribed developers. The advantage of check-in logs is that they are based on the actual manipulation of project artefacts.

However, awareness information from check-in logs is not always sufficient. While these logs can help find other developers, they generally do not distinguish levels of expertise (McDonald and Ackerman 1998)—though Expertise Browser (Mockus and Herbsleb 2002) would appear to be an exception to this. This becomes more complicated when an artefact has been worked on by several developers with differing levels of expertise. While strict partitioning and code ownership rules may be enforced to help avoid this problem, in distributed software such as open source development, code ownership is not always obtainable. Gutwin et al. showed that partitioning was not so strongly applied on open source projects and developers were free to work where they saw fit (Gutwin et al. 2004). Study findings have also shown the lack of clear ownership and partitioning in distributed, open source projects and concluded that the structure of a project alone is not sufficient for developers to obtain the awareness necessary to coordinate their actions (Mockus et al. 2002).

The closest imitation of collocation during distributed software development has been through the use of media spaces. A typical media space consists of permanent video and audio connections between geographically distributed sites. The permanent connection of media spaces has been shown to reduce the cost of initiating collaboration and to contribute to the creation of a common social space irrespective of distance (Bly et al. 1993; Farshchian 2001; Singh 1999). While media spaces cannot replace face-to-face awareness they do provide an opportunity to obtain awareness information that is not normally possible without 'being there' (Bly et al. 1993). The use of video conferencing, video phone and desktop video with audio capabilities in distributed development have also been studied by de Freitas et al. (2008). However, the use of such synchronous channels is highly challenging for distribution that is characterised by different time zones. Furthermore, this mode of awareness lacks the flexibility of mechanisms such as email and check-in logs, where information can be more easily searched and referred to at convenience.

On the whole, these studies suggest that awareness support for distributed software development teams is still inadequate from both a historical and a recent perspective. Empirical studies have revealed that most tools are designed to answer a specific kind of question, focussed on a particular type of code artefact (Sillito et al. 2008). Also, most approaches treat information seeking questions as if they were asked in isolation rather than part of an ongoing dialogue which can be necessary to obtain full contextual awareness during distributed software development. Finally, it is clear that collocated software development has

awareness information benefits that are more difficult to obtain during distributed development. As a result, collocated teams are likely to achieve higher productivity, shorter schedules, and higher satisfaction among stakeholders (Teasley et al. 2002). The goal is therefore to build tools for distributed teams that emulate the attributes of collocation awareness in their design.

## 2.4. Enhancing context awareness in distributed software development environments

Current software development environments (IDEs), such as Eclipse, NetBeans, and Visual Studio, are enhanced with facilities to make software development easier. These include source code editors, compilers, interpreters, debuggers, visualisations and code generators. The size, complexity and distributed nature of current projects also bring a demand for further features to support development. The following discusses a range of approaches that have been used to enhance contextual awareness during distributed software development within such IDEs.

### 2.4.1. *Obtaining context by social tagging*

A tag is a keyword assigned to a piece of information to help describe it. Social tagging describes the collaborative activity of marking shared content to organize it for future navigation, filtering or search (Yew et al. 2006). This concept has been introduced into a number of IDE components to enhance contextual awareness during distributed, collaborative software development. Storey et al. (2006) presented TagSEA (Tags for Software Engineering Activities in Eclipse) based on the concept of waypoints (locations of interest) and social tagging (social bookmarking). The waypoint analogy corresponds to marking specific locations in the software such as Java source code elements (classes, methods, packages etc.). User-created annotations, written as comments embedded in the code, result in very explicit landmarks for readers and support navigation and coordination. While preliminary feedback suggests that implicitly captured meta-data combined with the lightweight nature of tagging is a promising technique for supporting contextual awareness in distributed software development, it can become unwieldy in practice and outdated over time. The concept of social tagging can also be found in Jazz,[1] which is a real-time team collaboration platform based on the Eclipse IDE for integrating work across the different phases of a software development lifecycle. One of the aims of Jazz is to introduce contextual awareness into the collaboration environment (Hupfer et al. 2004). With Jazz developers can initiate chats, which can then be saved as code annotations on the section of the code artefact involved in the discussion (Cheng et al. 2004).

The use of tagging can also be found in CASS (Cross Application Subscription Services)—a software development awareness infrastructure (Kantor and Redmiles 2001). CASS provides a notification server for the distribution of

awareness information which enables developers to subscribe to types of information that they believe will affect them and to specify which types of awareness tool the information should be sent through (email etc.). The outcome of this configuration is that developers get contextual awareness information on the state of different aspects of a software system that directly affects their work. The main challenge with using CASS appears to lie with the configuration that needs to be carried out by the developer to get the tool running. This might be asking a lot from a developer who is not familiar with the development space.

Finally, Froehlich and Dourish (2004) presented Augur as a visualisation tool that provides a line-oriented view for supporting distributed software development processes. These views are formed by tagging developers to different aspects of an artefact they have been associated with. Initial evaluation of Augur with open source software developers suggests that generating views based on tagging the activities of developers to subsections of artefacts is both meaningful and valuable to software developers.

#### 2.4.2. *Obtaining context by mining relational properties among software project entities*

During software development a wide variety of relationships are formed. These relationships can be structural relations, based on direct and indirect links amongst artefacts that comprise a project, or they can be social relations based on direct and indirect links among developers collaborating on a project. A hybrid of these can also be obtained, based on associations between developers and artefacts that are associated with a shared software project. The use of such project structural and social relations has also been modelled by context awareness mechanisms within IDEs.

The Rational Team Concert (RTC), a plug-in to Jazz (Jazz 2008), enables contextual awareness by mining relational properties of entities within shared software projects. Each 'Project Area' contains the artefacts for a project and has an associated process which governs how the project is run and the way Jazz behaves. Project areas are decomposed into a set of 'Team Areas', which describe the teams that work on the project. Each team area has a list of team members and the 'Process Role' they play within the team. A user can be a member of more than one team. Each team area can define 'Process Customizations' of the process to tailor Jazz for the team and its sub-teams. Finally, the planned work is described by 'Work Items'. The types of work items used in a project area are defined by the process (Jazz 2008). This rigorous relational view offers the potential to enhance traceability and contextual awareness of the state of different entities within a project. For instance, information about the state of a work item can be derived by viewing the code artefact resources and the developers that the work item is associated with. Similarly, contextual information about a project team can be derived by navigating the different team members and processes associated with the project team. The potential downside of RTC is that each of

these entities and their inter-relations has to be defined by the user-for non-trivial projects this could be quite demanding. It is also worth noting that, during evaluation, participants expressed some concern that Jazz might be used by unethical managers to monitor their work instead of being used as a coordination aid to enhance awareness (Cheng et al. 2004).

The work of de Souza et al. (2007) in developing Ariadne also demonstrated the use of relational properties among entities to derive contextual awareness within IDEs. Ariadne, another plug-in for the Eclipse IDE, analyses software projects for dependencies, whilst collecting authorship information from configuration management repositories. The tool translates technical dependencies (e.g. call graphs) among components into social dependencies among developers (by annotating components with social information) and creates a visualisation to convey this information. Ariadne is used to identify developers who are more likely to be communicating, by assuming that developers with similar dependencies are likely collaborators. The accuracy of Ariadne is therefore dependent on the state of a versioning system.

A number of other tools and models have been developed to enhance contextual awareness in distributed software development based on mining relational properties that exist amongst software project entities. Bruegge et al. (2006) presented Sysiphus as a tool that supports the creation and subsequent browsing of a graph created by linking artefacts, as well as annotations and comments on those artefacts. It achieves this by encouraging collaborators to make communication and issues (tasks) explicit in the context of system models and also to become aware of relevant stakeholders. Cubranic et al. (2005) describes the design and evaluation of Hipikat, a tool that draws on information retrieval techniques to help developers identify artefacts that are related to an initial artefact used to generate a query. Evaluation shows that the tool finds useful starting points for exploring the code. Finally, Expertise Browser (Mockus and Herbsleb 2002) uses data from version control systems to locate developers with desired expertise in geographically distributed software development projects. Expertise is automatically constructed from 'experience atoms' which correspond to individual revision control changes (deltas). Evaluation of Expertise Browser showed that newer and remote development sites tended to use the tool to find individuals with particular expertise while larger, more established sites used the tool to discover the particular expertise held by individuals or organisations.

### 2.4.3. *Obtaining context by monitoring developer interactions*

Interactions that are carried out in collaboration space can be viewed as having different levels of impact on the state of a project. By associating different weightings to interactions based on their perceived levels of impact on the state of a shared project it is possible to obtain contextual awareness.

This concept of weighting the severity of developer interactions can be seen in the modelling of Palantír (Sarma and Hoek 2002; Sarma et al. 2003). Palantír is a

workspace awareness tool that complements configuration management systems. It enhances awareness by continuously sharing information regarding operations performed by all developers. The tool specifically informs a developer which other developers change which other artefacts. Furthermore, Palantír provides a measure of the severity of those changes (based on the proportion of the file that has changed) and graphically displays this information in a configurable manner. The accuracy of Palantír is highly dependent on consistent use of a version management system by collaborating developers.

FASTDash (Fostering Awareness for Software Teams Dashboard) (Biehl et al. 2007) is a visualisation tool that highlights the current activity of team members, such as which files are changing, who is changing them and how they are being used. The visualisations can also be annotated, allowing members to supplement context information with status details. FASTDash is optimised for developers working in close proximity and time and may be less useful across different time zones or in cases where collaborators are given the freedom of choice of place and time of work.

#### 2.4.4. *Obtaining context by combining developer interactions and relational properties*

The Team Tracks project (Deline et al. 2005) utilises the notion of relational property and frequency of an interaction. Team Tracks helps developers understand unfamiliar source code by mining navigation data as development teams go about their daily programming activities. Team Tracks is based on two insights: the more often developers visit a part of the code the more important it is; and the more often developers visit two parts of the code in succession the more related they are. To help a newcomer quickly find the most important parts of the code, Team Tracks limits the code overviews to the most frequently visited items (favourite classes view). To help a newcomer find code related to the module currently being worked upon Team Tracks recommends parts of the code visited just before or after that module (related items view). A controlled laboratory study has shown that Team Tracks significantly improves a developer's ability to perform updates to unfamiliar code.

#### 2.4.5. *Obtaining context by combining developer interactions and relational properties with the notion of time and its expiration*

This notion of context formation can be seen in the modelling of Mylyn (Kersten 2007) as a task-focused interface for Eclipse. The main objective of Mylyn is to reduce information overload and make multi-tasking easier. Mylyn monitors a developer's work activity to identify code artefacts relevant to the task in-hand; it then uses the task context to focus the Eclipse user interface on relevant artefacts. As the interaction history is captured from a developer's activity, a degree of interest (DOI) function assigns real number weightings of artefacts to tasks. The weighting is based on the frequency of access to the artefact and a *decay* factor that corresponds to the total number of interaction events captured. Accessing an

artefact in the context of a task increases its weight, while accessing other artefacts decays the weight of infrequently accessed artefacts.

The weight associated with an accessed artefact, or decrease in weight of other artefacts, is determined by the interaction events monitored by Mylyn as shown in Table 2. Mylyn also considers events that do not directly affect the state of a code artefact. For instance, a command event, such as a preference setting or a save button press, increases the relevance of an active artefact to the current task. An artefact is active if it is open within the tooling environment while the command event is being executed. Propagation and prediction events cause artefacts that have not yet been interacted with directly to be associated with a task context. For example, a selection event may trigger a propagation event for structurally related elements such as sub classes and the package containing the class. Value ranges in the DOI specify which artefacts are relevant to a task. Relevant artefacts are those with a positive DOI value. Empirical evaluation of Mylyn, in which professional programmers used the tool for their daily work on enterprise-scale Java systems, showed developers spending more time working on code than navigating it (as opposed to the other way round).

### 2.5. Recommender systems for general task awareness

There is a body of related research, outside the specific domain of software engineering that uses computer interaction data to automatically identify resources that are relevant to tasks. Typical of this work is that of Dragunov et al. (2005) on TaskTracer and Kaptelinin (2003) on UMEA.

These systems aim to identify the resources that are relevant to user defined tasks. Typically, they operate in Microsoft Windows environments using COM extensions to monitor and capture events associated with a range of tools e.g. word processing, spreadsheets, databases, web browsers and email clients. The goal of these systems is to be able to automatically identify the resources (and processes) that are relevant to tasks such as writing a report, so that if a user is interrupted, temporarily switches task, or comes back to a similar project at later date, these tools can quickly recover the resources that are relevant to that task.

*Table 2.* Interaction events monitored by Mylyn (Kersten 2007).

| Event kind | Interaction | Description |
|---|---|---|
| Selection | Direct | Editor and view selection via mouse or keyboard |
| Edit | | Textual and graphical edits |
| Command | | Operations such as saving, building, preference setting and interest manipulation |
| Propagation | Indirect | Interaction propagates to structurally related elements |
| Prediction | | Capture of potential future interaction events |

These general task recommender systems and approaches have some similarities to the model, and its implementation, described in this paper. They are based on the monitoring of computer interaction data. They face the common, major challenges of identifying the task that a user is working on, and detecting when the user switches task. Typically the onus is on the user to identify both of these, and they have to deal with noisy data. Often users will temporally change the focus of their work without switching task.

However, there are also major differences between the goals of these more general systems and a more specific distributed, collaborative software engineering model. Both the TaskTracer and UMEA systems appear only to present resource context from the perspective of an individual user and individual tasks—these recommender systems identify the resources that are relevant to a specific user addressing a specific task. In collaborative software engineering the aim is to provide awareness across multiple, *collaborating* users (developers), tasks and resources (software artefacts) from multiple perspectives. Therefore, as well as discovering what artefacts are relevant to a specific individual performing a particular task, the model should support discovering what tasks and developers are relevant to identified artefacts, what developers have contributed to a task etc. To address the general problem of noise, caused by temporary changes of context (e.g. answering an email) or mistaken interaction (e.g. opening the wrong file) there is a need for an underlying weighting system, or relevance model, that disregards spurious interactions and rewards repetitive interactions, thereby offering the potential to rank entities in terms of their relevance.

