In-Time Role-Specific Notification as Formal Means to Balance Agile Practices in Global Software Development Settings

Dindin Wahyudin[1], Matthias Heindl[2], Benedikt Eckhard[1], Alexander Schatten[1], and Stefan Biffl[1]

[1]Institute of Software Technology
and Interactive Systems
Vienna University of Technology,
Vienna, Austria
{Dindin, Eckhard, Schatten, Biffl}@ifs.tuwien.ac.at

[2]Support Center Configuration Management Siemens Program and Systems Engineering,
Vienna, Austria
{matthias.a.heindl}@siemens.com

Abstract. In global software development (GSD) projects, distributed teams collaborate to deliver high-quality software. Project managers need to control these development projects, which increasingly adopt agile practices. However, in a distributed project a major challenge is to keep all team members aware of recent changes of requirements and project status without providing too little or too much information for each role. In this paper we introduce a framework to define notification for development team members that allows a) measurement of notification effectiveness, efficiency, and cost; b) formalizing key communication in an agile environment; and c) providing a method and a tool to implement communication support. We illustrate, with an example scenario from an industry background, the concept and report results from an initial empirical evaluation. From the evaluation it follows that the concept allows determining and increasing the effectiveness and efficiency of key communication in a global software development project in a sufficiently formal way without compromising the use of agile practices.

Keywords: Software project management, Software process improvement, Methods and tools of software development, Agile practices in global software development, Context-specific notification.

1 Introduction

Today business competition forces highly distributed and global software development (GSD) players to be more responsive and adaptable to uncertainty during development processes (e.g., changes of requirements, technologies

implementation; involvement of partners/subcontractors), especially in novel product development [14].

The Agile Manifesto[1], promised that higher customer satisfaction can be achieved by addressing such uncertainty aspects and delivering working software frequently with shorter timescale. However adoption of agile practices such as daily planning, daily synchronization and daily build [7], [16], requires overall more intensive communication and information exchange among project team members regarding project changes when compared with typical plan driven approach. Especially, in the context of GSD projects, effective communication is an important issue as one has to take into account distant locations and different time zones [9].

Usually, in order to communicate a change in requirements and in other project artifacts, a GSD team member who committed the change notifies other team members in some informal way (e.g., by phone). Such an approach requires extra effort and/or results in loss of information or delay. Another common practice is subscribing by team member to particular tools (e.g., a project manager may subscribe to SVN/CVS to be notified about each check-in performed by his/her developers). Although this approach is a cheaper way of notification, it is often that the target user receives too much information and most of them are out of his current work context or interest.

Hence, to effectively manage such an agile and distributed project one has to address issues specific to agility and distribution, i.e. (a) all team members should be aware of relevant project status (b) information supply should meet the current work context of each role, (now it is hard to measure information supply due to informal way of communication between team members in GSD), and (c) cost and effort of key communications should be reduced. To address these issues, we propose a concept of "in-time role-specific notification". In-time and role-specific means delivering the right information to the right person within his/her current work context (context aware). We define notification in a way that allows measurement of its effectiveness, completeness and correctness. We suggest that such an approach can also be used in the agile context. To address the need for effort and cost reduction we extend the functionality of GSD tools by introducing plug-in integration of the tools to support in-time role-specific notification in GSD settings. Moreover, we present scenarios, based on industrial experience, which illustrate the need for in-time role-specific notification.

The remaining part of the paper is organized as follows: Section 2 describes related work on agile methods adoption in GSD settings and issues important when controlling agile GSD projects. Section 3 introduces the research questions and the concept of "in-time role-specific notification". Section 4 presents scenarios from industry background and later, in Section 5, we provide initial evaluation of the concept. Section 6 discusses the initial evaluation results and compares them with related work. Section 7 concludes and outlines future research on in-time role-specific notification that would be needed to better support collaboration in distributed projects.

.

[1] http://agilemanifesto.org/principles.html (accessed on 15 August 2007)

2. Related Work

Global software development (GSD) projects can benefit from agile practices to react to changing requirements and project circumstances, however to maintain the overview and control of this project extra care has to be taken to maintain the timely communication between distributed teams and team members. Formalization of key communication and supported by proper infrastructure can take away the burden of communication "work" from team members while maintaining communication effectiveness and efficiency. The key question is what kind of communication can be automated and how tools can support such automation

2.1 Agile Methods Adoption in GSD Settings

Boehm and Turner [2] describes balancing agility with discipline such as introducing agile practices in plan-driven GSD projects may provide complementary values derived from both approaches. As the usage of plan-driven GSD methodologies promise access to larger competence developer pool with lower development costs, and work effectiveness due to time zone exploitation [9]. While agile software development offers several benefits for GSD such as adaptation to changing requirements, higher customer satisfaction, rapid releases, and lower defect rates [1]. However, Boehm and Turner also suggest that,
The key success is finding the right balance between agility and discipline within the development process, which will vary from different project to project according to circumstances and risk involved.

