

# Traceability-based Change Awareness

J. Helming, M. Kögel, H. Naughton, J. David, A. Shterev, B. Bruegge

<sup>1</sup> Technical University Munich, Department for Computer Science  
Chair for Applied Software Engineering  
85748 Garching (Munich)  
{helming, koegel, naughton, david, shterev, bruegge}@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 Motivation

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. Therefore we will call the entirety of such models the *system model*. On the other hand, there are models in a software project such as tasks lists, schedules, or the organizational structure, which do not describe the system, but the project itself. We will therefore call them the *project model* [14]. In the course of a software project, all models – the system model as well as the project model – constantly evolve over time. This is especially true when applying agile methodologies [1]. This implies that the model of the system under construction as well as the project plan and organization is subject to change. 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.

“Not only does a developer have to keep up with what changes have occurred, they might also have to determine if these changes are relevant to them and if so how they will deal with these changes. If the project is even moderately large, this process can

be quite challenging.” [27] 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 [21]. As a consequence, most tools, which store software engineering models, offer some degree of support for change awareness. UML tools for example visualize the differences between distinct versions graphically. Task management systems provide notifications, e.g. a new task being assigned to a project participant creates a notification informing him or her about this. Furthermore, there are several research approaches to better support change awareness in various contexts [2, 5, 10, 12, 15, 17, 24, 26, 28]. To notify project participants only about changes relevant to them, change notification strategies are required. For example bug-tracking systems notify users only about changes on bug reports, which are assigned to them.

In typical software development projects the relevant models are stored in different tools, thus aggravating the problem. Therefore every tool has to provide its own change awareness support. Consequently the change awareness support of every tool can only access a limited set of information to create notifications for the project participants. If there is a change in the requirements specification, project participants who are currently implementing this requirement cannot be notified automatically. This paper proposes a novel change notification approach to notify project participants about such relevant changes. The approach was implemented and evaluated based on a CASE-Tool called UNICASE [3]. On the one hand UNICASE provides a unified model, which explicitly combines the system model and the project model into one model stored in one repository. On the other hand UNICASE employs operation-based change tracking which allows us to generate more accurate notifications preserving the changes original time order with only small performance requirements.

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.

The paper is organized as follows. In the next chapter, we present the related work and approaches to the problem of change awareness. Chapter 3 deals with the prerequisites for our approach, namely the unified model of UNICASE and its operation-based change tracking. Based on these prerequisites, chapter 4 details the traceability-based notification strategy. This strategy is evaluated in chapter 5 in form of a case study. The paper then concludes and lists future work in chapter 6.

## **2 Related work**

Being aware of all the changes that occur in a large, often globally distributed software development project is a problem for many of today’s developers [15]. Understanding rationale and being aware of changes, together with efficiently switching tasks and finding code duplicates, is most important to developers. This was elicited by a large-scale survey by LaToza et al. [18]. If tools do not support

developer communication, teams face communication and coordination breakdowns [9].

Many approaches related to change awareness operate on the level of source code or collaborative documents. Tools such as Palantir [24] and Hipikat [8] address change awareness when working collaboratively on source-code. Palantir uses information available from a developer's workspace to show team members which source-files are currently being edited. This helps to notify other developers on impending changes based on a fine-grained event structure. Hipikat offers links to newsgroups, bug-reports, etc. based on the current context of the developer by maintaining a "group memory" of the project. This helps newcomers to a software project to become productive faster, using the information contained in the recommended artifacts.

Papadopoulou et. al. [22, 23] suggest remedies for change awareness in collaboratively authored documents by use of document structure and edit profiling. Tam and Greenberg [28] developed a theoretical framework for tracking and displaying change information on collaborative documents. Carroll et. al. [5] introduce the concept of "activity awareness" and relate it to awareness problems witnessed in the use of LiNC (Learning in Networked Communities) project.

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. [26] showed that none of the 12 tools included in the study made significant use of project documentation (e.g. requirements, tests, design or architecture).

While Jazz [6] does include artifacts such as requirements, change notifications are only calculated on the level of authorship or team authorship. Change notifications are provided as feeds, which interested parties can subscribe to.

FASTDash [1] provides a "visual dashboard" for developers in order to foster team activity awareness by providing a quick visualization of which files or code parts are being viewed or changed by other developers of the team.

Damian et. al. [10] propose a theoretical feature set for geographically distributed requirements management. This provides specific relationships between people and requirements, tracking states of requirements and allowing for impact analysis.

ADAMS by de Lucia et. al. [11] can also provide event-based change awareness using traceability links between software artifacts. This is done using subscriptions on a per artifact basis, with the need for detailing the distance for indirect notifications and the required change event level beforehand.