### 2.6. Research motivation

This review has centred on contextual awareness and dissemination mechanisms in the form of tools that can be used to enhance contextual awareness during distributed, collaborative software development. The initial insight obtained from this review is that awareness information needs, and the mechanisms for dissemination in software development teams, are more easily obtained in collocated than distributed scenarios. While distributed teams also have potential advantages, to fully achieve these requires that more research is carried out on the modelling and dissemination of contextual awareness.

Table 3 is a general classification of the reviewed systems based on the identified elements of workspace awareness initially described by Gutwin et al. (1996) and shown in Table 1. The classification demonstrates that the majority of systems enable collaborators to identity who is participating in an activity, the changes they have made, and the objects used. Identifying the current location, current actions, activity levels, extents, abilities, sphere of influence and expectations of collaborators within the workspace are less supported. Tools that are characterised by support for intention are mostly tag based, and rely on developers to explicitly state their intention within the work space.

*Table 3.* Classification of systems based on Gutwin et al.'s (1996) elements of workspace awareness.

| Element | TagSEA | Jazz | Expertise Browser | Sysiphus | Hipikat | Palantír | FASTDash | Team Tracks | CASS | Augur | Ariadne | Mylyn |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Identity | × | × | × | × | × | × | × | | × | × | × | |
| Location | | | × | × | | | × | | | | | × |
| Activity level | | | | | | × | | × | | | | × |
| Actions | | | | | | | × | | | | | × |
| Intentions | × | × | | × | | | | | × | × | | |
| Changes | × | × | × | × | | × | × | | × | × | | × |
| Objects | × | × | | | | × | × | | × | × | | × |
| Extents | | × | | | | × | | | | | | |
| Abilities | | | × | | | | | | | | × | |
| Sphere of Influence | | × | × | | | | | × | | | | |
| Expectations | | | | | × | | | × | | | | × |

The approach of systems such as TagSEA, CASS, Augur and some aspects of Jazz depend on social tagging of different aspects of a software system by developers. Such systems are not based on the physical manipulation of code and are thus subject to a level of potential misrepresentation of context. Also, tag based systems can become outdated over time, especially when developers fail to update tags as aspects of the system change.

An alternative approach is adopted in systems like Sysiphus, Ariadne, Hipikat and Rational Team Concert which obtain context by mining relational properties among software project entities. These have recorded a number of successes in generating and representing contextual awareness information, but challenges may arise as the relational dependencies amongst entities become increasingly complex. Furthermore, when abstract models such as use cases or bug definitions are used as the basis of task definition, these systems only focus on concrete artefacts and do not consider the relationships between the more abstract models and concrete artefacts. Similar limitations exist in systems such as Palantír which is based on monitoring the activity captured by configuration management systems during check-in and check-out. In addition, FASTDash appears to be intended for developers working in relatively close proximity and not separated in time. Expertise Browser addresses some of the key goals by automatically generating rankings of developer expertise related to code artefacts by monitoring version control systems, but this approach omits potentially relevant activity such as when a developer views an artefact and also includes no notion of task awareness.

Team Tracks derives contextual awareness by articulating entity dependencies and weighting the severity of developer interaction on an underlying relation. Team Tracks is limited in the nature and amount of dependencies it represents and because the weighted severity of dependencies is based only on view interaction events (rather than edits etc.). In contrast, Mylyn's relative strength of dependency is determined by a degree of interest function based on frequency of access to entities and the nature of interaction being performed by a developer. However, Mylyn does not currently focus on *distributed* software development. Its notion of task context for collaborative software development is analogous to passing tokens of context generated by one developer to another to continue building upon. While it is useful and interesting to obtain awareness of the impact of a selected task on the state of a code artefact (as demonstrated in Mylyn), it is potentially more useful for enhanced coordination to obtain awareness of the relative impact of *every* task and *all* associated developers that have affected the state of the artefact.

Central to this paper is the observation that relevance relations amongst entities in a collaboration space are asymmetric. For example, assuming a task instance $T_x$ is achieved using artefacts $A_1$, $A_2$...$A_n$, it cannot be assumed that the relevance of $A_n$ to $T_x$ is the same as the relevance of $T_x$ to $A_n$. This is because relevance is context sensitive—the relative relevance of an artefact to a task depends on the

other artefacts associated with that task, whereas the relative relevance of a task to an artefact depends on the other tasks to which the artefact is relevant.

The main novelty of the approach proposed in this paper compared to the previous work described above is a model that provides a perception of the relevance and impact of tasks, developers and artefacts associated with a distributed software project in a *selected work context*. For a selected task instance, awareness is provided of the relative impact of project developers and code artefacts. For a selected code artefact, awareness is provided of the relative impact of project tasks and developers on the state of the code artefact. Similarly, for a selected developer, awareness is provided of the relative impact of tasks and code artefacts on the work context of the developer. The provision of such awareness is independent of configuration management systems or the need for tagging, and is from a collaborative perspective rather than that of an individual developer. Furthermore, the provision of this contextual awareness is not limited by time or space and continually changes to keep pace with software project dynamics.

## 3. Example software development scenario

The following is an example used to motivate the research and, later in the paper, explain the functionality in the proposed model and its implementation.

Bill, Amy and Ruben are members of a team collaborating to develop an online cinema ticketing system called TickX. There are two front-end use cases required to accomplish TickX: Purchase Tickets and Browse Movies. (Here use cases are viewed as a structure for the definition and assignment of tasks. In addition, there will be some use cases for system administrators which are not included here.) A number of code artefacts are being developed to realise TickX and include Ticket.java, Customer.java, Account.java, Booking.java, Movie.java, MovieCatalog.java and Cinema.java. A class diagram for TickX is as shown in Figure 1.

While Amy and Bill have been collaborating to implement the Purchase Tickets tasks/use cases, Ruben has been responsible for Browse Movies. The following interaction trails take place as these collaborators go about their allocated tasks:

- While Amy was working on Purchase Tickets she created and updated the Account.java and Customer.java code artefacts. She viewed and updated Booking.java a number of times. She also viewed MovieCatalog.java and Cinema.java.
- In the initial phase of Bill's collaboration on Purchase Tickets, he viewed the Account.java and MovieCatalog.java code artefacts. This was subsequently followed by his creation and update of the Ticket.java and Booking.java code artefacts.
- Ruben's execution of Browse Movies involved creation and further updating of the MovieCatalog.java, Cinema.java and Movie.java code artefacts. Ruben also viewed Ticket.java a number of times.

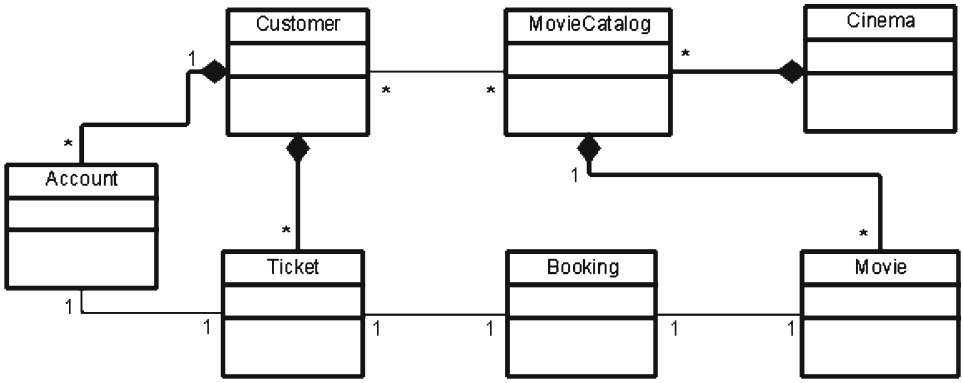



*Figure 1.* Class diagram for TickX.

In this scenario, the Purchase Tickets task is associated with Bill and Amy, and a number of code artefacts. Also, MovieCatalog.java is associated with all three collaborators as well as the two tasks. Some typical context awareness questions that can now arise include: Who is the appropriate developer to seek for help on Purchase Tickets? Which artefact has most impacted the state of Purchase Tickets? Which of the code artefacts or tasks has Amy contributed most to? Which of the tasks or developers has most affected the state of MovieCatalog.java?

The model proposed here helps to address such questions.

## 4. The continuum of relevance (CRI) model

This research proposes that context awareness in distributed, collaboration spaces can be achieved by capturing development events that occur within these spaces. These events may then be used to identify the current actions, activity levels, extent and sphere of influence of the different entities that exist in a collaboration space. Such cues are built up as developers go about their daily tasks, leaving historical traces behind. The basis of the CRI model is the monitoring of core interactions with system artefacts (program files) such as views, updates, creates and deletes. The model is used to provide relevance rankings that depend on the context of work being carried out by a developer. Rankings are then provided of tasks, developers and artefacts in that context. The nature of a shared collaboration space means that a developer can be identified as being highly relevant to the current state of a particular task or artefact instance, but not in any way relevant to the state of another task or artefact instance, though all such instances exist in the same shared collaboration space.

The entities considered in the model are defined as follows:

– A *project* is an endeavour embarked on to create a software product or service and serves to bound the collaboration space.

- A *task* is viewed as an activity that is required to be accomplished in order to achieve a software project. Tasks can be use cases, user stories in agile processes, bug reports, etc.
- *Developers* are the team members that work within a project context.
- *Artefacts* are project components such as software modules and documents that are manipulated by developers.

This collection of entities is strongly inter-related. Projects are realised by developers working on artefacts within the context of a task. There exist many-many relationships amongst developers, tasks and artefacts, although no direct relations are currently supported for entity instances of the same type.

Relationships between entities are established by interactions events—the operations that a developer can carry out upon an artefact within the context of a task. Rather than monitor the entire space of possible interactions that can occur, the CRI model focuses on a core set of four interaction types that influence the changing state of a software project—create, update, view and delete. The following assertions are made about these core interaction events:

- A *create* event is responsible for the manifestation of a tangible artefact within a collaboration space.
- An *update* event affects the state of an entity instance directly. Associated with an update is the update delta—the absolute difference in the number of characters associated with the artefact before and after the event (Mockus and Herbsleb 2002).
- A *view* event *indirectly* affects the state of entity instances—viewing an artefact instance can enhance understanding in order to update the same artefact or other artefact instances.
- A *delete* is responsible for transforming an entity to an intangible state, where it is unable to receive any further events. (Deleted entities are retained in the historical perspective).

During collaborative software development project, different *work contexts* (associations between task, developer and artefact entities) are formed that characterise the relationships amongst entities in a collaboration space. These work contexts are constantly changing in response to events, and entities may participate in one or many work contexts.

From the example scenario described in Section 3 it is possible to create work contexts (represented as graphs) for each entity to capture the relational properties between them. Each interaction event related to an entity can contribute a node to the context graph (if an interaction event refers to an entity instance not yet represented in the graph, a node for the instance is added to the graph). For example, the context graph of Amy will consist of every task participated in (just one—Purchase Tickets) and code artefacts that she has created, updated or viewed (there are five of them). Similarly, the context graph of the Purchase

Tickets task will consist of every code artefact that was created, updated or viewed and the developers that carried out the interaction events while actively working on the task. Finally, the context graph of each artefact (consider MovieCatalog.java, for example) will consist of every task and developer associated with the views, updates and create carried out on that artefact. Figure 2a, b and c illustrate these work context graphs. Similar graphs are created by CRI for all other developers, tasks and code artefacts.

To further investigate the properties of interaction events, and their weighted influence on the relevance of entities in a collaboration space, a study of CVS records associated with real development projects was performed. These records were derived from a group project software engineering class at the University of Strathclyde and open source Eclipse IDE technology and tools projects. CVS repositories of 200 artefacts from a combination of the Eclipse Communication Framework (ECF),[2] Dash,[3] Mylar,[4] Equinox,[5] and Eclipse Modelling Framework (EMF)[6] open source projects were analysed. Only artefact check-ins with version repositories associated with more than one developer were considered.

The results showed that developers associated with the first check-in of an artefact were also associated with 49.6% of subsequent checked in versions. It is therefore asserted that the create event is particularly important relative to other interaction types. Furthermore, while it is expected that view events can enhance understanding of project entities, studies conducted by Zou and Godfrey (2006) suggested that random view events, which are irrelevant to on-going development work, can also occur. In weighting the influence of view events on the relevance of entities in a collaboration space, it is therefore important that the effects of such irregularities are inhibited.

Based on the insight obtained from these interaction types, the weightings shown in Table 4 are assigned to each interaction event type. A view interaction event is equivalent to 10 units of absolute update delta,[7] while a create interaction

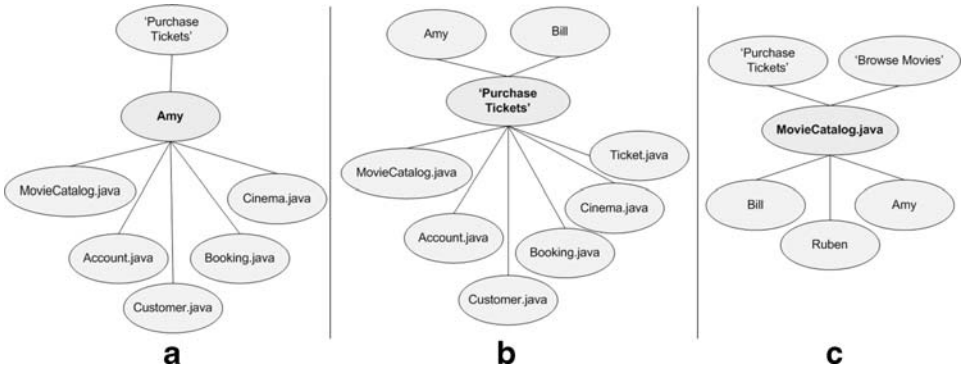



*Figure 2.* Work context graphs for Amy, Purchase Tickets and MovieCatalog.java.