Several literatures in Distributed and Global Software Development report experiences of agile methods practices in distributed project settings. Schawber [16] reports a case study in scaling Scrum for large project in an outsourcing company. He created multiple small to medium size Scrum teams to perform shorter Sprint cycle and shorter daily Scrum meeting in order to reduce deliverable time of software product. Other study by Martin Fowler [7] reports extreme programming (XP) adoption in large distributed project in USA. The projects successfully used practices such as continuous integration to reduce problems with integrating the work across multi-site teams, short iteration, and multiple communications. To keep communication between teams effective yet relatively intensive as required by XP, he employs a "team ambassadors" as communication buffer or team representative to interface with other distributed teams. Nisar et al. [13] and Xiaohu [18] report their experiences in adopting extreme programming (XP) in offshore teams collaborating with onshore consumers. The development work is done in offshore teams with tightly involvement of the onshore customer. Xiaohu further explained the main issue for implementing XP practices was to reduce the communication delay and improve communication quality between the customers and the offshore development team.

All these experience reports conclude that applied agile methods (such as XP and Scrum) can benefit distributed development; however, research is needed to address

issues on communication between project team members which is limited and expensive in GSD settings [14].

2.2 The Needs for Formalization in GSD and Agile Contexts

To deliver high quality software, in GSD projects typically multiple distributed teams work on the software development. During collaboration, the team members spend more than 50% development time for communication [15], and about 70% of this time accounted for cooperative activities [17]. Other studies in distributed software development suggests that direct /face to face communication is very important in uncertain software development such as to fill in activity details, fix mistakes and inaccurate prediction, counter measures for the effect of project changes [9], to address coordination and interdependency issues [5]. Therefore, direct communication limitation and breakdown regarding the recent changes in requirement and project status make critical situation in software development processes. From GSD project management point of view it is very important to provide information that should meet the roles expectation in order to keep the team member aware to current requirement changes and status of development artifacts, and help to support the multi-sites collaboration activities. However as direct communication and frequent formal reporting of performance status in GSD is luxury and somehow very limited, hence depict the need for an approach that can significantly reduce the effort and cost of communication.

One approach is tool supported notification exchanges between teams and team members by a network of notification server as proposed by deSouza et al. [5] which benefit collaborative development such as in GSD by managing interdependencies of task and artifacts [4]. They suggested that the event data flowing in the project system network and work tools encapsulate critical information necessary to improve coordination of activities, and communication. An event and its attributes (such as requirement changes, automatic build) can represent stakeholder interactions or communication during a software project execution. However although notification server propose automation of some key communication in GSD, however deSouza et al. did not mention how to formalize such notification (e.g. notification specification, rules, and model) which is necessary to bring discipline to the automated notification generation processes. We need to formalize in order to reduce cost, effort, and risk such as delay which is necessary in GSD context. On the other hand we should not formalize everything because it reduces the flexibility which is necessary for certain aspects in the project, too costly and not practical. Therefore, it is necessary to balance formalization and flexibility using cost-benefit analysis.

3. The Concept of In-Time Role-Specific Notification to Balance Agile Practices in GSD settings

This section motivates the research issues and the proposed concept of in-time role-specific notification to address our research question. We also envision the GSD tool

support for collaboration of GSD team members, by introducing the integration of plug-in which allows the information exchanges as part of team member work tools.

3.1 Current Reality of Agile-Global Software Development

To examine the cause and effect logic behind current agile adoption in GSD settings, we employed Current Reality Tree suggested (CRT) in Goldratt's theory of constraints [3] as problem analysis tool. CRT begins with identifying the undesirable effects we see in today practices in GSD and trace back to a few root causes, or a single core problem. Later we can select what to improve that will have the greatest positive effect to agile-GSD development. Figure 1 illustrates the current reality of typical Agile-GSD project, the lower level represent the root cause, while the upper level signify undesirable effects.

The rectangles represent entities such as core problem, root cause, effect and undesirable effect, while an ellipse represents AND operator and arrow signify the impact direction.

