In-Time Project Status Notification for All Team Members in Global Software Development as Part of Their Work Environments

Dindin Wahyudin[1] Matthias Heindl[2] Ronald Berger[3] Alexander Schatten[1] Stefan Biffl[1]
[1]*Institute of Software Technology and Interactive Systems, Vienna University of Technology, Favoritenstrasse 9-11/188, A-1040, Vienna, Austria*
{dindin, schatten, biffl}@ifs.tuwien.ac.at
[2]*Siemens Program and Systems Engineering (PSE), Austria,* [3]*Frequentis GmbH, Austria*

Abstract

In global software development (GSD) projects, all team members (e.g., project manager, quality assurance, technical leader, and developer) can benefit from being aware on the project status regarding their area of responsibility. While there is considerable attention on providing project overview data and tool support to the project manager, there is only weak support for the other roles, resulting in higher project risk in a GSD project due to less opportunity for informal communication. In this paper we suggest the needs and propose a concept to allow keeping all relevant roles informed using in-time notification on significant project events. Team members can subscribe to specific notification services provided by project infrastructure. In a typical usage scenario these notification services promise to provide information for more effective and efficient change impact analysis in concurrently evolving software development artifacts.

Index Terms — Data-driven management of distributed development projects, Software Project Monitoring, Strategies for distributed controlling, Role-based in-time project status notification.

1. Introduction

Typical characteristics of global software development (GSD) projects (e.g., geographical distribution and cultural differences of team members) bring with them challenges such as higher communication complexity, more complicated collaboration interdependencies, and more risk in knowledge management, management and technical issues [4]. Each challenge demands appropriate approaches from project management and increases the need for better monitoring of multi-site development projects. In a successful GSD project the project manager (PM) typically has as a top priority to periodically monitor overall GSD project performance based on (human) progress report data and sometimes facilitated by an integrated tool support or a project monitoring cockpit. However, a highly distributed environment demands effective collaboration and communication among the team members to allow delivering good quality software. Hence, project members such as testers, technical leader, and developers also need to be kept informed and notified for certain information and events which are relevant to their roles' objectives in timely manner and provide basis for in-time decision making. While a PM may need summarized reports on project performance and software quality status, a quality assurance (QA) person may want to monitor detailed reports test performance over the time to determine the quality status of software artifacts before an approaching release; a developer should receive immediate feedback if a change in his work causes a quality problem with other (concurrently evolving) components of the software product. In this paper we adopt the concept of role-specific in-time notification on the status of project artifacts supported by an event-driven monitoring infrastructure. We extend this concept with providing the team members with relevant notifications in the set of tools they typically use, e.g., as part of the software development environment rather than in a separate management or collaboration tool. We argue that by providing such notification well represented as part of team member's work tools will better support collaboration among the team members during GSD project execution.

2. Related Work

This section provides an overview of related work on performance measurement and monitoring as well as recent experiences with designing project monitoring cockpits for distributed software development.

2.1. Performance Measurement and Monitoring in Distributed Projects

Software project monitoring performed by people is common practice in GSD projects; a common strategy is to gather and derive data on project performance and achievements from periodic progress reports submitted by team members and traces of artifacts' attributes against the project plan (usually at a milestone completion) [6].

To assess the project status, observers often want to balance progress monitoring at milestones with in-between milestones monitoring, i.e., by directly gathering data (informally) from team members during the development process. Herbsleb and Moitra [3] suggest such informal communication to be very important in software development, e.g., to fill in activity details, fix mistakes and inaccurate prediction, and counter measures for the effect of these changes. Nevertheless the high degree of distribution of GSD projects complicates the coordination of these tasks.

The development processes in distributed software development typically generate two classes of observable project performance data: product-relevant artifact data (e.g., number of modules, function points, defects per KSLOC, number of document pages) and time-relevant event data (e.g., number of subversion check in, trend line in email conversations, and automatic build results). The first group of data represents the deliverable of development process therefore very useful to keep project milestones on track.