*Table 4.* Interaction type weightings.

| Interaction type | View | Update | Create |
|---|---|---|---|
| Weighting factor | 0.001 | 0.0001*Δ | 0.01 |

Δ—Absolute update delta (magnitude of the update)

is equivalent to 100 units of absolute update delta. Related work by Fritz et al. (2007) has also suggested the importance of create events and of identifying the authors of code artefacts (Kersten 2007). A similar approach in associating weights to interaction events has been used in the development of the Mylyn degree of interest model, where the selection event in Mylyn corresponds to a view in CRI, and edit in Mylyn corresponding to updates in CRI. In Mylyn, a scaling factor of 1 was assigned to selection, propagation and prediction events. Similarly, factors of 0.7 and 0.017 were assigned to edit and decay events respectively (see Table 2). These values were determined based on usage statistics during the programming of Mylyn itself and validated based on feedback from other developers' usage of the tool (Kersten 2007).

A fundamental assumption in CRI is that the size of an entity's work context or the number of other entities that an entity exacts its presence on, is proportional to the relative influence such an entity exacts on the collaboration space. For example, a task that has existed for a long time in a collaboration space and has several developers implementing the task using a number of artefacts is considered to hold more information about the state of the project compared to a task that is newly introduced into the collaboration space and has a small number of associated developers and artefacts. A similar analogy holds for artefacts and developers. This size dimension is captured by the concept of *sphere of influence* (SOI).

SOI is a general concept used to capture both geographic and semantic groupings, and provides a well-defined boundary for interactions. For example, Gutwin et al. (1996), in their work on workspace awareness for groupware systems, refer to SOI as where collaborators can make changes within a shared artefact. SOI in CRI refers to a region over which an entity exacts some kind of relevance (which is in turn determined by the interaction events) and is defined by its work context (and is directly proportional to the number of entities that constitute a work context).

The SOI ratio is used to represent the relative influence an entity exacts on the collaboration space. The SOI ratio of an entity is defined as the ratio of the total number of unique entity instances directly associated with an entity (the size of its work context) compared to the total number of unique entity instances in the whole collaboration space (excluding same-type associations—developer-developer etc.).

Based on the example scenario described in Section 3, it is possible to calculate the SOI of each entity represented in the collaboration space. Figure 3 is a sphere of influence representation of developers for the TickX project scenario. Similar representations can be created for tasks and artefacts. As shown in Figure 3, the

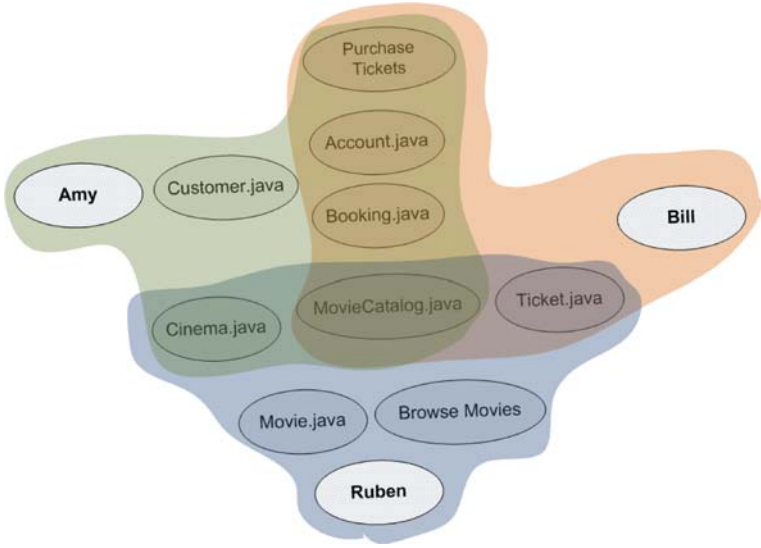



*Figure 3.* Sphere of influence representation for developer entities in the TickX collaboration project.

sphere of influence of Amy is defined as 6/9 (entities within Amy's work context/ total number of entities—2 tasks and 7 classes). Similarly SOI ratios are calculated for other developers (e.g. Bill's is 5/9) and also for artefacts and tasks.

Entities that compose a defined sphere of influence can be characterised with overlapping properties. For instance as shown in Figure 3, Customer.java only falls within the sphere of influence of Amy, while MovieCatalog.java falls within the sphere of influence of each of Amy, Bill and Ruben.

The maximum sphere of influence that an entity can achieve is 1. This is for a case where an entity is associated with every other entity that is not its type in the collaboration space. This is typical for scenarios where the collaboration space consists of a single artefact, task or developer. A minimum value of 0 is achieved if the work context is empty; this is typical for scenarios where, for example, a developer has not interacted with any task or artefact. In general, as the number of entities in a work context increases relative to the number of entities in collaboration space, the sphere of influence ratio also increases.

The concepts of work context, interaction events associated with an entity, and the variation of its SOI ratio, forms the basis of the CRI model. This model is intended to provide an accurate, real-time perception of the *overall* work effort of individual developers as well as their *recent* work; an indication of which tasks and artefacts have consumed most effort over all developers; and hence an indirect indication of the relevance of entities to a project. CRI is a linear model that cumulatively builds the relevance values of entity instances as they are associated with interaction events and as their SOI ratios vary. These cumulative relevance values are derived for two modes: *history* and *recent*.

### 4.1. CRI history mode

The history mode aims to provide awareness of the *overall* dissipation of work effort across entities that constitute a selected task, developer and artefact work context respectively. This is computed by linearly combining the relevance value associated with an entity in a selected work context before an interaction event with the relevance gained as a result of the interaction event. The relevance gained as a result of an interaction event is dependent on the type of interaction event and the SOI ratio of the selected entity work context. More formally, the cumulative relevance gained by an entity instance *e* in response to an interaction is represented by Eq. (1). The type of interaction is represented by *t* and the different values it can assume are shown in Table 4. The SOI ratio is represented by *s*, *n* is the total number of interactions to date associated with entity *e*.

$$X_{(n)e} = X_{(n-1)e} + t_{(n)e} * S_{(n)e} \tag{1}$$

In other words, the relevance value for entity *e* after *n* interactions is based upon its previous value plus the value of the interaction multiplied by the SOI ratio of the entity.

### 4.2. CRI recent mode

The recent mode aims to provide real-time awareness of the *current* dissipation of work effort across entities that constitute a selected task, developer and artefact work context. The core difference between history and recent mode is how the relevance values of *inactive* entities—those untouched by an interaction event—are computed. In the history mode relevance values of inactive entities remain unaffected, while in the recent mode relevance values of inactive entities in a work context decay for every interaction event that impacts that work context. Thus, the longer the duration of inactivity associated with an entity within a selected work context, the more the relevance of the inactive entity decays. This process of decay in relevance is represented using the notion of *periodic decay* and is dynamically determined by the SOI ratio of the selected entity work context and the interaction event type. (Periodic decay was influenced by a similar notion in Mylyn (Kersten 2007)). Relevance lost due to periodic decay represents the negation of the relevance gained by the active entities defined in an event that impacts a selected work context. The effect of periodic decay is implemented by decreasing the relevance values of inactive entities e′ when an interaction takes place and is defined in Eq. (2).

$$X_{(n)e'} = X_{(n-1)e'} - t_{(n)e} * S_{(n)e} \tag{2}$$

So the impact of period decay is to subtract the additional relevance computed for the active entity away from all inactive entities in that work context. Both these equations are applied after every interaction takes place.

The outcome generated by both history and recent mode calculations is the association of numeric values to the relevance of entities that constitute a selected work context. From this a ranking can be created from which collaborating developers can then obtain awareness of overall and recent work effort that has impacted the different work contexts of distributed entities bound by a software project in a collaboration space. As well as entity rankings, coloured labels of varying intensity are provided to indicate the relative strength of relevance.

## 4.3. Illustration

To illustrate how the CRI model can be used to obtain a perception of the relevance of an entity instance to a selected work context it is assumed that the interaction trails shown in Figure 4 were the events used to achieve the earlier TickX project. Any selected time on the timeline corresponds to at least one event associated with a developer, a task and an artefact. For instance, the project started with the creation of the Account.java code artefact by Amy while contributing to the Purchase Tickets task on timeline 1. Timeline 7 corresponds to two events occurring at the same time: Ruben updated Cinema.java (update delta 50) as he worked on Browse Movies, while Bill viewed Account.java as he worked on Purchase Tickets (indicated by the three '✚'s in timeline 7).

Figure 5 represents the history and recent mode relevance list outcomes for the entities that constitute the work context of the Purchase Tickets task. Entity instances with greater relevance values are positioned at the top of the relevance list. Also the relative differences in cumulative relevance values are proportional to the relative distance between instances in the list. Each list consists of entity instances of the same type that constitute a selected work context.

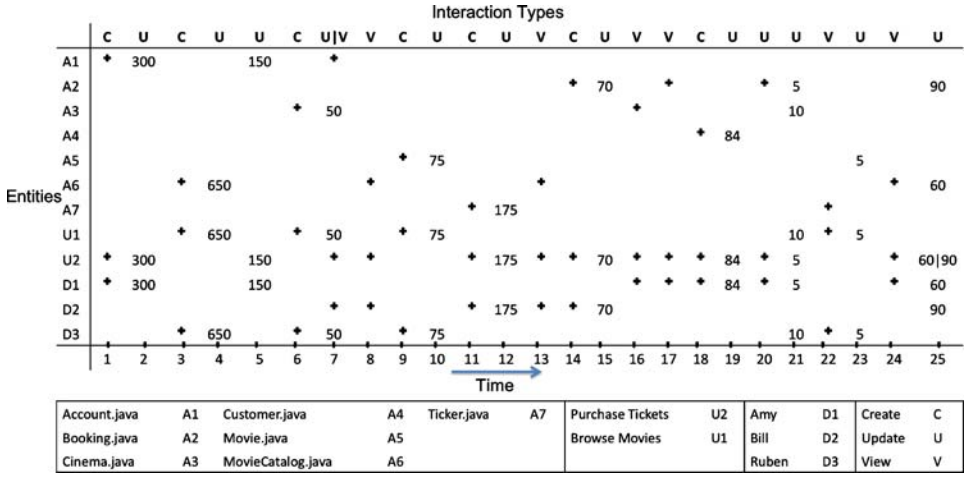



*Figure 4.* Monitored interaction trails used to achieve TickX across 25 timelines.

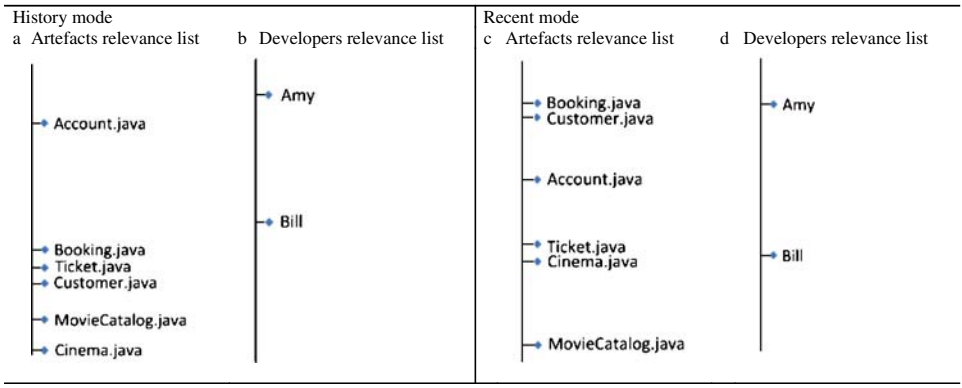



*Figure 5.* History and recent mode artefact and developer relevance lists for the purchase tickets work context.

The relative positions of entity instances on history and recent mode relevance lists can be used to obtain insight into overall and recent work effort. Figure 5 shows that within the Purchase Tickets task work context Account.java has had greater overall influence on the state of the task, while, in recent mode, Booking.java is associated with most coding effort. Also, the figure shows that Amy is attributed with most overall and recent coding effort in achieving the task. Similar relevance lists are created for every other task, developer, and artefact.

## 5. Model implementation

A client-server architecture was chosen to implement the CRI model where each developer's Eclipse IDE is a client and the model processing logic and storage of interaction sequence data is performed on the server. The client monitors sequences of view, update, create and delete interaction events executed within Eclipse. Eclipse was chosen because of its open, plug-in architecture. When a network connection exists, this event data is offloaded to the server and synchronised with that of other developers. While there is no connection (or a slow connection) the client can temporarily store event data locally and perform local model processing logic to give the developer a partial view of current relevance—*offline mode*. The CRI implementation architecture is as shown in Figure 6. The architecture is distributed across client and server ends, and consists of four core layers: the model, event, messaging and Rich Client Platform (RCP). The client end of each layer is plugged into the Eclipse platform while the server end resides on an Apache Tomcat web application server.