We grouped the entities into 4 groups to avoid confusion of reader due to number of entity represented in this model. The first group (box I) represents typical characteristics of global software development process as suggested by many literatures in distributed and global software engineering domain such as in [9] [10] and [12]. The distributed participants with different culture, different language may have impact in the content of information being exchanged, as the result team members sometimes have misinterpretation or misunderstanding of the conveyed message, on the other hand the distant location and different time zone, make face-to face communication such as daily synchronization more expensive, worth more effort and hard to coordinate.

The second group (box II), express the need for more intensive communication among team members due to their work dependencies and changing in project environment (e.g. requirement and artifact changes), however as direct communication is infrequent in GSD context, in group 3 (box III) reveals that the communication of changes are committed either in informal way or by subscribing to work tools as described in introduction.

The fourth group (box IV) illustrates the undesirable effects due to current communication methods in Agile-GSD project. As the team member missed vital information this will cause lack of awareness of important project status concerning his work context. This information deficiency may lead team member to perform a task with flaw direction, and increase the possibility of versioning problem, rework and delay. The tool subscribed method, often shower a team member with information spam; consequently he needs more effort to select which information is relevant for his current work context, which sometimes can be a frustrating task. Both of these undesirable effects (lack of awareness and more reading effort) will decrease the developer motivation, and eventually will have larger impact to overall development process.

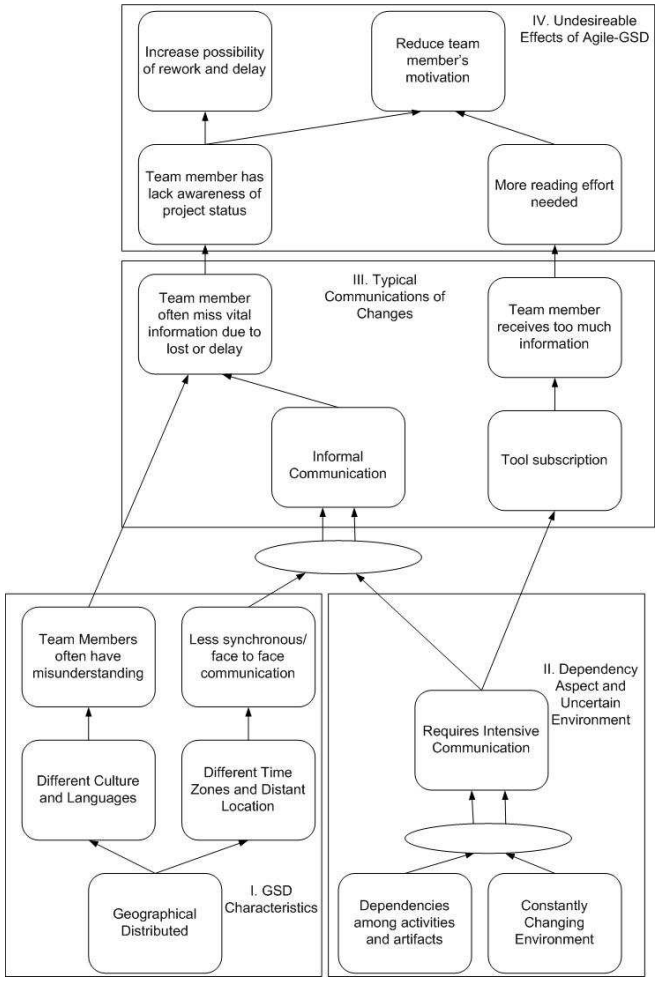

Fig 1 Current Reality Tree of Agile-GSD Project, undesirables' effects such as delay and motivation degradation of developer can be derived from (a) higher effort and higher cost to retrieve information of project status and (b) the poor quality of conveyed information

3.2 Research Issues

Based on Current Reality Tree in section 3.1, direct communications between team members are extensively required by agile methods but missing in GSD due to cost and effort allocation as the result of geographical distribution. Hence, the agile

practices adoption in GSD settings will face greater challenge to traditional GSD project.

This hybrid Agile-GSD projects requires a novel method which promise cost and effort reduction in information exchanges between GSD team members. One solution is to automate the communication supported by tools as described in related work, however the challenge is how much formalization of communication is enough, as in agile context, we still need to maintain some aspect of flexibility due to project uncertainty. Therefore in this paper we propose two research questions which are:

(a) What kind of communication can be automated during development processes?
(b) How can tools support such automation?