In a distributed software development context, an event (and its attributes) can represent stakeholder interactions or process state changes. Monitoring events during project execution can be especially helpful if human-based reports are suspected to be unreliable, incomplete, or inconsistent with reality [4] as event data provide extra evidence of project performance between milestones, e.g., the pattern of number of tests passed divided by number of test executed by test tools to derive current test performance and quality improvement/decrease before a component release. Event monitoring has successfully been adopted in business process monitoring as dynamic business environments have been forcing many organizations to employ more sensitive and flexible systems in order to be more responsive for capturing time-sensitive business risks and opportunities [7] such as in stock market monitoring systems.

Similar to GSD, the event data flowing in the project system network encapsulate critical information necessary to improve coordination of activities, and communication [8]. However, a major challenge is to define which events are relevant for project measurement objectives from a stream of events during project execution, and how to better process collected events into meaningful measures or notification. In recent work [9] we proposed how to collect, correlate and analyze event data to provide measurement of project community contributions in global open source project development, and discern healthy from risky projects.

2.2. Project Monitoring Cockpit Concepts

Large distributed projects are typically supported by integrated monitoring cockpit software to outline the project portfolio in order to assess status of the software project and early warning in order to take necessary actions against certain risk conditions (e.g., decreasing quality of key artifacts, severe bug unresolved for too long). Typically due to the large amount of data obtained from multiple development sites, project executives need integrated tool support such as project control center or monitoring cockpit. In the cockpit project status achievement (i.e. artifacts data) and activities are visualized in unified way [1] or in aggregate form of report, hence a project manager can have an outlook of the project more efficiently, and retrieve detailed information only when he needs reasoning for certain conditions.

To successfully conduct a GSD project, the collaboration of all team members is necessary. Hence current focus of monitoring cockpit on project manager needs should accordingly be extended to all collaborating project members (e.g., developers, technical leaders, and quality assurance). As to support their specific role in a GSD project, each team member requires assessing project status aspects in a timely manner, which potentially increases the complexity of data collection and provision and thus needs comprehensive tool support for defining measurement needs, filtering and aggregation rules, and finally providing correct data in an appropriate user interface. A key challenge is how to provide appropriate notifications for a specific role efficiently and to tackle the "yet-another-tool" syndrome, as a team member may refuse to use an additional tool, e.g., a requirements management tool to capture and maintain traces [2].

One solution approach is to use for GSD a notification server suggested by de Souza et al. [8]. In a notification server, the basic events (e.g. artifact changes) represent component interaction and stakeholder activities during project execution coming from informa-

tion sources (e.g. work tools) to the notification server (connected to a network of collocated notification servers), which ensures the delivery of these events to all interested information consumers (e.g., a developer at a different site) using a publish/subscribe schema. The event-based model of a notification server offers several advantages in supporting collaboration in GSD: (1) flexibility due to decoupling notification providers and consumers; (2) integration among software development tools; (3) internet/global scale of event notification, which enables the construction of internet distribution application; and (4) support of informal communication among different roles of team members. However, de Souza et al. do not mention explicitly how the notification should be presented to the consumer, as their scenarios cover the basic event for notification but not how to derive more meaningful information and notification with the aggregation and correlation of events [5].

3. Role-Specific Status Notification

In this paper we propose extensions to the notification server concept: 1) we define a notification as an object that collects together information about change state, errors, early warning and other time-relevant project status information and communicates it to the presentation of particular user; 2) a notification can be triggered not only by basic events but by derived events, e.g., by correlation of related events, or measurement data passing a given threshold during project execution; 3) presentation of notifications in the user interface of a tool routinely used by the target role in time.

3.1. Definition of notification rules

The syntax to formulate notification rules consists of the following parts: <*whom*> to notify <*in what way*> (e.g., e-mail, SMS, entry in change log) <*when*> (e.g., immediately; batch every hour/day) due to <*change event*>.

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 (see example rules in section 4).

Escalation of Notification: If a condition can not be handled by the system based on the rule set, then the issue should be escalated to a sufficiently competent role that can provide a reasonable decision.