Projects are defined within Eclipse itself and the current collaboration project is identified via a unique project identifier which is then associated with any code

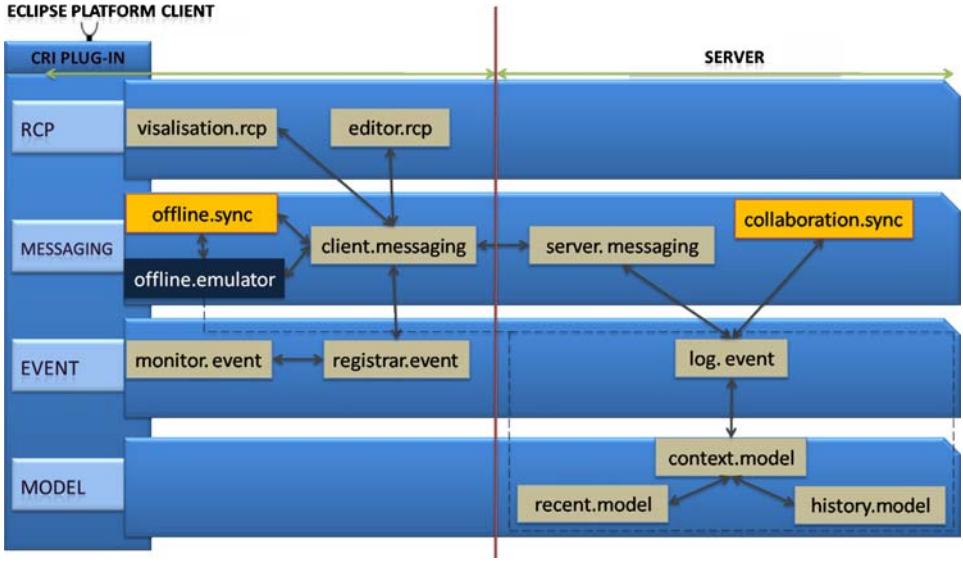



*Figure 6.* Core layers of the CRI implementation architecture.

artefact being monitored by CRI. Each time a code file is opened in Eclipse it is registered as a view event by the *registrar.event* component in the event layer. Similarly, each time a new file is created it is registered as a create event. The update delta is determined by comparing the absolute difference in the number of characters in a file before and after a save action is triggered in the editor. When artefacts are deleted the work context graph of the deleted node freezes. The history of interaction events associated with the deleted node is, however, retained in the log of event trails and can always be viewed in history mode to obtain the relevance of the entity instance prior to its deletion.

The model layer is the main event processing unit in the architecture. This layer is responsible for the formation of entity work contexts and their related SOI ratios. The event layer is responsible for capturing and archiving interaction event sequences generated within a collaboration space. The *log.event* component is the clearing centre and data warehouse of all events generated by collaborators. The messaging layer carries out asynchronous processing of request/response messages from the server. The *offline.emulator* component emulates the server end functions of the model and event layers while a developer is generating interaction events in the offline mode. Finally, the RCP layer resides only on the client end, and provides the minimal set of components required to build a rich client application in Eclipse.

Figure 7 is a snapshot of an Eclipse view of the *visualisation.rpc* component (which takes up a small area of the Eclipse real estate). System developers can open, activate and deactivate their tasks (use cases) of interest by using the popup menu labelled 3 in Figure 7. All interaction events carried out by a developer

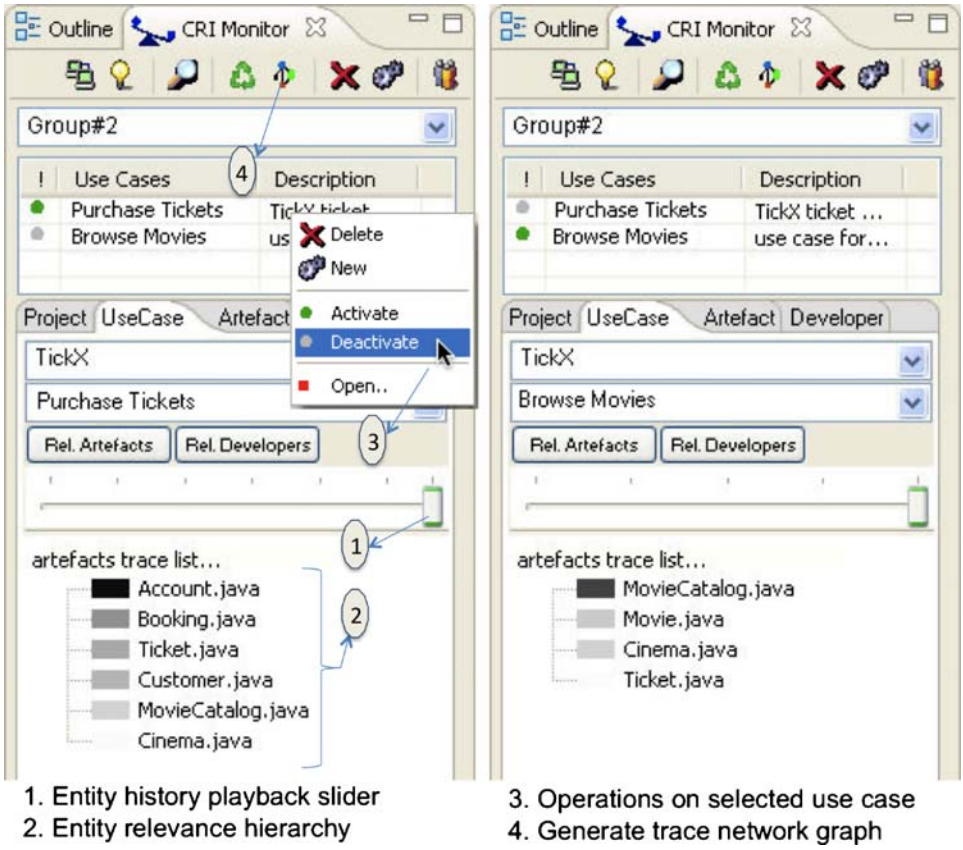



*Figure 7.* Relevance based ordered visualisation of artefacts associated with TickX tasks/use cases.

are traced to the work context of an activated use case. The RCP layer is also responsible for generating relevance based ordering and social graph visualisations.

The visualisation of entities involved in a selected work context is structured such that entity instances with greater relevance values are positioned at the top of the relevance list. The relative difference in the relevance values of entities is depicted using varying colour intensity. Entities at the top of the relevance list are represented with greater colour intensity. Closely related entities show the same relative colour intensity. Label 2 in Figure 7 highlights a relevance-based, ordered visualisation of code artefacts that constitute the work context for the Purchase Tickets (Browse Movies is to the right of this). It can be seen in the figure that although MovieCatalog.java has been used to achieve both tasks, its relative impact on the state of Browse Movies is greater than on the state of Purchase Tickets. Similar hierarchies are also provided for the relative impact of developers on the executable state of the two tasks.

The CRI implementation also includes the capability to *replay* the evolving state of each context. As an example, a playback of the evolution of the code artefacts and their relative relevance associated with Purchase Tickets can be obtained by sliding through the slider bar labelled 1 in Figure 7. The generation of the social graph visualisation—see Section 6.2.1 and Figure 15—is triggered by the button labelled 4 in Figure 7 (this uses the JUNG[8] (Java Universal Network/Graph) framework).

## 6. Evaluation of the CRI model

To evaluate the model the following research question was investigated in an empirical study:

> *Can a model based on real-time monitoring of IDE interactions, such as creates, edits and views, enhance contextual awareness during distributed, collaborative software development?*

### 6.1. Methodology

The study involved ten advanced software engineering students in the third year of their Integrated Masters/Honours programme in Computer Science at the University of Strathclyde, UK, all of whom volunteered to participate. All participants had at least 2.5 years of object-oriented development experience using Java. They were all participating in the group project class developing 'Gizmoball'[9]—an editor and simulator for a pinball table first proposed by MIT—and working in groups of three. Of the ten participants two groups of three were the best two performing groups in the class ('G1' and 'G2'), another group of three participants came from a relatively strong group ('G3'), and a single student came from a group that was of average performance ('G4'). The groups had been designed to consist of individuals of similar academic ability so as to encourage equal participation.

Participants were not restricted to time or place of work. Groups were required to have at least one face-to-face meeting every week; during this time they also discussed their progress with the teaching assistant (TA) coordinating the group. Feedback from participants suggested that, besides the mandatory meeting, they also held occasional collocated meetings. All the groups used a version control system. Feedback from group G1 suggests occasional pair programming practice, while group G2 also used a wiki system.

CRI data was gathered over a 6 week development period—2 weeks of prototype development and 4 weeks of full-scale development. The model was used during development (rather than maintenance) and was used in both a distributed and collocated setting—all participants recorded instances of working from home and within the university campus, the gathered data suggested that participants spent more time working at different times or places than they spent

working together. At the end of the 6 weeks, structured interviews were conducted with eight of the participants—the two who did not wish to be interviewed had used CRI the least ('Blair' and 'Greg'—see Table 5). The interviews were personalised to include CRI relevance views and other project information specific to the participant being interviewed. It also had a mix of open and closed questions to allow the interviewer (the first author) to follow up interesting responses with more detailed questioning. All data was anonymised for analysis and presentation.

An audio record of all the interview sessions was carried out with the permission of the participants. These audio records were then transcribed and analysed. Each participant was interviewed separately. Apparent agreements and disagreements in feedback were addressed after the interview sessions by comparing each recorded feedback snippet with a replay of actual interaction event trails captured by CRI for the work context that matched the feedback described by the participant. The outcome of this analysis showed no apparent contradiction in statements made by different members of each group that was interviewed.

### 6.2. Results

During the study 7166 CRI interactions over 16 tasks were recorded—see Table 5. Of this total 0.11% were delete interactions, 1.98% creates, 45.72% updates, and 52.20% views. 50% of the tasks involved two or more collaborating developers. On average, 448 CRI interactions were registered per task with a minimum of 3 and a maximum of 2,479. An average of 717 CRI interactions was registered per participant with a minimum of 4 and a maximum of 1,157. 142 artefacts were created and monitored by CRI, 18% of the artefacts were associated with two or more collaborators. 62.4% of the total update delta was associated with artefact creators.

*Table 5.* Total interactions associated with each participant in the detailed study.

| Group | Collaborator | Updates | Deletes | Creates | Views | Total interaction events per collaborator |
|---|---|---|---|---|---|---|
| G1 | Alex | 550 | 1 | 23 | 740 | 1,314 |
| G1 | Tony | 567 | 1 | 18 | 937 | 1,523 |
| G1 | Luke | 232 | 1 | 11 | 210 | 454 |
| G2 | James | 1,016 | 1 | 54 | 1,157 | 2,228 |
| G2 | Paul | 42 | 0 | 9 | 222 | 273 |
| G2 | Tracy | 778 | 1 | 8 | 223 | 1,010 |
| G3 | Blair | 12 | 0 | 5 | 43 | 60 |
| G3 | Greg | 0 | 1 | 0 | 3 | 4 |
| G3 | Boris | 57 | 2 | 12 | 134 | 205 |
| G4 | Smith | 22 | 0 | 2 | 71 | 95 |
| | Total | 3,276 | 8 | 142 | 3,740 | 7,166 |

As part of the initial evaluation tests were carried out to ensure that the implementation of the model behaved according to its design when used in practice. This was firstly achieved by carrying out a controlled injection of real project updates starting from a known state. The changes in relevance rankings were then tracked and verified to ensure artefacts progressively moved up and down the rankings as expected both in recent and in history mode, with changes occurring more rapidly in recent mode due to the effect of periodic decay. Also, the impact of each of the CRI components was investigated individually. SOI was shown to increase the relevance ranking of entities associated with high SOI e.g. when an artefact was worked upon by a developer who was also associated with many other artefacts. Similarly for each interaction type: create interactions were important to identify the developer and the task that were responsible for artefact creation—though view and update interactions dominate over the lifetime of a project; there were examples where developers were persistently viewing artefacts but not editing them but it was clear that reading the code was important to their current task; and, update interactions indirectly captured edit activity by developers working in the context of a task. Finally, the effect of periodic decay was shown to have the desired impact of dampening relevance of inactive entities in recent mode (as opposed to history mode).

In investigating the main research question, three contributions to increased awareness during collaborative development as a result of the use of CRI were identified.

#### 6.2.1. *Results related to accuracy of CRI compared to participant opinion*

Firstly, the study uncovered examples where it appeared that CRI provided a more accurate record of relevance than that of individual developers themselves. A set of questions in the structured interview explored the perception of relevance held by participants compared to the CRI relevance rankings. Before the commencement of each interview session, a separate paper list (in random order) was made for each set of artefacts and tasks that the participant had worked on. Before the participant had a view of any ranking information from CRI, they were asked to rank the top four artefacts and tasks in descending order based on a number of criteria, including overall coding effort and recent coding effort. Analysis of these results showed that 62.5% of the time a participant's top selection, in terms of overall work effort, matched that of CRI; for recent work effort the match was 37.5%. 87.5% of the time the participant's top selection was in CRI's top 5 for overall work effort, and 62.5% of the time for recent effort. So there was quite a mismatch between the CRI ranking and the estimates provided by individual developers, particularly with respect to recent work.

One possible reason for this is the uncontrolled factors that may have impacted on CRI results. In the interviews each participant was asked to state how frequently they remembered to log into CRI, answered on a scale of 0–100%. To understand if CRI impacted upon the normal working practices of participants they were also asked: how frequently a new task was created or activated as the

participant's work context changed, how difficult it was to work within the context of an identified task, how difficult it was to create a new task, and how difficult it was to activate an existing task in CRI (all answered on a Likert scale of 1–7). Interview data suggested that CRI may have only captured 60–90% of the total work effort of developers—see Figure 8, and that developers only changed task within CRI 25–50% of the time they actually switched task in practice—see Figure 9a–d. CRI results are therefore likely to be subject to an element of inaccuracy.

To investigate these results further, participants were shown the *actual* CRI relevance rankings for both history and recent modes, after they had given their initial responses, and asked for any insights into possible discrepancies. This led to the identification of possible reasons for the mismatches between CRI rankings and those of the participants.