To address these research issues, we introduce a framework to define notification for development team member which allows:

- Measurement of notification effectiveness, and effort. To determine the effectiveness and effort of key communication and the value of notification in global software development project in formal way without compromising the use of agile practices.
- Formalizing key communication in an agile environment. We provide example scenarios from industry background to explain the concept of formalization of key communication in form of notification exchanges between GSD team members
- To provide method and tool support to implement communication support. Tool support to increase the effectiveness and efficiency of key communication in global software development project in formal way without compromising the use of agile practices. We also perform initial empirical evaluation from one of the scenarios as the proof of concept

3.3 In-Time Role-Specific Notification: Definition and Concept

In global software development setting, collaboration between team members from multiple sites is essential. Figure 2 illustrates the typical work and collaboration in GSD, here a team member first assigned a role within specified work context, e.g. project manager, developer, and tester, in certain location. In agile practices, role assignment may not be a static position, for example a team member can be assigned as software architect at the beginning of the project, later he can act as a developer once the designs and specifications completed.

Based on current assigned role, a team member should perform some activities or task typically supported by a set of work tools to deliver software artifacts. Every change of software artifacts can be considered as an event which is also typically recorded in the work tool where the event happened. Typically works in GSD environment are not stand alone; a team member may have dependencies of artifact developed by other team members. Based on these dependencies, a team member needs to be notified for certain events represent changes of the artifact. Hence he should specify a notification and retrieve the correct notification in time. To receive information which is delayed, partial or not relevant will reduce a team member work

performance and also may affect other development tasks performed by other team members who depend on his deliverables, as the consequences the project may face some risky condition such as version conflict, release delay, and quality reduction of to-be delivered software.

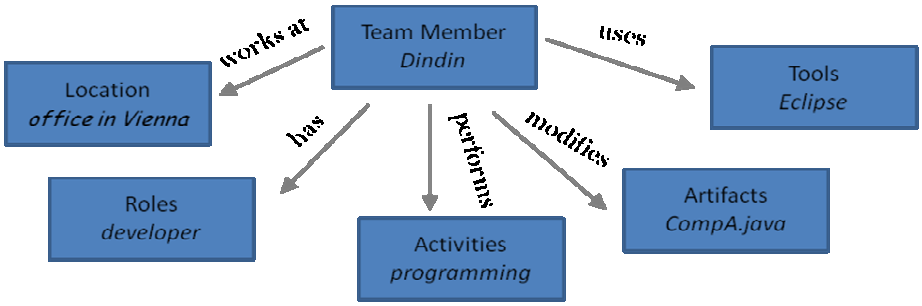

Fig 2. GSD Team Member role, and the need for notification based on his current work dependencies

3.3.1. In-Time Role-Specific Notification Definition

We define a notification as an object that collects information about status changes, errors, early warnings and other time-relevant project status information to be presented to target roles. A notification can be triggered by events, correlation of events or measurement data passing pre-defined threshold during project execution. For example a tester needs to be notified when a developer closed a development ticket (ticket closing events), which required to be tested before adding the new code-set to current body of code of to-be delivered software.

The meaning of in-time aims to localized notification to meet the user expectation of particular timely effective information awareness, as he may only concern to be notified for relevant project status changes in particular time of deliverable (immediately, or summarized) and within his current work context (e.g. what I'm doing now, with whom/what my work connected with) and consider other out of context and delayed notification as information waste or noise. Meanwhile the role-specific term means to deliver the notification to the right notification user.

3.3.2. Notification Specification: How Much Formalization is enough?

The intention to specify a notification is to provide correct notification for target user in formal way. In our context a notification derived from selected key communication between team members. We use three selection criteria to select which key communications are worth enough for formalization and automation by tool support, such as: (a) the key communication is significantly important to support collaboration of GSD team members according to circumstances in development processes; (b)

repetitive or frequently occurrences in larger part of development life cycle(e.g. hours and daily occurrence); and (c) data transmitted has significant probability of risks, such as to become lost, error, impartial or delayed in manual way of transmission. Table 1 provides some examples of key communication selection for formalization and automation, these key communications passed the first selection criteria as considered important to support collaboration in GSD.

Based on our Industry background we assumed the values of the selection criteria for each key communication as described in table 1, communication of changes of requirements and components are feasible for formalization and automation. After selection of key communication, the next step is to specify what kind of notification should be provided for target user. The specification also needed to localize the scope of formalization as we only need to formalize several relevant aspect of key communication, and leave the rest to stay flexible.