3.2. Tool Support Mechanism

Tool support allows to implement notifications using a rule engine providing as a tool to capture and process the complex event stream into meaningful information or notification using tool based complex events processing techniques [5] e.g., a correlated events processor (CEP). Figure 1 illustrates how GSD work tools can be connected to an enterprise service bus (ESB) using a publish/subscribe mechanism. Each tool publishes certain events and consumes notifications based on standard rules and events configured dynamically by the user (GUI-based configuration for a general user and an event selection pattern language for more sophisticated user).

An event is published to the ESB by work tools or activity triggering automation (i.e., automatic build and test). The CEP captures these events and further processes (transforms, correlates, filters, aggregates, and measures) based on a rule set and eventually derives notifications, which may be consumed by subscribed users (see Table 1 for examples of the events and derived notifications). In Table 1, “X” represents a correlation between an event with a derived notification, for example to have a notification of quality improvement of an assigned task to a particular developer is derived from measuring how many correlated issues encountered during automatic build (i.e., build warning, and automatic build send result of particular code change submission) and automatic test (i.e., unit test resulted test case coverage, and a quality check for particular code change are reported as successfully tested).

Notification Filtering: Role-specific notification means that a user may define about which events he wants to be notified or wants to notify defined roles. However, an important challenge is to create an effective “notification spam guard” as part of CEP rule implementation, so users can focus on relevant information instead of getting uncontrolled notification bombardment. Another approach is to provide an interactive tuning mechanism where a user can adjust the scope and level of information detail of notification delivery dynamically.

Benefits: This approach can support collaboration in GSD by providing the following benefits: (1) lower entry hurdle for team members to access project information, (2) faster, more reliable information flow, and, (3) reduction of delay and avoidable rework due to lost or late information.

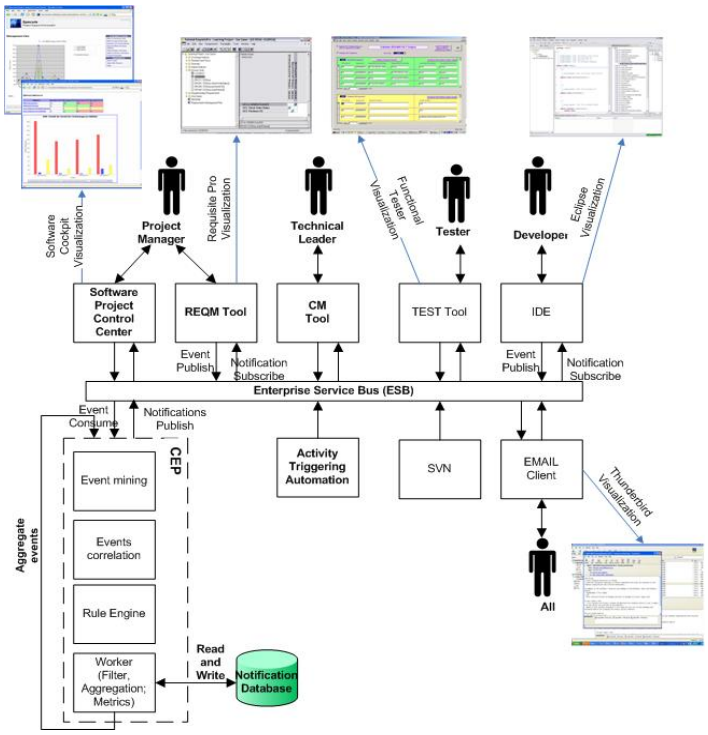

Figure 1 Project Status Notification Infrastructure.

(2) User interface: e.g., GUI to define standard or to select for general user and event selection pattern for sophisticated user, urgent notifications are sent via email to user email client or by SMS. Extension of work tools for notification visualization, and defining rules in usual work environment i.e. plug-in for Eclipse IDE or a REQM tool.

Table 1 Events Published by GSD Work Tools and Derived Notifications.