One potential reason is that participants formed a perception of effort based on a related cluster of code artefacts. This snippet from participant 'Paul':

> *"I am not sure of Wall.java, I don't think I put as much coding effort into Wall.java as I put into the flipper related classes…"*

Analysis of the interactions associated with Paul showed that the CRI ranking was 'correct'—the *total* effort of Paul on the flipper classes, as measured by CRI, was greater than Wall but none of them individually exceeded Wall (for Wall 12 views and a total update delta of 1,400 characters were recorded, for Flipper it was 8 and 214, for LeftFlipper 7 and 778, and for RightFlipper 19 and 25).

Another reason that may have influenced the mismatch was the *size* of the artefact—one participant discounted a code artefact because it was only a small driver module, but then acknowledged that it was modified each time the user interface was tested; another was the perceived *difficulty* associated with an artefact—one participant ranked an artefact lower down as it was perceived as "*straightforward*" with many edits that were "*not hard to implement*"; and lastly a participant discounted an artefact because all the effort had been in terms of "... *simply copied and pasted from an online source*".

The other main reason uncovered for mismatches appears to have been due to flawed recollection and estimates by participants—which is to be expected. For example participant 'Boris':

> *"Yes I understand why OuterWall.java should be there, I was recently working on it…I don't know why BouncingBall.java will be higher than Flipper.java…I know why! …"*

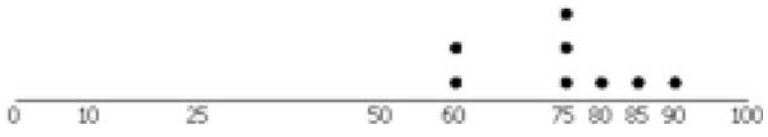
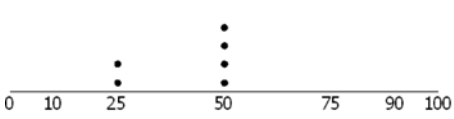
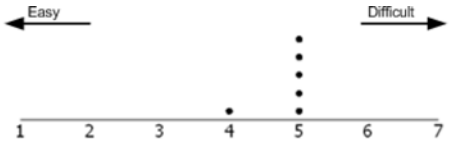


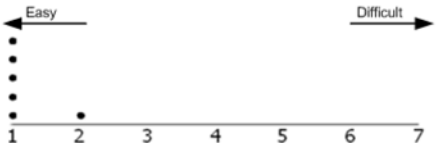

*Figure 8.* Dot plot showing percentage frequency participants remembered to log into CRI.



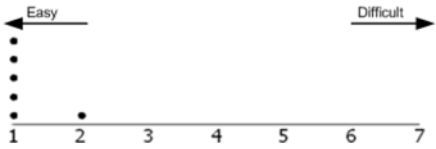

a: Percentage frequency participants activated or created tasks as work context changed




b: Difficulty experienced by participants in always working within the context of a task




c: Difficulty experienced by participants when creating a new task using CRI tool




d: Difficulty experienced by participants when activating an existing task using CRI tool

*Figure 9.* Experience feedback from participants.

Participant 'Alex':

*"I am a bit surprised that CollidableCircle.java is positioned that high; I thought it would have been a bit lower… Yeah, thinking more about it, the ranking looks about right, just that sometimes I depend on my recent coding experience…"*

While the factors identified as possible reasons for potential mismatch between CRI rankings and those of the participants do not necessarily suggest CRI is 'correct', these examples suggest that an automated awareness system such as CRI has the potential to maintain more accurate rankings than might be possible by human developers who can be influenced by recent effort or overwhelmed by the amount of work done or scale of a project. However, the goal of CRI is to be more useful than that—it aims to *enhance* the awareness of developers—providing them with information that would be otherwise difficult to obtain in an environment that is both distributed in space and time.

### 6.2.2. *Results related to deeper collaboration insights provided by CRI*

The second, and main, contribution of CRI is in enhancing awareness of the complex dependencies that can exist in different work contexts within a distributed, collaborative project. These dependencies include the fact that a code artefact may be associated with a number of developers and used to achieve a variety of project tasks including system use cases and maintenance changes. Similarly, a project task can be associated with a number of collaborating developers and a range of code artefacts. Finally, a developer will be working on a number of project tasks and using a wide range of code artefacts to achieve each

task (Gutwin et al. 2004; Mockus et al. 2002). The aim of CRI is to be able to extract relevant awareness information for particular work contexts from this network of entities and their interdependencies.

A snippet of feedback from Luke from group G1 on insights he obtained while sliding through the history of entities constituting their collaboration space is shown below:

> *"…I had a slide through the relevance positions of developers and java classes for the File Demo task… I noticed that it has only been 'Tony' working on that task. TriangleBumper.java and MainProgram.java were the original classes I noticed he started with, and it was so for quite a while…*
>
> *Currently there are a number of other classes he has used for that task…*
>
> *If a maintenance task is to be carried out on my system, such information will really be useful too… Since I can see the relative change of relevance that an artefact or a developer would have had in association with a task used to realise the system…"*

The feedback snippet from Luke suggests that he did not need any formal or informal collocated meeting with other members of his group to obtain awareness of the state of the File Demo task. Through his use of CRI, he was able to understand that it had only been Tony that had been working on File Demo; from this, he also obtained insight into the relative significance of the artefacts Tony was using to accomplish the task.

Analysis of developer interaction data captured during the study enabled the detailed investigation of these snippets. In particular it was possible to recreate development paths and investigate the validity and details of comments made by participants during the interviews.

The analysis of File Demo in Table 6 shows the percentage of interaction events Tony was associated with for each artefact he used in achieving the File Demo task. A total of 13 code artefacts were used, he interacted significantly with TriangleBumper (32.03%), LeftFlipper (23.60%) and MainProgram (17.55%). Figure 10 shows an activity-time plot of File Demo and snapshots of the related artefacts relevance list at different intervals over the history of File Demo. The artefact relevance lists in Figure 10 labelled 1–6 show that the initial phase of Tony's work on File Demo actually involved TriangleBumper and MainProgram and later progressed to a number of other code artefacts. The positions of TriangleBumper and MainProgram have also been consistently high on the relevance list over the history of File Demo as demonstrated by the artefacts labelled a and b on the artefact relevance lists. This analysis confirms the awareness obtained from CRI by Luke on File Demo.

Obtaining such awareness would be difficult in a distributed setting without the use of a relevance model such as CRI. In particular CRI captured the fact that

*Table 6.* Percentage of developer and artefact interaction events associated with File Demo task.

| | Views | Updates | Absolute update delta | % of standardised interactions |
|---|---|---|---|---|
| **Artefacts** | | | | |
| GameModel.java | 3 | 5 | 184 | 2.23 |
| MainProgram.java | 21 | 21 | 1,475 | 17.55 |
| TriangleBumper.java | 24 | 21 | 2,836 | 32.03 |
| GameObject.java | 4 | 1 | 32 | 0.75 |
| RightFlipper.java | 9 | 5 | 859 | 9.88 |
| LeftFlipper.java | 10 | 4 | 2,166 | 23.60 |
| Ball.java | 2 | 1 | 2 | 0.23 |
| GizmoHandler.java | 7 | 2 | 563 | 6.59 |
| CircleBumper.java | 3 | 2 | 12 | 0.44 |
| ApplicationWindow.java | 2 | 3 | 67 | 0.91 |
| Absorber.java | 3 | 2 | 6 | 0.37 |
| FileHandler.java | 6 | 3 | 144 | 2.12 |
| GameWindow.java | 3 | 9 | 286 | 3.29 |
| **Developers** | | | | |
| Tony | 97 | 79 | 8,632 | 100.00 |

Tony had a wide range of interactions with a variety of artefacts, so for example Ball had little significance since Tony had minimal interaction with it and Triangle had much more significance due to the increased interaction with that artefact.

Another snippet of feedback from 'Tracy' in group G2 provides an insight he obtained while sliding through the history of entities constituting the G2 collaboration space:

> *"…Sliding through the history of a task or artefact gives you a feel of how things have moved on, especially after sliding through a history of the artefacts I have been associated with…*
>
> *Having a slide through a task view I can gauge how important an artefact has been to the task over time, I did notice that MainScreen.java has retained high relevance over a long duration now; recently KeyConnectFrame.java has turned out to be high also…*
>
> *… This gave me the clue that these classes are quite important to the User Interface task…*
>
> *I got particularly interested in MainScreen.java when I noticed 'James' and 'Paul' have also used this class… I have been the only one working on KeyConnectFrame.java*
>
> *I believe this information will again be very important to me when carrying out a maintenance task on a system I am not really familiar with…"*

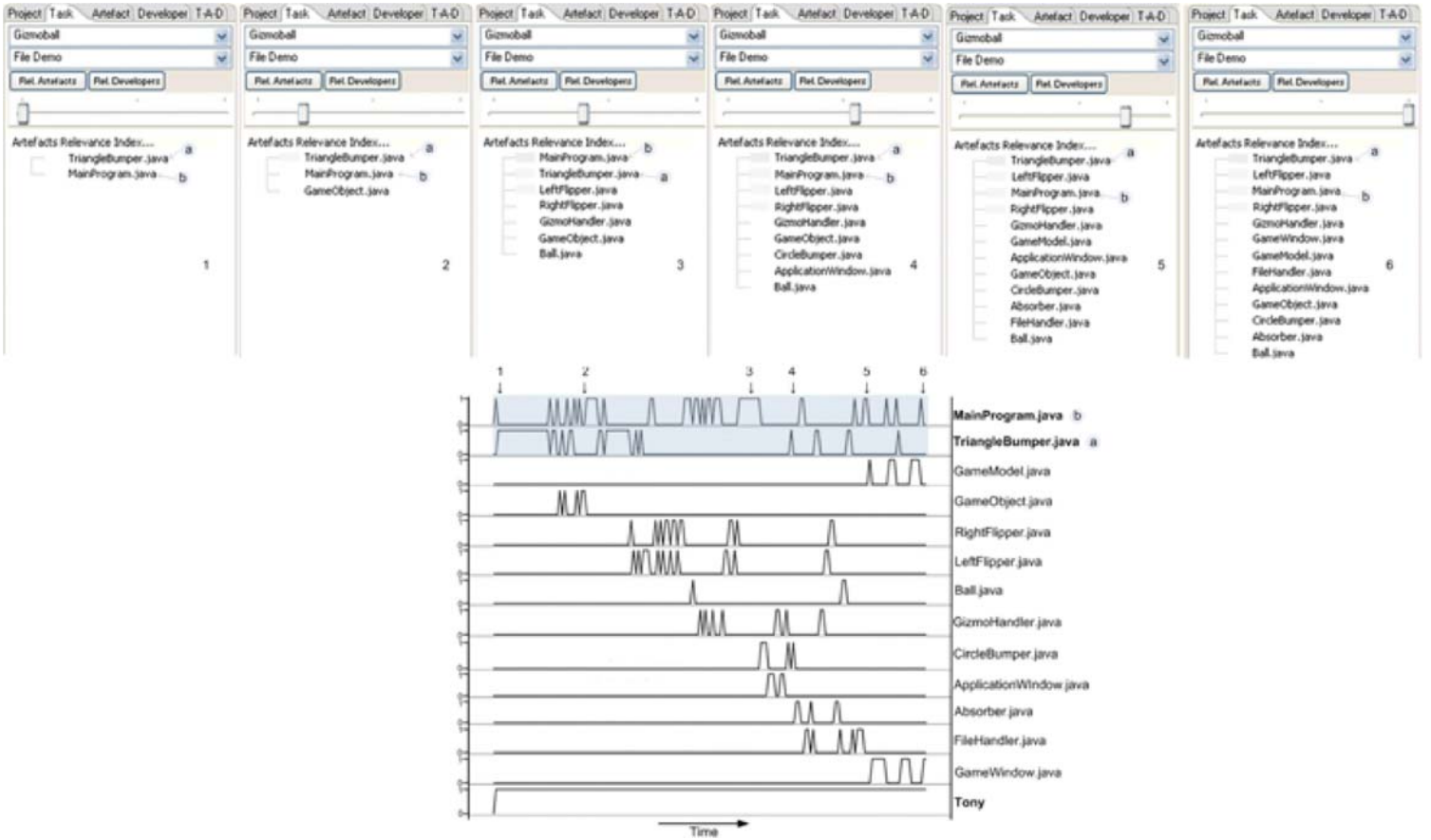



*Figure 10*. Activity-time plot and snapshots of related artefacts relevance list over the history of File Demo.

Tracy's particular interest had been to be aware of other developers that have been collaborating on the same task he was working on and the code artefacts being used to achieve the task. The feedback from Tracy also supports the capability of CRI to enhance contextual awareness of distributed software project processes. Tracy became aware of the impact of the MainScreen and KeyConnectFrame code artefacts on the User Interface task which he was collaborating on with 'James' and 'Paul'. While he had worked primarily on KeyConnectFrame, he had not at any time interacted with MainScreen while performing the User Interface task. Irrespective of not working with MainScreen, using CRI he was aware of the relative impact of the artefact on the task he was working on. Again, he obtained this awareness without having to meet formally or informally with James or Paul.