Table 1 Examples of Key Communication Selection in Agile-GSD settings

Key Communications	**Roles Involved**	**Frequency of Occurrence**	**Risk of loss and delay**	**Need for Formalization**
Changes in requirements	Project manager, developer, tester	Medium	High	Yes
Requirement traces	Project manager, developer	High	High	Yes
Component changes	Developer	High	High	Yes
Fix defects in code	Developer	High	Low	No
Fill in plan	Project manager, Technical leader, QA	Low	Low	No

There are several elements of key communication that should be formalized to specify a notification such as: processes performed during communication task (e.g. impact analysis of requirement change, decision approval for requirement change), all roles involved in information exchange (e.g. project manager as target user, and developer as events provider in changing requirement scenario, see section 4), data transmitted during communication (e.g. traceability information of requirement), distributed events to publish-subscribed the notification (e.g. source code element changes published by the developer to trigger notification consumed by the project manager), and delay allowance of notification represents the time between artifact/requirement changes and capturing of notification by target user.

The next step of formalization is to model the work-flow to trigger the notification from abovementioned elements. We can use a process centric model such as IDEF0 or extension of UML proposed by Penker and Eriksson [6]. In this paper we use Penker and Eriksson extension to illustrate notification for proposed scenario in section 4, as this extension offers more capability in expressing and formalization of notification by mitigating the ambiguity often associated with narrative specifications or scenarios.

3.3.3. Rules Definition and Notification Escalation

In order to deliver and present notification in-time and within context of particular roles, those we need to formulate the notification rule. The syntax to formulate notification rules consists of the following parts: Notify <whom> in <what way> (e.g., e-mail, SMS, entry in change log) by <when> (e.g., immediately; batch every hour/day) concerning <in which context> (e.g. implement particular task, managing certain project) due to <change event> (e.g. requirement changes, component changes).

Whom: list of persons, roles, or groups. Change can be any observable or derived event or state change regarding an artifact or project state, e.g., some expected event did not happen during the given time window. While context can be any task that assigned to the user, and selected as his current work focus or need to be notified when certain changes occur. For example in requirement changes scenario as described in section 4, a notification for John a developer if particular requirement changed, can be described as: *Notify* John in his Eclipse workspace, *immediately* concerning his task T1 to implement requirement R1 *due to* changes of Requirement R1.

If a condition can not be handled by the system based on the rule set, and then the issue should be escalate to a sufficiently competent role that can provide a reasonable decision. For example in continuous integration build scenario as applied in XP adoption in distributed off-shore project by [13], in this scenario typically a developer will automatic build his code before send it to the repository. For each build he will get notification of build status either success or broken build, however in certain situation such as in an approaching deadline, if a developer experiences too many build failures which is risky situation as there is possibility of he may not deliverer his task on-time. This issue should be escalated to the project manager, so then he can take some appropriate counter measures to address such risk. This example reveals the benefit of notification as early warning sign that may be used to complement information from developer, and to reduce delay for information dissemination.

3.3.4. Derived Measurement

The value of in-time role-specific notification influenced by several factors that can be measured such as:

- **Effort** (*E*) is an accumulation of work hours to prepare (*Tpr*), to process (*Tpc*) and to create notification of changes (*Tcr*). Integrated tools' plug-ins supported notification should be able to reduce significantly the overall effort allocated by the GSD team members.

  Here we can formulate effort as $E = Tpr + Tpc + Tc$ (1)
- **Correct Notification** (*CN*) is number of notifications created and transmitted to target user within the scope of pre-defined specification.
- **False Positives** (*FP*) is number of notifications determined not in the scope of correct specified notifications for a target user.
- **False Negatives** (*FN*) indicates number of notifications determined in the scope of correct specified notifications but do not reach target users

- **Effectiveness** (*EF*) is number of correct notifications (*CN*) in proportion to all generated notifications (*GN*) for a specified notification set (*SN*). We expect that tool support increase notification transmission effectiveness as expected in agile context.

  Here we can formulate: $EF = CN/GN$ (2)

  $GN = CN + FP + EF$ (3)

These factors are considered as general measurement of value of notification and should be applicable to almost every scenario in GSD and Agile context. We can use this measurement for balancing agility and formalism in notification, by comparing the results of several alternatives of delivering the notification. For example in scenario described in section 5 we can compare the effort needed by two traditional alternatives of requirement tracing (with Excel and Req.Pro) with our proposed plug-in alternatives, if the results reveal that plug-in offers significant effort reduction with respect to cost to develop such plug-in, then GSD project manager may need to consider to apply such alternatives, on the other hand if the effort reduction is considered not worth enough compare to plug-in development's cost and other set-up effort, then PM may just discard the idea of the plug-in approach.