		GSD Tools								
		REQM		Ticket Tracking System			Activity Triggering Automation			
	Event Published	new requirement inserted	requirement changed	New change requests	State of changes of Ticket	Check in of a Ticket	Build Warning	Build results	Uni test Coverage results	Quality Checker Result
Derived Notifications	Requierement changes	X	X							
	Developer Contribution				X	X		X		
	Quality Improvement						X	X	X	X
	Change request backlog			X	X					
	Closure time				X					

For data analysis, the current infrastructure design covers simple scenarios in GSD project, however for more complex scenarios in-time notification may need to be supported by approaches which enable complex analytical queries in rapid time such as OLAP.

4. Scenarios for Role-Specific Notification

The following typical scenario in a distributed software project illustrates benefits for role-specific notifications. In the "continuous integrated build" paradigm[1], a developer typically runs the production build before committing his change set to the subversion (e.g. SVN); if a build succeeds, he deploys the code on the application server and runs the test suite using a quality checker to check the code quality. For each commit, the developer needs to get notified is something wrong with the code or there is a need for improvement. An SQL-like rule for developer build automation scenario could be:

```
if AUTOMATIC BUILD B05 FAILED then NOTIFY (Stephan Gates, the developer) VIA (Eclipse IDE) and SEND FAILED BUILD CASES.
```

After a task or ticket is closed (because of either enhancement or defect) a series of tests are performed by the test team to check the code quality. Normally, the tester will regularly check the ticket tracking systems to find out what development tasks are finished and ready for testing which is time consuming task. The idea is to send notification whenever a ticket is closed containing a ticket case data (e.g., ticket information, code change set, test coverage) he rule for tester notification could look like:

```
if A TICKET CLOSED then NOTIFY (Susanne Warmick, the tester) VIA (Email Client) and SEND TICKET CASE
```

Meanwhile a technical leader assigned to monitor actual progress for individual tasks of a developer, and need to obtain report of total number of issues without details for each change code. Later he gives feedback to developer such as unit test covered by the change code, quality improved/decreased by the committed change and quality guidelines fulfillment. Typically the technical leader does not have the time to check all the changes manually in subversion and the quality checker. The idea is to provide an aggregate report containing the total number of issue without details for each change code prior to tickets closing. The rule for such notification is:

```
if TICKET T02 CLOSED then NOTIFY (Peter Simon, the technical leader) VIA (email client)and SEND TICKET TO2 SUMMARIZED ISSUE REPORT.
```

The project is monitored by a project manager who wants to improve the software product quality and to get a warning if development is late. Typically the project manager periodically scans the ticketing system and quality improvement reports from the technical leader.