Figure 11 shows the context graph of User Interface demonstrating that it was accomplished by the three collaborating developers and the use of 71 code artefacts. Useful awareness insights into the state of User Interface would have been more difficult to achieve without the use of a relevance model such as CRI given the number of developers and artefacts involved. Each of the developers and artefacts

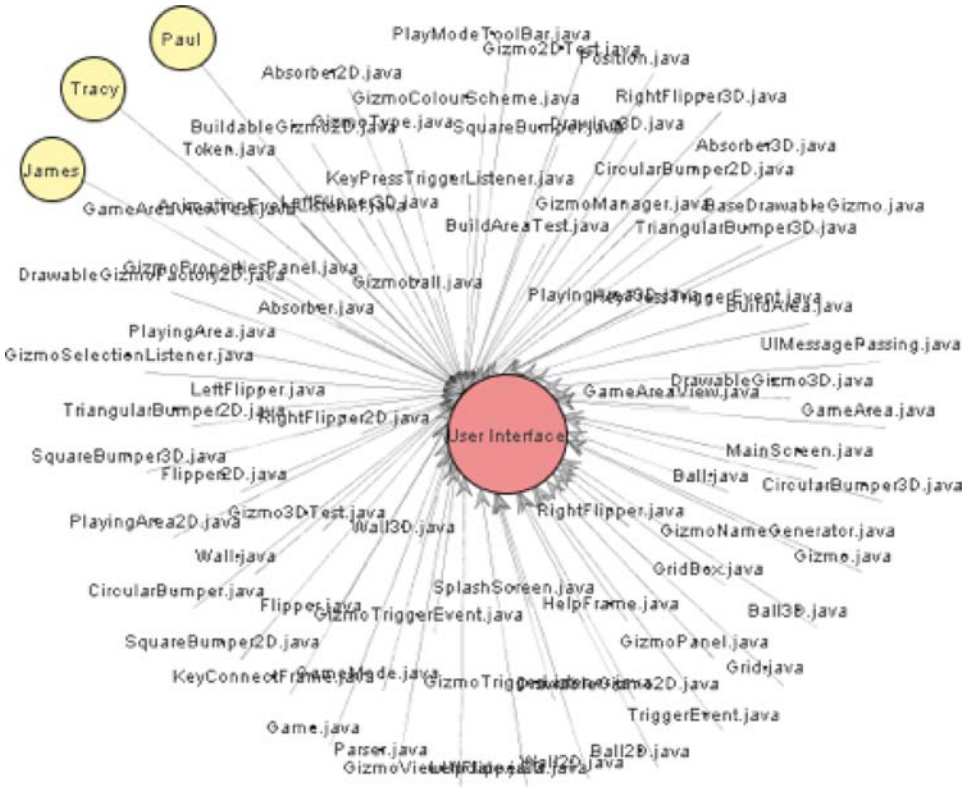



*Figure 11.* The context graph of the G1 User Interface task showing the 71 code artefacts and 3 developers involved in achieving the task.

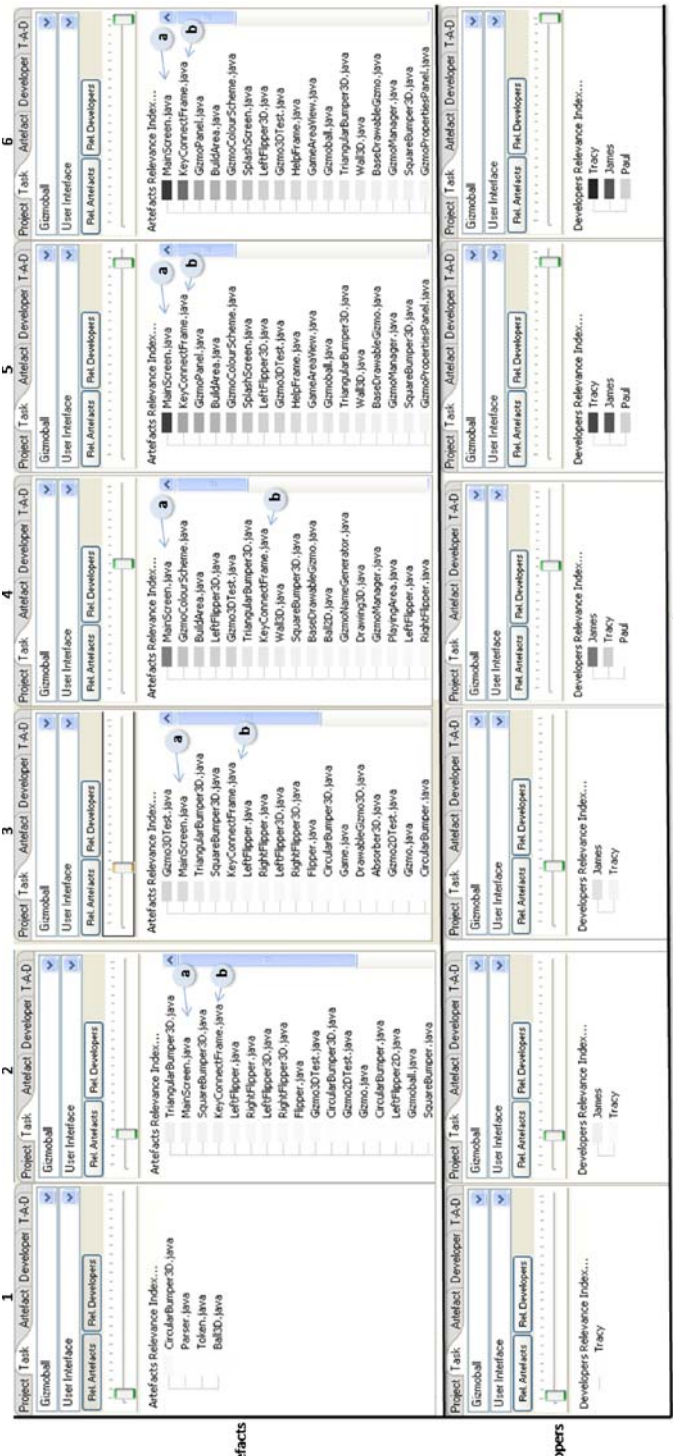



*Figure 12.* Snapshots of developer and artefact relevance lists throughout the history of the User Interface task.

had different levels of influence on the state of User Interface. Figure 12 shows six sequential snapshots of developer and artefact relevance lists at different intervals over the history of the User Interface task. The labelled entities are the artefacts and developers that Tracy was interested in during his collaboration on User Interface. The screenshots show the relatively high positioning of MainScreen over the history of User Interface and the higher positioning of KeyConnectFrame at the later phases of the task. The developer relevance lists also show that the early phase of development work on User Interface consisted only of Tracy while James and Paul became associated with the task as it progressed. The positions of James and Tracy switched on the relevance list at different periods of User Interface development. Figure 12 confirms the awareness obtained by Tracy using CRI during the development of User Interface.

Finally, a snippet of feedback from 'Alex' in group G1 on insight he obtained using the CRI history mode relevance list is shown below:

> *"…If there is an artefact that has remained high on the ranking over a considerable time line, it tells me where the main focus or problems have been in the project…*
>
> *I have been watching the PlayWindow.java and BuildWindow.java classes recently on the Build Mode task…*
>
> *Although I have not worked much on them, I know they have been important in achieving Build Mode …*
>
> *I also noticed that classes are high on 'Luke's' relevance ranking…*
>
> *He is probably doing a lot of work on it…"*

The feedback from Alex implies that using CRI he was able to build awareness of the relevance of PlayWindow and BuildWindow to the Build Mode task. These were the top two artefacts on the Build Mode task artefacts relevance list. Furthermore, he was able to obtain awareness that Luke was more relevant to the state of these two artefacts compared to other developers within the collaboration space. Again, it would have been difficult for Alex to achieve this awareness without a relevance model such as CRI since Build Mode was dependent on all three developers in the group and 70 code artefacts—see Figure 13. Again, each of the developers and code artefacts had different levels of influence on the state of Build Mode.

Figure 14 shows snapshots of entity relevance lists for the final work context state of Build Mode (labels 1a and b), Luke (label 2), PlayWindow (labels 4a and b) and BuildWindow (labels 3a and b). Label 1a demonstrates the high relevance positions of PlayWindow and BuildWindow on the Build Mode artefact relevance list and label 1b shows the high relevance position of Luke compared to the other developers on the state of Build Mode. The artefact relevance list for Luke shown in label 2 further demonstrates the relative relevance of PlayWindow and

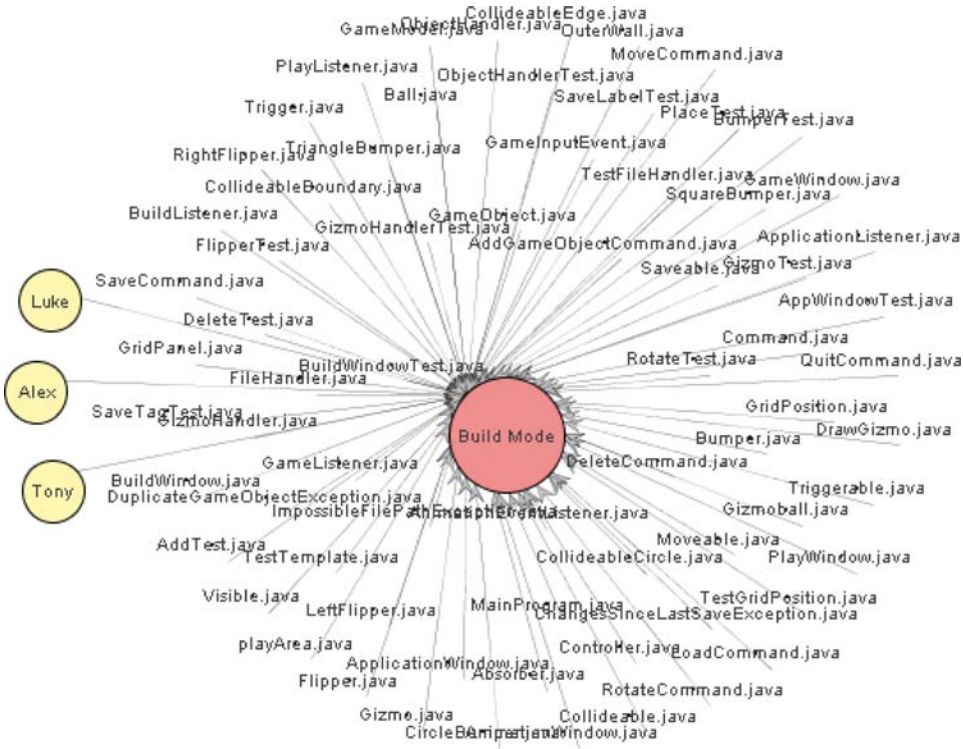



*Figure 13.* The context graph for the Build Mode task showing 70 code artifacts and the 3 developers involved in achieving the task.

BuildWindow to Luke's work context. The developer and task relevance lists for PlayWindow and BuildWindow work context shown in labels 3a, b and 4a, b also confirms the awareness stated by Alex in his feedback.

Firstly, the Build Mode task has had significant impact on the state of PlayWindow and BuildWindow given that they are positioned top on the task relevance list for each of the two code artefacts (labels 3b and 4b). Secondly, while Alex had minimal impact on the state of BuildWindow (label 3a—bottom in the developer relevance list) Luke has had significant impact on the state of this artefact (label 3a—being top in the developer relevance list). Tony also impacted on the state of BuildWindow but not as much as Luke. Furthermore, Alex had not at any time worked on PlayWindow while Luke was the most relevant then Tony (label 4a). Amid the numerous artefacts and three developers that had collaborated on Build Mode (see Figure 13), Alex did not need to formally or informally meet with Luke or Tony to become aware of their relevance or to discover the most important artefacts in this task.

The snippet of feedback for 'Tony' shown below again demonstrates the usefulness of CRI for distributed software development. His aim in sliding

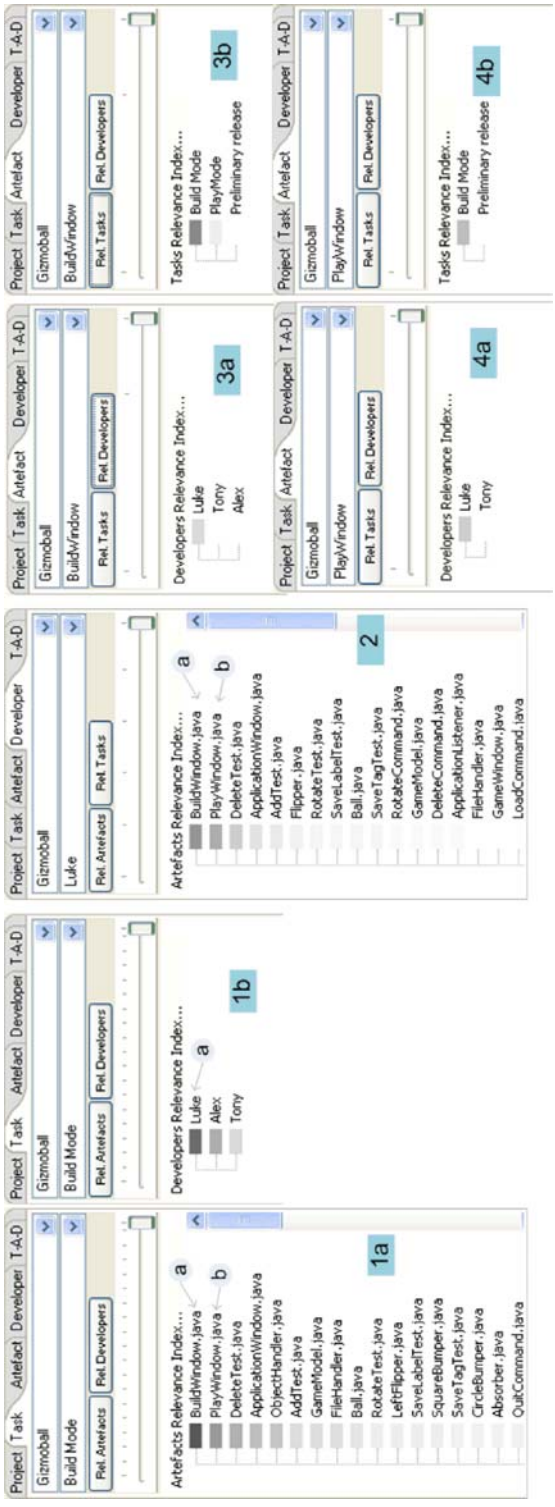



*Figure 14.* Entity relevance lists used by Alex to obtain awareness of group G1 development work.

through the history of entities constituting his collaboration space was to monitor the real-time progress of work on tasks and code artefacts:

> *"... It is always good to see over time, how much you have worked on certain code or tasks...*
>
> *After our group meeting, I will watch to see if there is a certain growth in the task we discussed during the meeting...*
>
> *I noticed some time delay in processing as the timeline gets longer.*
>
> *This feature will be more intuitive if timeline definitions are more specifically defined..."*

Tony's feedback also suggests that sliding through histories of entities could suffer from network latency as the lifetime of the project increases. Also, the usability of the history slicing feature in CRI could be improved by adding more intuitive timeline definitions such as the particular date or hour an interaction was carried out.