3.4 Tool Integration and Support

In this work we propose the presentation of notifications in the user interface of a tool routinely used by the target role in order to reduce team member refusal due to "another-tool-syndrome". Tool support mostly consists of tool sets (requirements, development, configuration, tracking and test tools) that can interact in principle providing the basis for redundancy-free, consistent storage of data and exchange of data between tools (via tools interfaces). Tool-based notification also promise cost-reduction which make information exchange can be much more affordable in GSD context, moreover a comprehensive tool support is needed to enable consistent, error-free, and up-to-date information exchange in a GSD context. The interfacing between tools using plug-ins can support information exchange of events recorded by tools during project execution.

Tool support allows to implement notifications using a rule engine, which can be captured and processed into meaningful information or notification using complex events processing techniques [11] e.g., a correlated events processor (CEP). Figure 2 illustrates how GSD work tools can be connected to an enterprise service BUS (ESB) using plug-ins (plug-ins integration). These plug-ins captured particular events occur in the tools, and publish the events to the ESB in XML format. These events later captured and processed by the CEP, and if a measurement threshold or certain rules apposite with an event or correlated events then a notification (also in XML format) is triggered and published to the ESB. Some subscribed tools' plug-ins consumes the notification and presents it to the user as part of their work tools. In summary these plug-ins act as notification or event publisher and as notification subscriber/consumer, and can be configured dynamically by the user (GUI-based configuration for a general user and an event selection pattern language for more sophisticated user).

In continuous integration practices, some activity triggering automation (e.g., automatic build and automatic test) benefit agile software development by reducing effort and time for certain tasks, these activities also may trigger events that considerable worth noting for roles involved in development process such as build status, build error. Correlating these events (e.g. correlating build failures for particular task in certain period of time) can derive time-relevant status information such as quality degradation and quality prediction of software product.

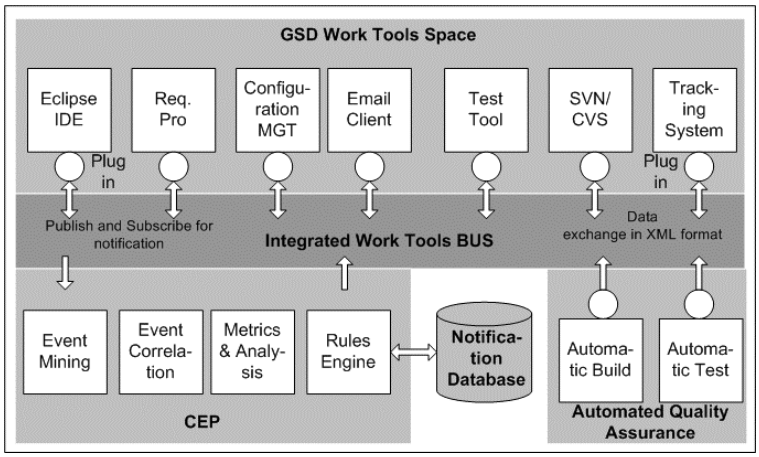

Fig 3. Integrated tool support for In-time Role-Specific Notification in Agile-GSD settings

4 Example Scenario

The following scenario illustrates how in-time role specific notification provides support to current global software development especially when agile practices introduced to the development processes. We provide initial empirical evaluation based on the result of implementation of the scenario. In this scenario several distributed team members such as a project manager on site A who has responsibility in requirement management which later implemented by the developer from site B. The project manager manages the requirement in requirement management tool such as Requisite Pro, while the developers use IDE tool such as Eclipse as their development platform. If a change of requirement X arrives from the customer, accordingly the project leader performs impact analysis, in order to decide whether such change should be implemented or not (see figure 4), he needs to know the current status from developer who assigned to implement the requirement X, and what kind of impact may derived by this change e.g. risks and cost.

Typically developers in GSD create some *Excel* matrices to store traceability information of implementation status which can be considered as ad-hoc approach or systematically draw a license for the project's requirements management tool (e.g. Req.Pro). Project manager then manually assesses this information, performs the

analysis and creates an impact report as the basis of decision approval whether a change should be implemented or not. Based on this scenario, we can define the *impact analysis* as the processes, *project manager* and *developer* as roles involved in this process, and *traceability information* transmitted by the developer as key communication to be automated. Let's assume that we extended the functionality of tools used by developer (Eclipse) and project manager (Req.Pro) with plug-ins to provide interface between the two tools. In this extended scenario whenever a developer committed some changes in his code set, the Eclipse plug-in will store this event and correlate these changes to relevant requirement (Req.X), and automatically publish a requirement traces notification (N) consists of developer ID, source code elements that have been changed, date of changes and its correlation with Req.X. The Req.Pro plug-in which subscribed for this type of notification then captures notification N from the integrated work tools BUS (see figure 3), and present this notification in the project manager's Req.Pro interface.