[1] http://www.martinfowler.com/articles/contionousintegration.htm

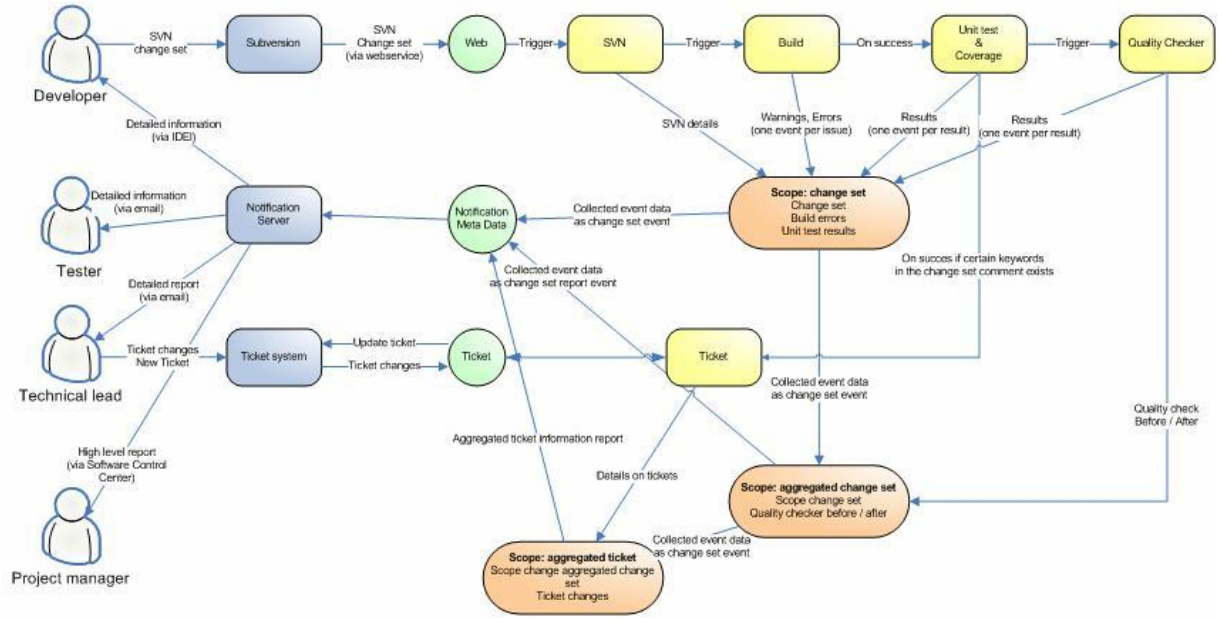

Figure 2 Notifications in Automated Build Scenario

However, this task is also time consuming as he requires to manually processing obtained data and may result of decision delay. One solution is to periodically send an automatic notification containing the quality status of a work package (a series of tickets), and current achievement report (i.e., as a burn down chart) to his software control center. The rule could look like:

```
For Each Month OF TICKET SERIES TXX
DEVELOPMENT NOTIFY (Jack Rabbit, the project
manager) VIA (SOFTWARE CONTROL CENTRE) and
SEND SUMMARY OF TXX QUALITY IMPROVEMENT.
```

In this scenario each role observed several types of notifications for different purposes according to their role objectives. Providing such notifications and reports without tool support is a very time-consuming and an error-prone task. The idea is to process events published by developer automation tools (activity triggering automation) into notification for role-specific users and present the notification as part of their work tool as illustrated in Figure 2.

6. Conclusion and Future Work

In this paper we proposed a concept of an event-driven monitoring approach and role-specific notification tool support. Our concept offers to complement the current project monitoring model in GSD in order to obtain extra evidence and in-time project status notification to support all team members in conducting their tasks in more efficient ways. However, in application scenarios major challenges for future work were identified: a) how to formulate in a team effective rules as the basis of notifications, b) how to guide event monitoring and rule definition that the retrieved data really makes sense for data-driven management; c) how much effort seems reasonable to spend on creating and maintaining the rules; and d) the need for empirical evaluation of the concept derived from some measurement to indicate the effectiveness, completeness, and correctness of the notification.

References

[1] Froehlich, J., and Dourish, P., Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams, Proc. ICSE, 2004, IEEE, 387-396

[2] Heindl, M., Biffl, S., Risk Management with Enhanced Tracing of Requirements Rationale in Highly Distributed Projects, Proc. 1st Int. Workshop on Global Software Development for the Practitioner, ICSE, IEEE 2006

[3] Herbsleb, J., Moitra, D.. Global software development. IEEE Software, 18(2), March/April 2001, pp. 16–20

[4] Keil, M., Smith, H., Pawlowski, S., Jin, L.. "Why Didn't Somebody Tell Me?" Climate, Information Asymmetry, and Bad News about Troubled Projects. ACM SIGMIS Database, 35, 2 , spring, 2004, 65-84.

[5] Luckham, D. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, Addison Wesley, 2002.

[6] Philips, J. IT Project Management: On Track from Start to Finish, Mc Graw Hill, 2002.

[7] Schiefer, J., Seufert, A. Management and Controlling of Time-Sensitive Business Processes with Sense & Respond, In Proc. CIMCA 2005, IEEE, 77-82

[8] de Souza, C., Basaveswara, S., Redmiles D., Supporting Global Software Development with Event Notification Servers, Int. Workshop on Global Software Development, In Proc. ICSE 2002, IEEE.

[9] Wahyudin, D., Schatten, A., Mustofa, K., Biffl, S., Tjoa, A. Introducing Health Perspective In Open Source Web-Engineering Software Projects, Based On Project Data Analysis, Proc. Int. Conf. on Information Integration and Web Based Appl. Services, IIWAS, 2006.