Traceability-Based Change Awareness

Jonas Helming, Maximilian Koegel, Helmut Naughton,
Joern David, and Aleksandar Shterev

Technical University Munich, Department for Computer Science,
Chair for Applied Software Engineering,
85748 Garching, Munich
{helming,koegel,naughton,david,shterevg}@in.tum.de

Abstract. Many tools in software engineering projects support the visualization and collaborative modification of custom sets of artifacts. This includes tools for requirements engineering, UML tools for design, project management tools, developer tools and many more. A key factor for success in software engineering projects is the collective understanding of changes applied to these artifacts. To support this, there are several strategies to automatically notify project participants about relevant changes. Known strategies are limited to a fixed set of artifacts and/or make no use of traceability information to supply change notifications. This paper proposes a change notification approach based on traceability in a unified model and building upon operation-based change tracking. The unified model explicitly combines system specification models and project management models into one fully traceable model. To show the benefit of our approach we compare it to related approaches in a case study.

Keywords: Change awareness, traceability, unified model, operation-based, notification.

1 Introduction and Related Work

A common technique to handle the complexity in software development projects is the use of different models. On the one hand, models such as requirement models and detailed specifications are used to describe the system under construction on different levels of abstraction. We call this the *system model*. On the other hand, there is the *project model* [7] containing tasks lists, schedules, or the organizational structure. All these do not describe the system, but the project itself. In the course of a software project, all models constantly evolve over time. A change in one model often triggers a change in another. For example, a change in the functional requirements most likely affects the work break down structure.

Change awareness is the ability to keep up with changes that were made to development documents and artifacts. It is difficult to achieve this without computer assistance, especially for complex systems [10]. As a consequence, most tools, which store software engineering models, offer some degree of support for change awareness. To notify project participants only about changes relevant to them, change notification strategies are required. Existing approaches provide notification strategies based on either the system and project model (e.g. [3], [5], [11], [13], [15]), based on source code (e.g. [1], [12]) or even both (e.g. [4], [9]). Unfortunately, the effects of changes crossing the boundaries between the system model and the project model have been studied a lot less. A comparison done by Storey et. al. [14] showed that none of the 12 tools included in the study made significant use of project documentation (e.g. requirements, tests, design or architecture).

This paper proposes a novel change notification approach to notify project participants about such relevant changes based on traceability. The approach was implemented and evaluated based on a CASE-Tool called UNICASE [3]. UNICASE provides a unified model, which explicitly combines the system model and the project model into one model stored in one repository. The underlying unified model enables model-based notifications: As even the user is part of the unified model, notifications can be generated based on the context-rich and highly traceable model. For example, a user can be notified about direct or indirect changes on artifacts he is currently working on. To show the benefit of our approach, we compare it to related approaches in a case study.

2 Traceability-Based Change Notification

Our suggested change awareness approach uses links between system model and project model to determine which notifications are relevant for project participants. For example if a project participant is working on a task, which refers to a functional requirement, he might be interested in any change pertaining to this functional requirement. Furthermore he also might be interested in changes related to the detailing use case of the requirement as well as changes in requirements that refine the requirement he is working on. The proposed change notification strategy would trace these links and notify the user about changes on all mentioned model elements.
In the example (see Figure 1), a change of the functional requirement by *UserA* would lead to a notification of *UserB*, because he or she is working on a task related to the functional requirement. Starting from the task the change notification strategy follows a trace of length one. Therefore we will call this type of change notification trace-based change notification of length one.

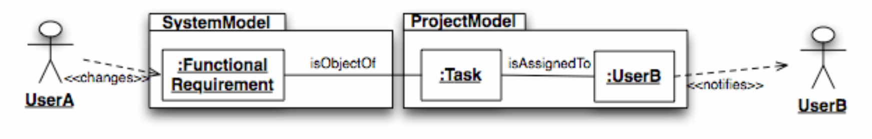


Fig. 1. Notification workflow (custom UML diagram)

The exclusive use of trace-based notifications of length one would ignore the fact that there are dependencies inside the system model, which require change propagation. If there was a detailing use case for the functional requirement in the previous example, *UserB* should be also notified about changes on that use case. UNICASE provides a concept called *opening link taxonomy* [6], which determines which links require such change propagation. The opening link taxonomy allows us to find all model elements that are influencing a given model element in a way that the model element in question cannot be entirely completed without completing these. In other words, the opening links allow us to calculate the transitive closure on model elements a user is working on.

This leads to trace-based change notifications of length two or more. We will compare different lengths later in the evaluation in chapter 3.