Despite of cost and effort reduction, as result of automation, this approach can benefit distributed project controlling as a project managers can decide if a requirement should be changed although development has already been started. They can also easily get in contact with the developer that is working on it to ask him about the current progress or potential implications. As the consequences notification may enhance the impact analysis processes in order to avoid potentially dangerous situation such as to put barrier to the developer against risky or unnecessary changes (as in Scrum before a Sprint release).

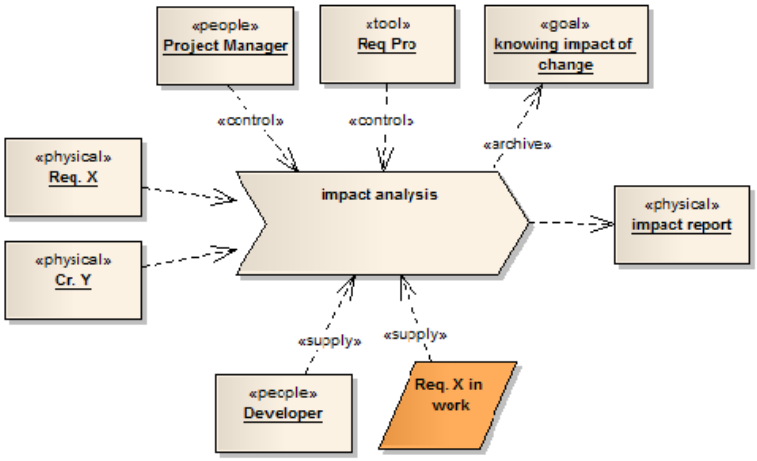

Fig 4. Impact Analysis is performed by project manager based on requirement traceability information from the developer

5 Initial Empirical Evaluations

We performed an initial feasibility study of the integrated tool plug-in support for scenario of requirement traces to support impact analysis as described. In order to compare the plug-in-based approach with other alternatives, we observed a set of projects at Siemens PSE to evaluate the tracing efforts, correctness and completeness of each alternative. The projects were different in domain (transportation systems, telecommunication, etc.), but similar in size: medium size projects, with 2 to 4 sites (e.g., Austria, Romania, Slovakia), and between 10 and 60 team members.

The number of requirements of each project is between 150 and 300; number of source code methods to be traced range from 6000 to 13000, while number or traces per requirements is between 150 and 300. Based on these project setting factors we compared the effort to trace requirements to source code methods, the completeness and correctness of traces for the tracing alternatives described in section 4. For more detail information and scenario of evaluation can be found in Heindl et. al [8].

Comparison of the 3 alternatives of requirement tracing, reveal that using plug-in alternatives for tracing requirement may significantly reduce the effort of developer teams and increasing completeness and correctness of tracing. Heindl et al, also reported such improvement lead to higher developer motivation, as developer will have more awareness of changes in requirement, lower effort to trace the requirement and more confidence of correctness of trace information, which also reduce possibility of delay or rework.

Table 2 Comparison of Tracing Effort and Tracing Qualities, *source* Heindl et al. [8]

	Effort for tracing (in working hours)		Tracing Qualities	
	For 150 requirements	For 300 requirements	Correctness (%)	False Positives (%)
Ad-hoc	450 to 1350	900 to 2700	5% to 30%	5-10%
Systematic	50 to 167	99 to 334	20% to 40%	10%
Continuous/ Plug in	8 to 26	16 to 53	60% to 75%	5%

In this paper we compare three alternatives of requirement tracing, and to investigate the continuous tracing approach using in-time role-specific notification concept on the effort and quality of traces. However as reported by Heindl et. al, this approach will have greater benefit for medium to large projects, as for smaller projects, the tracing effort might be too high compare to traditional ad-hoc tracing. Automation of notification in this scenario also has to consider the amount of investment needed especially in project with a low number of requirements and requirement changes.

We use requirement tracing scenario for our initial evaluation because we believe that changing of requirements is the most prominent factor in current software development which need more attentions from the development teams.