### 6.2.3. *Results related to the social graph view of CRI*

Finally, another contribution to enhanced awareness by CRI is at the abstract level provided by the social graph view. CRI social graphs are constructed by merging all the context graphs for each group. The relevance position of an entity, relative to all the work contexts it has been associated with can be interpreted as an estimate of its global importance or 'centrality' compared to all other entities in the graph. All edges (representing interactions) are between entities in different subsets and no entities in the same subset are adjacent. The size of an entity in the graph reflects its importance, and is proportional to its Markov centrality in the network (Latora and Marchiori 2007), arcs between entities reflect dependencies i.e. artefact-developer, artefact-task, and task-developer. Figure 15 shows the social graph for group G1 in this study.

The potential benefit of social graphs is that they present a high level view of a collaboration space which can help identify key entities in terms of their size and relationships. Ideally, they can be used to visualise the potential impact of making a change to a project e.g. removing a developer or an artefact, updating a task; or they may help identify potential bottlenecks in a project. Again the potential benefits of the social graph view were highlighted, with, for example, 'Alex' saying:

> *"...It's the fastest way to get all the information from CRI... I always use the graph to get a general state view of the project... I do check it every few days just to give me a grasp of what is going on with developers in the group and which tasks have had a considerable change recently..."*

Another quote from 'Tony' reflects on the accuracy of the social graph view, highlighting the main task and artefact that have been the focus for his group's

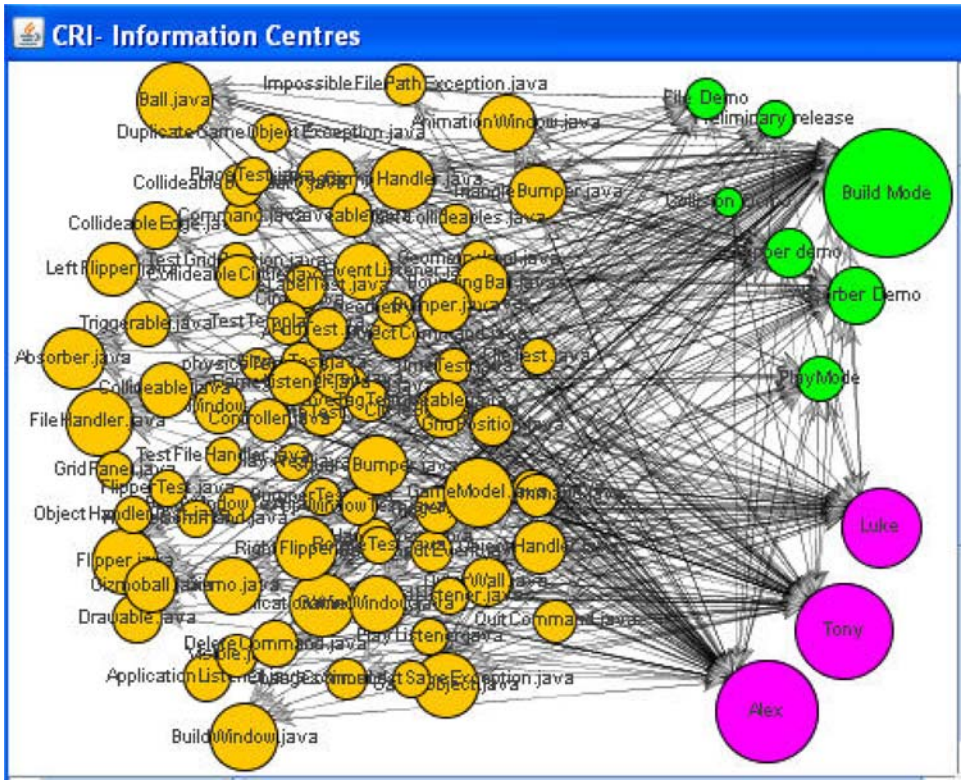



*Figure 15.* Group G1's social graph.

efforts, and also pointing out the limitation of scaling associated with the social graph view:

*"…Will say that is spot on…*

*It's easy to see where the biggest problems are… For instance, I can see that Build Mode task and BuildWindow.java have really been a problem spot…*

*We have spent some few days really trying to figure some buggy stuff there…*

*The number of artefacts has made it cluttered and complex…"*

Figure 16 shows a summary of participant responses to an interview question on their opinion of the accuracy of the social graph view.

Evidence captured from the empirical study suggests that the social graph can present a useful high level view of the state of a collaboration space including work that *isn't* being done because an entity is not present or is relatively small—see the following comment from 'Boris':

*"…If we have done 'JUnit Test' how come it only (shows) Gizmo.java, Square.java and GizmoModel.java…?*

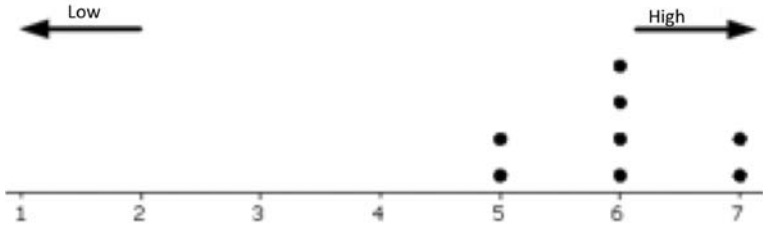



*Figure 16.* Participants level of acceptance of the social graph views during the Gizmoball study.

*Because I know that it should be looking at virtually all of the code…*

*There is something wrong…*

*This tells that there is more work to be done (on) 'JUnit Tests…"*

On the other hand, a couple of participants identified the potential disadvantage that social graphs can quickly become overwhelming as the size of a collaboration space grows.

#### 6.2.4. *Results summary*

The evaluation has demonstrated a number of findings. Firstly, it has shown that it is possible to build and implement a model, based on the identification of developers, tasks and the capture of interaction events (create, update and view) on code artefacts, that can provide contextual awareness in a distributed, collaborative environment. It has been shown that the proposed model worked in practice, during this study, in keeping with its theoretical design. It was shown that often the CRI model appeared to have a more accurate record of relevance than individual developers—this is not really surprising since individuals are missing key information regarding the work efforts of their colleagues, particularly during distributed development. It was shown that the social graph view can provide a useful high level summary of the state of a collaborative project. The main finding is that a model such as CRI can enhance developers' awareness of the state of a collaborative project in a range of ways: what developers, tasks and artefacts are most relevant in particular work contexts, both from an overall effort perspective and from a most recent work perspective. Finally, the capability to 'slide through' the history of a project enabled developers to get a quick and effective overview of how the project has developed from a range of perspectives over time.

## 7. Discussion

The main aim of this work has been to propose and evaluate a model that provides a perception of the relevance and impact of tasks, developers and artefacts associated with a distributed, collaborative software project in a selected work context. The

evaluation study has demonstrated that the features of CRI appear to address many of Gutwin et al.'s elements of workspace awareness—see Table 1, including some of the elements that are not well addressed in other tools. It is argued that the identity, activity level, actions, objects, extent, abilities, and sphere of influence can all, to a significant extent, be inferred from the position of tasks and artefacts on the relevance hierarchy of recent and overall work effort in the CRI work contexts of developers. Similar arguments can be made about task and artefact work context perspectives. The outstanding Gutwin elements that are not addressed are those concerned with the future (intentions and expectations), those to do with changes (which are well addressed by existing configuration management tools) and location.

While the research focuses on a distributed context, the outcome of the evaluation arguably suggests that CRI can be useful in a range of cooperative contexts—local or distributed. For instance, CRI was used by participants in different scenarios within the time/space matrix. This included working at different times within the university campus (different time/same place) or from their homes (different times/different place), as well as working at the same time within the university campus (same time/same place).

The proposed CRI model appears to extend awareness support provided by previous work in a number of respects. CRI provides a more holistic approach to awareness by integrating information relating to *developers*, *artefacts* and associated *tasks*. While awareness information in Ariadne and Hipikat is centred only on developer and code artefacts respectively, Team Tracks, FASTDash and Expertise Browser are centred on relational properties between developers and code artefacts. Furthermore, Mylyn is centred on relational properties between tasks and code artefacts. The evaluation study does suggest that additional awareness information, specific to different work contexts, can be provided by integrating relational properties amongst these three entities types. Secondly, CRI provides collaborative awareness information based on both recent and overall work effort. While recent work effort can be deduced in Mylyn, its implemented degree of interest model does not extend to a collaborative context. Thus, while it is useful to provide awareness of the impact of a task on the state of an artefact, as done in Mylyn, it is arguably more useful for developers to obtain awareness of the relative impact that *all* tasks have had on the state of code artefacts over the project history. Finally, as demonstrated in the study, CRI's relevance based ordering view of entity relevance in the history mode provides the opportunity to replay the evolving relevance of entities over the lifecycle of a software project.

There have been a number of lessons learned from the modelling, implementation and subsequent evaluation of CRI. One of the important lessons learned from the modelling of CRI is that the SOI ratio can be central in revealing a number of latent properties of a collaborative software development process. For instance, a high SOI ratio for a developer may suggest that they are working with many parts of the system and hence central to the development process. Furthermore, if most

developers tend to be associated with a high SOI ratio, then it might imply a shared code ownership development model such as extreme programming. If a task has a high SOI ratio then this can indicate its importance to the development process. On the other hand it might indicate poor task definition and allocation practice—for instance, the task has not been broken down enough or that the development process has not been well segmented. The use of SOI as the basis of a forensic analysis of the design and its development has rich potential for future work.

CRI specifically excluded relations between same type entities e.g. artefact to artefact. It is believed that such awareness information could also be relevant and useful within a cooperative setting. Such relations would provide awareness of the relevance of developers to each other depending on their context of work or the relative interdependence between task or artefact instances. Developer-developer relations have been studied in Ariadne. The outcome suggested that such relations can be used to identify developers who are more likely to be communicating (de Souza et al. 2007).

The study also reveals the need for a more scalable visualisation of the social graph (e.g. fisheye), particularly if it were to be used in real-world applications with potentially thousands of entities and inter-relations.

Finally, CRI does not measure the time developers spend viewing code artefacts. Developers may spend more time on entities which are more important (though there are obvious dangers here such as being interrupted while viewing). It is anticipated that measuring viewing time (within certain limits) and potentially the size of a view event, based on scrolling and mouse movement, may help increase the accuracy of CRI. Another possibility is to distinguish between local updates that that are never committed and updates that are actually made visible to others. The granularity at which interactions are currently recorded is the file level; there may be benefits in focussing on lower level granularity, such as the method level, to identify artefact relevance in more detail.

## 8. Threats to validity

A standard criticism of this kind of university-based research project is the use of students. The best that can be done is to use experienced students working on realistic development projects. The project only lasted 10 weeks, and was only monitored for 6 weeks. Therefore these findings must be treated with caution; however it is still argued that they provide a reasonable indication of the potential strengths and weaknesses of a CRI-like model in real world distributed, collaborative development. A related threat is that the participants had previously had very limited experience of collaborating in groups and this may have impacted their working practices compared to more experienced participants.

Another threat is that CRI did not accurately capture all development data. It is clear that participants did not record all tasks that they worked on and did not

always change task as they changed work context—this is a real challenge for a task-based model such as CRI.

The studies were part of an assessed University course. Participation was entirely voluntary and the lecturer associated with the course (the fourth author) was not involved in any interviews or data analysis. He only saw anonymised data.

The results may have been impacted by the lack of experience of participants with CRI. Again, for pragmatic reasons, participants were only provided with a CRI user guide and a 30 minute tutorial. Some participants may not have developed a sufficient understanding to fully utilise CRI and develop deeper insights into its strengths and weaknesses.

A real threat to CRI usage both in this study and in practice is that it is possible for developers to forcibly increase their relevance in a collaborative space. Developers can easily perform meaningless or routine views and updates that boost their relevance. This is a real danger in any development environment where CRI might be used as the basis of judging individuals, and is a strong reason, along with privacy concerns, why this must not be done. In this study we tried to stress that CRI outputs were not, and should not be, used as the basis to judge the performance of individuals.

Finally, this study was carried out in the context of a forward engineering project. It is believed that CRI offers significant potential benefits when used in reverse engineering or maintenance contexts. Although a few participants hinted at perceived benefits of CRI models in these contexts, little can be deduced about this without further research.

## 9. Conclusions

This paper has presented and evaluated a model intended to enhance contextual awareness in distributed, collaborative software engineering spaces where developers are free to work at any time and in any location. Key results demonstrate that it is possible to derive real time relevance rankings of project entities that exist in collaborative space by monitoring developer interactions. These interactions have been used to derive: an indication of the overall work effort of individual developers in particular work contexts through the history mode as well as their current work through the recent mode; an indication of which tasks and artefacts have consumed most effort over all developers; a history slicing capability that allows a developer in particular work contexts to ‘playback’ the development process; and, a social graph that provides an abstract view, void of context, of the overall state of a project which can help determine potential bottlenecks and the potential implications of deleting artefacts, updating tasks or removing developers from a project.