In an application context the notifications are generated whenever the user receives changes from the central repository to synchronize the local model instance. The changes in UNICASE are represented as so-called operations [8]. An operation describes one atomic change on a model such as assigning a task to user. Using operation-based change tracking to generate notifications results in two major benefits: performance and time-order preservation. The changes do not have to be calculated and are in the order in which they occurred in time [15]. This means the notifications can be generated efficiently and are ordered by the time at which they occurred. From the list of given operations a set of notifications is derived and presented to the user in a list oriented view.

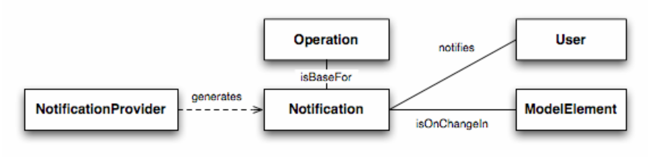


Fig. 2. TraceNotificationProvider (UML class diagram)

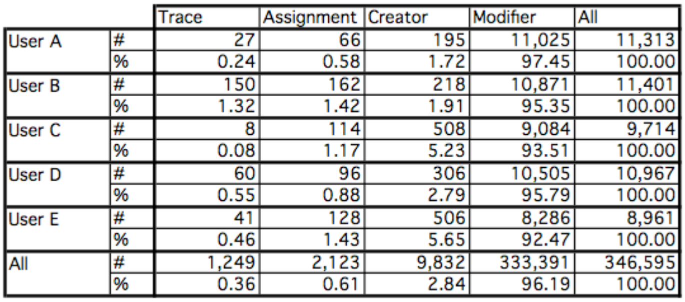
A *NotificationProvider* generates notifications based on a certain strategy (see figure 2). Each *Notification* is targeted at a specific *User*, describes a change on a certain *ModelElement* and was derived from a given *Operation*.

3 Evaluation

To evaluate our approach we conducted a case study in which users manually rated change notifications according to their personal relevance. To allow for comparison with other approaches, we implemented the most common automatic approaches for providing change notifications:

- Modifier-based notifications (e.g. [5]): A user is notified on every change in any artifact he or she has previously modified.
- Creator-based notifications (e.g. [23]): A user is notified on every change on any artifacts he or she created.
- Assignment notifications (e.g. [25]): A user is notified on the assignment of a task and on any subsequent change to an assigned task.

We did not include notification strategies that rely on manual selection of the artifacts the user wants to be notified about, since they are not comparable with automatic approaches and have a different objective. The data for our case study was collected in a project named DOLLI2 (Distributed Online Logistics and Location Infrastructure) at a major European airport. More than 20 developers worked on the project for about five months. All modeling was performed in UNICASE resulting in a comprehensive project model that consisted of about 1000 model elements and a history of more than 600 versions. Using the traceability-based notification provider as well as the three other notification providers, we generated notifications for the DOLLI2 project data post mortem. To evaluate the quality of the notifications generated by the different providers, we interviewed 5 project participants on 12 of their notifications randomly selected for each provider. Figure 3 shows an overview of the statistics of the number of notifications and their percentage for the different users and for the different providers.



		Trace	Assignment	Creator	Modifier	All
User A	#	27	66	195	11,025	11,313
	%	0.24	0.58	1.72	97.45	100.00
User B	#	150	162	218	10,871	11,401
	%	1.32	1.42	1.91	95.35	100.00
User C	#	8	114	508	9,084	9,714
	%	0.08	1.17	5.23	93.51	100.00
User D	#	60	96	306	10,505	10,967
	%	0.55	0.88	2.79	95.79	100.00
User E	#	41	128	506	8,286	8,961
	%	0.46	1.43	5.65	92.47	100.00
All	#	1,249	2,123	9,832	333,391	346,595
	%	0.36	0.61	2.84	96.19	100.00

Fig. 3. Notification Overview Statistics Table

Out of the three notification providers we implemented for comparison, we expected the assignment provider to be rated highest based on the assumption, that everybody should be notified about changes in his or her directly assigned tasks. Further, we expected the creator provider to be rated second best, as the creation of an artifact usually implies some kind of ownership. As the modifier-based provider also includes creator-based notification, but generates a significantly higher number of notifications, it was expected to be rated third.