6 Discussion

From related work, we can conclude that agile practices adoption in GSD settings may provide several benefits needed by current industry. However one challenge is to provide a means of communication and information exchanges between team members concerning occurrence of changes. Referring to our initial research questions, distributed project needs to define some key communications which is feasible for automation in order to reduce cost and effort. In our initial feasibility study we selected traceability of requirement changes as the key communication that can be automated. The integration of plug-in for developer's tool (Eclipse) and project manager's tool (Req.Pro), provide an interface between the tools, which allows automating this key communication.

The framework also allows measurement of the value of notification, as in our initial empirical study we found that integration of tool support significantly reduces the effort for requirement tracing compared to more expensive time consuming alternatives (e.g. using Excel and Requisite Pro) which are commonly used in current GSD projects. However the evaluation of the concept from other context of agile-distributed development such as different development process scenarios and measuring it's the impact to overall development performance will be further work.

7 Conclusions

In the paper we proposed a concept of role-specific and context-aware notification supported by integrated tools and oriented towards distributed projects. The goal was to complement current distributed project controlling mechanisms and to address communication issues associated with application of agile practices in GSD settings.

Formalization and automation of some key communications between team members in a form of notification may provide benefits such as cost and effort reduction but seems limited to GSD settings. Moreover, we believe that such notification will provide GSD team members with more timely and context-aware information on project status changes. Our initial empirical evaluation provided promising results. However, we would like to perform similar evaluation in the industrial setting with larger size of development team.

Acknowledgments. We would like to thank Franz Reinisch from Siemens PSE Austria and Prof. A Min Tjoa from IFS TU Wien for their contributions to the paper. The paper has been partly supported by The Technology-Grant-South-East-Asia No. 1242/BAMO/2005 Financed by ASIA-Uninet. More details on In-time Role-Specific Notification can be found in our technical report, available at http://qse.ifs.tuwien.ac.at/publications.htm.

References

1. Boehm, B.: Get ready for agile methods, with care. Computer 35(1) (2002) 64–69
2. Boehm, B., Turner, R.: Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2003)
3. Dettmer, H,: Goldratt's Theory of Constraints: A System Approach to Continuous Improvement. Quality Press (1997)
4. de Souza, C., Redmiles, D., Mark, G., Penix, J., Sierhuis, M.: Management of interdependencies in collaborative software development. In: International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003. (2003) 294–303
5. de Souza, C., Basaveswara, S., Redmiles, D.: Supporting global software development with event notification servers. In: the ICSE 2002 International Workshop on Global Software Development. (2002)
6. Eriksson, H.E., Penker, M.: Business Modeling With UML: Business Patterns at Work. John Wiley & Sons, Inc., New York, NY, USA (1998)
7. Fowler, M.: Using agile process with offshore development. http://www.martinfowler.com/articles/agileOffshore.html (June 2007)
8. Heindl, M. Reisnich F., Biffl, S. : Integrated Developer Tool Support for More Efficient Requirements Tracing and Change Impact Analysis,. Technical Report. Institute f. Software Technology and Interactive System, Vienna University of Technology (2007)
9. Herbsleb, J., Moitra, D.: Global software development. Software, IEEE 18(2) (2001) 16–20
10. Herbsleb, J.D., Paulish, D.J., Bass, M.: Global software development at Siemens: experience from nine projects. In: ICSE '05: Proceedings of the 27th international conference on Software engineering. (2005) 524–533 xxx
11. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison Wesley (2002)
12. Mockus, A., Herbsleb, J.: Challenges of global software development. In: Seventh International Software Metrics Symposium, 2001. METRICS 2001. Proceedings.. (2001) 182–184
13. Nisar, M., Hameed, T.: Agile methods handlfing offshore software development issues. In: 8th International Multitopic Conference, 2004. Proceedings of INMIC 2004.. (2004) 417–422
14. Paasivaara, M., Lassenius, C.: Could global software development benefit from agile methods? International Conference on Global Software Development (2006) 109–113
15. Perry, D.E., Staudenmayer, N., Votta, L.G.: People, organizations, and process improvement. IEEE Softw. 11(4) (1994) 36–45
16. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice Hall PTR, Upper Saddle River, NJ, USA (2001)
17. Vessey, I., Sravanapudi, A.P.: Case tools as collaborative support technologies. Communication of ACM 38(1) (1995) 83–95
18. Xiaohu, Y., Bin, X., Zhijun, H., Maddineni, S.: Extreme programming in global software development. In: Canadian Conference on Electrical and Computer Engineering, 2004.. Volume 4. (2004) 1845–1848 Vol.4