Empirical evaluation using a small but realistic case study demonstrated that the implementation of the model appeared to work in practice according to the

design. In particular, both SOI—representing the importance of an entity in a collaborative space—and periodic decay—reducing the importance of inactive elements in recent mode—were shown to have a clear impact on the relevance rankings in keeping with the CRI model design.

Investigation of whether the model can support awareness during collaborative development highlighted three areas of strength: a number of examples were identified where it appeared that CRI was maintaining more accurate relevant rankings than individual developers; developers used the history slider to 'replay' project development to help enhance their understanding of who had contributed what at each stage of development and what tasks and artefacts were most relevant throughout the project lifecycle; and, the social graph view of CRI was shown to provide an effective high level summary of a collaborative project—showing what entities were important and also highlighting areas where development may not have been as much as it should have been.

This research has focused on the development and evaluation of a model that can enable collaborators achieve contextual awareness based on tasks, developers and artefacts that are being used to achieve a distributed software project. For a selected task instance, awareness of the relative impact of project developers and code artefacts is provided. Similarly, for a selected code artefact, awareness is provided of the relative impact of project tasks and developers on its current and historical state. Finally, for a selected developer, awareness is provided of the relative impact of tasks and code artefacts on their work context. The CRI relevance model is based on a collaborative perspective rather than an individual one.

Further work should investigate the potential to track important software development artefacts beyond code e.g. requirements, design, tests and maintenance requests. The main reason for focussing on code alone in this initial work was the ease with which code changes could be tracked in the Eclipse architecture. Other areas that that should be explored include: measuring length and extent of artefact viewing; examining improved techniques to capture task creation and change; and, the potential for forensic analysis of the development process based on SOI data. Finally, major ethical considerations exist in the real world use of CRI since it could be abused as the basis of capability judgment and reward structuring. This can be partially addressed by appropriate management attitude and also mechanisms within CRI to allow developers to switch monitors on and off at any stage.

**Acknowledgements**

The authors are extremely grateful to the anonymous students who spent a significant amount of time using CRI in their project work and provided insightful feedback during the evaluation of this work. We are also very grateful to the reviewers for this special issue who provided really useful advice on how to improve the original version of this paper.

## Notes

1. http://www.jazz.net (Verified 04/2009)
2. http://www.eclipse.org/ecf (Verified 02/2007)
3. http://www.eclipse.org/dash (Verified 02/2007)
4. http://www.eclipse.org/mylar (Verified 02/2007)
5. http://www.eclipse.org/equinox (Verified 02/2007)
6. http://www.eclipse.org/emf (Verified 02/2007)
7. Absolute update delta is the positive or negative difference in the number of characters associated with a code artefact before and after an update interaction event.
8. http://jung.sourceforge.net/ (Verified 04/2009)
9. http://www.mit.edu/~6.170/assignments/gizmoball/gizmoball.html (Verified 04/09)

## References

Alexanderson, P. (2004). Peripheral awareness and smooth notification: The use of Natural sounds in process control work. NordiCHI '04. Proceedings of the Third Nordic Conference on Human-computer Interaction. ACM, pp. 281–284.

Biehl, J. T., Czerwinski, M., Smith, G., Robertson, G. G. (2007). Fastdash: A visual dashboard for fostering awareness in software teams. CHI '07. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 1313–1322.

Bly, S. Harrison, S. & Irwin, S. (1993). Media spaces: bringing people together in a video, audio, and computing environment. *Communications ACM, 36*(1), 28–46.

Boland, D., Fitzgerald, B. (2004). Transitioning from a co-located to a globally-distributed software development team: A case study at Analog Devices Inc. Third International Workshop on Global Software Development (GSD 2004), 26th International Conference on Software Engineering, pp. 4–7.

Bolchini, C. Curino, C. Quintarelli, E. Schreiber, F. & Tanca, L. (2007). A data-oriented survey of context models. *ACM SIGMOD Record, 36*(4), 19–26.

Bruegge, B., Dutoit, A., Wolf, T. (2006). Sysiphus: Enabling informal collaboration in global software development, International Conference on Global Software Engineering (ICGSE'06), pp. 139–148.

Busch, P., Richards, D. (2001). Graphically defining articulable tacit knowledge. Selected Papers from the Pan-Sydney Workshop on Visualisation, Australian Computer Society, Inc., pp. 51–60.

Cadiz, J. J., Venolia, G., Jancke, G., Gupta, A. (2001). Sideshow: Providing peripheral awareness of important information. Microsoft Research Technical Report MSR-TR-2001-83.

Cheng, L.-T. de Souza, C. Hupfer, S. Patterson, J. & Ross, S. (2004). Building collaboration into IDEs. *Queue, 1*(9), 40–50.

Chisan, J., Damian, D. (2004). Towards a model of awareness support of software development in GSD. Third International Workshop on Global Software Development (GSD 2004), 26th International Conference on Software Engineering, pp. 28–33.

Cramton, C. (2001). The mutual knowledge problem and its consequences for dispersed collaboration. *Organization Science, 12*(3), 346–371.

Cubranic, D. Gail, M. Singer, J. & Booth, K. (2005). Hipikat: a project memory for software development. *IEEE Transactions on Software Engineering, 31*(6), 446–465.

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

Cutrell, E., Czerwinski, M., Horvitz, E. (2001). Notification, disruption, and memory: Effects of messaging interruptions on memory and performance. Proceedings of Interact 2001, IFIP Conference on Human-Computer Interaction, Tokyo, pp. 263–269.

de Freitas, G. Tait, T. & Huzita, E. (2008). A tool for supporting the communication in distributed software development environment. *Journal of Computer Science and Technology, 8*(2), 118–124.

de Souza, C., Redmiles, D., Mark, G., Penix, J., Sierhuis, M. (2003). Management of Interdependencies in collaborative software development. Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE '03), IEEE Computer Society, pp. 294–303.

de Souza, C., Quirk, S., Trainer, E., Redmiles, D. (2007). Supporting collaborative software development through the visualization of socio-technical dependencies. Proceedings of the 2007 International ACM Conference on Supporting Groupwork (GROUP '07), pp. 147–156.

Deline, R., Khella, A., Czerwinski, M., Robertson, G. (2005). Towards understanding programs through wear-based filtering. In: Proceedings of ACM 2005 Symposium on Software Visualization, ACM, pp. 183–192.

Dey, A. Abowd, G. & Daniel, S. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction, 16*(2), 97–166.

Dix, A., Finlay, J., Abowd, G., Beale, R. (2004). Human-computer Interaction, 3rd ed. Prentice Hall.

Dourish, P., Bellott, V. (1992). Awareness and coordination in shared workspaces. Proceedings of the 1992 ACM Conference on Computer-supported Cooperative Work (CSCW '92), pp. 107–114.

Dragunov, A., Dietterich, T., Johnsrude, K., McLaughlin, M., Li, L., Herlocker, J. (2005). Task tracer: A desktop environment to support multi-tasking knowledge workers. In: Proceedings of the 10th International Conference on Intelligent User Interfaces (IUI '05), San Diego, ACM, pp. 75–82.

Farshchian, B. (2001). Integrating geographically distributed development teams through increased product awareness. *Information Systems, 26*(3), 123–141.

Fritz, T., Murphy, G., Hill, E. (2007). Does a programmer's activity indicate knowledge of code? Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE '07), pp. 341–350.

Froehlich, J., Dourish, P. (2004). Unifying artifacts and activities in a visual tool for distributed software development teams. Proceedings of the 26th International Conference on Software Engineering (ICSE '04), IEEE Computer Society, pp. 387–396.

Gross, T. Stary, C. & Totte, A. (2005). User-centered awareness in computer-supported cooperative work-systems: structured embedding of findings from social sciences. *International Journal of Human-Computer Interaction, 18*(3), 323–360.

Gutwin, C. (1997). Workspace awareness in real-time distributed groupware. PhD Thesis, Department of Computer Science, University of Calgary.

Gutwin, C., Greenberg, S. (1998). Effects of awareness support on groupware usability. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '98), pp. 511–518.

Gutwin, C., Greenberg, S., Roseman, M. (1996). Workspace awareness in real-time distributed groupware: Framework, widgets, and evaluation. Proceedings of HCI on People and Computers XI (HCI '96), Springer-Verlag, pp. 281–298.

Gutwin, C., Penner, R., Schneider, K. (2004). Group awareness in distributed software development. Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (CSCW '04), pp. 72–81.

Hadas, W., Frank, A. (2001). Tacit knowledge: The link between organizational and individual knowledge. European CSCW Workshop Managing Tacit Knowledge.

Hargreaves, E., Damian, D. (2004). Can global software teams learn from military teamwork models? Third International Workshop on Global Software Development (GSD 2004), 26th International Conference on Software Engineering, pp. 21–23.

Heiner, J., Hudson, S., Tanaka, K. (1999). The information percolator: Ambient information display in a decorative object. Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology (UIST '99), pp. 141–148.

Herbsleb, J. (2007). Global software engineering: the future of socio-technical coordination. 2007 Future of Software Engineering (FOSE'07). *IEEE Computer Society*, 188–198.

Horvitz, E., Jacobs, A., Hovel, D. (1999). Attention-sensitive alerting. Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99), pp. 305–331.

Hupfer, S., Cheng, L.-T., Ross, S., Patterson, J. (2004). Introducing collaboration into an application development environment. Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work (CSCW '04), pp. 21–24.

Jazz. (2008). Jazz platform quick reference. Jazz Community Site, IBM Rational Software.

Kantor, M., Redmiles, D. (2001). Creating an infrastructure for ubiquitous awareness. Eight IFIP TC 13 Conference on Human-Computer Interaction (INTERACT 2001), pp. 431–438.

Kaptelinin, V. (2003). UMEA: Translating interaction histories into project contexts. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Ft. Lauderdale, pp. 353–360.

Kersten, M. (2007). Focusing knowledge work with task context. PhD Thesis, University of British Columbia.

Ko, A., DeLine, R., Venolia, G. (2007). Information needs in collocated software development teams. Proceedings of the 29th International Conference on Software Engineering (ICSE '07). *IEEE Computer Society*, 344–353.

Kommeren, R. & Parviainen, P. (2007). Philips experiences in global distributed software development. *Empirical Software Engineering, 12*(6), 647–660.

Latora, V. & Marchiori, M. (2007). A measure of centrality based on network efficiency. *New Journal of Physics, 9*, 188.

McDonald, D., Ackerman, M. (1998). Just talk to me: A field study of expertise location. Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work (CSCW '98), pp. 315–324.

McFarlane, D. (1999). Coordinating the interruption of people in human-computer interaction. Proceedings of the IFIP TC13 Conference on Human Computer Interaction, pp. 295–303.

Mockus, A., Herbsleb, J. (2002). Expertise browser: A quantitative approach to identifying expertise. Proceedings of the 24th International Conference on Software Engineering, pp. 503–512.

Mockus, A. Fielding, R. & Herbsleb, J. (2002). Two case studies of open source software development: apache and mozilla. *ACM Transactions on Software Engineering Methodology, 11* (3), 309–346.

Pacey, M., MacGregor, C. (2001). Auditory cues for monitoring a background process: A comparative evaluation. Proceedings of the IFIP TC13 Conference on Human Computer Interaction (Interact 2001), pp. 174–181.

Pedersen, E. (1998). People presence or room activity supporting peripheral awareness over distance. CHI 98 Conference Summary on Human Factors in Computing Systems, pp. 283–284.

Perry, D. Staudenmayer, N. & Votta, L. (1994). People, organizations, and process improvement. *IEEE Software, 11*(4), 36–45.

Sarma, A., Hoek, A. (2002). *Palantir: Increasing awareness in distributed software development*. International Workshop on Global Software Development (GSD 2002), 24th International Conference on Software Engineering.

Sarma, A., Noroozi, Z., Hoek, A. (2003). Palantir: Raising awareness among configuration management workspaces. Proceedings of the 25th International Conference on Software Engineering (ICSE '03), IEEE Computer Society, pp. 444–454.

Schmidt, K. (2002). The problem with 'awareness': introductory remarks on 'awareness in CSCW'. *Computer Supported Cooperative Work, 11*(3–4), 285–298.

Segal, L. (1995). Designing team workstations: The choreography of teamwork. In: *Local applications of the ecological approach to human-machine systems*. Hillsdale: Erlbaum.

Sillito, J. Murphy, G. & De Volder, K. (2008). Asking and answering questions during a programming change task. *IEEE Transaction Software Engineering, 34*(4), 434–451.

Singh, G. (1999). Guest editor's introduction, media spaces. *IEEE MultiMedia, 6*(2), 18–19.

Storey, M.-A., Cheng, L.-T., Bull, I., Rigby, P. (2006). Shared waypoints and social tagging to support collaboration in software development. Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work (CSCW '06), pp. 195–198.

Teasley, S. Covi, L. Krishnan, M. & Olson, J. (2002). Rapid software development through team collocation. *IEEE Transactions on Software Engineering, 28*(7), 671–683.

Webster. (2006). *Unabridged dictionary.* Springfield: Merriam-Webster.

Weiser, M., Brown, J. (1996). Designing calm technology. PowerGrid Journal, p. 1.

Yew, J., Gibson, F., Teasley, S. (2006). Learning by tagging: Group knowledge formation in a self-organizing learning community. Proceedings of the 7th International Conference on Learning Sciences (ICLS '06), pp. 1010–1011.

Zou, L., Godfrey, M. (2006). An industrial case study of program artifacts viewed during maintenance tasks. Proceedings of the 13th Working Conference on Reverse Engineering, pp. 71–82.