The table shows that the trace provider generates relatively few notifications, whereas other providers generate up to 1000 times more notifications for the same data. In a practical appliance the number of change notifications provided should be as low as feasible in order not to overwhelm the notified users. As the traceability-based notification provider generates a low number of total notifications, but also produces notifications not covered by other providers (because of the traces between system and project model), we claim it is useful if it is rated better than at least one of the existing notification providers. Our hypothesis was that the TaskTrace provider would result in a significantly higher rating than the Modifier provider, since we measured the mean ratings as $m_{Modifier} = 2.53$ and $m_{TaskTrace} = 3.19$ based on $n_1=60$ and $n_2=47$ items in each category, respectively. The negative correlation $\varrho(L,R)=-0.127$ (Pearson) between the trace length L and the rating R showed a tendency of shorter trace lengths causing higher ratings. Therefore, we only used those ratings that concerned notifications with a trace length of less or equal than two (resulting in 47 instead of 60 items). We used the non-parametric Kolmogorov-Smirnov test to analyze the difference in the user rating. To perform this kind of test, two (discrete) empirical distribution functions F and F' – one for the Modifier rating and one for the TaskTrace rating – had to be computed from the data sample. The supremum of the differences of the distribution functions $d_n=\sup_x |F_n(x) - F_n'(x)|$ for $n:=n_2=47$ was $d_{47}=0.2341 > c(\alpha=5\%; n) = 1.36 / n^{1/2} = 0.198$, which means that the maximum of the distances exceeds the acceptable constant $c(\alpha=5\%; n)$ corresponding to the 5% level. Thus the null-hypothesis has to be rejected on the 5% level of significance.

Summing up this statistical result, we showed that our traceability-based notification provider performs better than the modifier-based notification provider. Regarding the significantly lower number of created notifications, we claim our traceability-based notification provider to be useful in practical appliance.

References

1. Biehl, J.T., Czerwinski, M., Smith, G., Robertson, G.G.: FASTDash: a visual dashboard for fostering awareness in software teams. In: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 1313–1322. ACM, New York (2007)
2. Bruegge, B., Creighton, O., Helming, J., Kögel, M.: Unicase - an Ecosystem for Unified Software Engineering Research Tools. In: Third IEEE International Conference on Global Software Engineering, ICGSE 2007 (2008)
3. Carroll, J.M., Neale, D.C., Isenhour, P.L., Rosson, M.B., McCrickard, D.S.: Notification and awareness: synchronizing task-oriented collaborative activity. International Journal of Human-Computer Studies 58(5), 605–632 (2003)
4. Cheng, L., Hupfer, S., Ross, S., Patterson, J.: Jazzing up Eclipse with collaborative tools. In: Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange, pp. 45–49. ACM, New York (2003)
5. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Enhancing an artefact management system with traceability recovery features. In: Proceedings of 20th IEEE International Conference on Software Maintenance, 2004, pp. 306–315 (2004)
6. Helming, J.: Integrating Software Lifecycle Models into a uniform Software Engineering Model. In: Software Engineering Conference - Workshop Proceedings, Gesellschaft für Informatik, pp. 157–164 (2008)
7. Helming, J., Koegel, M., Naughton, H.: Towards Traceability from Project Management to System Models. In: ICSE 2009, 5th International Workshop on Traceability in Emerging Forms of Software Engineering Vancouver, British Columbia, Canada (2009)
8. Koegel, M.: Towards Software Configuration Management for Unified Models. In: ICSE 2009, CVSM 2009: Proceedings of the CVSM workshop, pp. 19–24. ACM, New York (2009)
9. LaToza, T.D., Venolia, G., DeLine, R.: Maintaining mental models: a study of developer work habits. In: Proceedings of the 28th international conference on Software engineering, pp. 492–501. ACM, New York (2006)
10. Luqi: A graph model for software evolution. IEEE Transactions on Software Engineering 16(8), 917–927 (1990)
11. Papadopoulou, S., Norrie, M.: How a structured document model can support awareness in collaborative authoring. In: International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2007. CollaborateCom 2007, pp. 117–126 (2007)
12. Sarma, A., Noroozi, Z., van der Hoek, A.: Palantir: raising awareness among configuration management workspaces. In: Proceedings of 25th International Conference on Software Engineering, 2003, pp. 444–454 (2003)
13. Sinha, V., Sengupta, B., Chandra, S.: Enabling Collaboration in Distributed Requirements Management. IEEE Softw. 23(5), 52–61 (2006)
14. Storey, M.D., Čubranić, D., German, D.: On the use of visualization to support awareness of human activities in software development: a survey and a framework. In: Proceedings of the ACM symposium on Software visualization, pp. 193–202. ACM, New York (2005)
15. Tam, J., Greenberg, S.: A framework for asynchronous change awareness in collaborative documents and workspaces. Int. J. Hum.-Comput. Stud. 64(7), 583–598 (2006)
16. Koegel, M.: Operation-based conflict detection and resolution. In: ICSE 2009, CVSM 2009: Proceedings of the CVSM workshop. ACM, New York (2009)