Kwan et. al. [17] propose to solve the problem of requirements change awareness by using a "requirements-centered social network diagram", grouping project participants together in "requirements-centered teams", whose members work on the same requirement. By tracking communication patterns, the RCSN is updated to reflect the current state of information flows. This can be used for example to inform developers of impending changes to requirements.

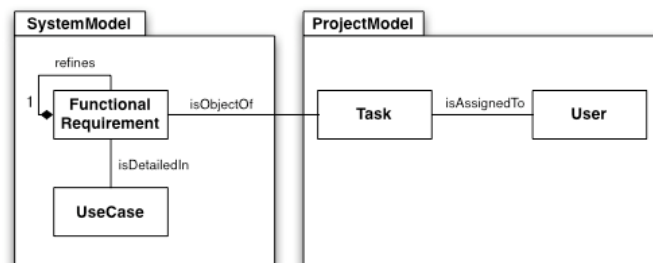
EGRET [25] by IBM Research manages requirements for global software development teams. The tool notifies owners of related requirements on requirement changes, but it does not trace to development related tasks that might be impacted by such a change.

### 3 Prerequisites

In this section, we will describe the prerequisites for our approach to supply change notifications. At first we present the unified model, which provides the traceability links we rely on for the notification generation. Secondly we give an overview of the change tracking system we use for determining the changes that occurred in the model.

#### 3.1 Unified Model

In this section we describe the unified model our approach is based on. The unified model is implemented in a CASE-tool called UNICASE [3], the successor system of Sysiphus [4]. UNICASE provides a unified and flexible repository, which is able to store arbitrary types of models. From a repository point of view any model is a graph that consists of a number of nodes that are connected by edges. In the following we will refer to the nodes as model elements and to the edges as model links or traceability links. Model elements can either be part of the system model or the project model. In other words system model elements like requirements or UML elements are part of the same unified model and stored in the same repository as project model elements such as tasks or users. Model elements from both these models can be directly linked with each other. The use of a unified model enables a central notification provider to support change awareness on the entirety of those elements. The notification provider can use the links between the different parts of the model to create relevant notifications for the user.



**Figure 1: Example part of the unified model (UML class diagram)**

Figure 1 shows an example of such links in UNICASE. Links can exist within the project model, e.g. a project participant can be linked to a task he which is assigned to him or her. Also links connect model elements within the system model, e.g. a functional requirement can be linked to a detailing use case or one requirement can refine another. Most importantly however links can also exist between system and project model, for example a task can be linked to a functional requirement. This implies the task represents some work necessary to fulfill the requirement. The

traceability within system or project model is referred to as vertical traceability while the traceability from the system to the project model is called horizontal traceability.

The unified model is not limited to the classes shown in Figure 1 but can contain artifacts belonging to any known software modeling technique. For example, the unified model in UNICASE also includes UML classes, packages, components, deployment nodes, work packages, groups, meetings and bug reports to name only a few. It can easily be extended to incorporate even more.

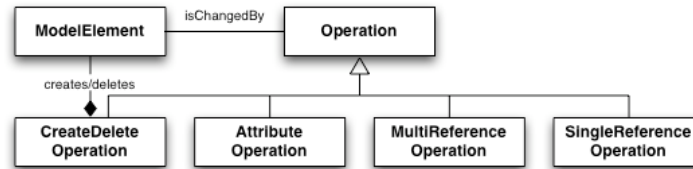
Our suggested change awareness approach uses links between system model and project model to determine which notifications are relevant for project participants. In the previous example, a project participant is working on a task, which refers to a functional requirement. Consequently he might be interested in any change pertaining to this functional requirement. Furthermore he also might be interested in changes relating 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. Section 4 describes the tracing in detail.

### 3.2 Operation-based change tracking

To create change awareness notifications, change tracking is required. Most approaches for supporting change awareness rely on so-called Software Configuration Management (SCM) systems. These systems track the history of artifacts and provide methods to determine differences between versions of an artifact – widely known as diffing. Existing SCM systems are operating state-based. This means the SCM system will derive the changes on a model *after* they occur by comparing the previous state of the model with the current state. In contrast we rely on a changed-based versioning system [16] for our approach. The difference to state-based SCM systems is that a change-based system records the changes *while* they occur. This eliminates the need for diffing, preserves the original time order of the changes and derives more accurate changes.

In this section we shortly introduce the concept of operation-based change tracking. We use the operation-based change tracking in UNICASE to derive the changes in the model that we rely on for notification generation. All changes, which are applied to an instance of the unified model, are recorded. These recorded changes are called operations and are subsequently used to synchronize (commit/update) different clients with a central repository server. On every commit and update the client will have the respective operations at hand and can use them for further processing such as change awareness notification generation. Also for each operation the SCM system can provide important context information such as author and date.

Figure 2 shows the taxonomy of operations. Each *Operation* is changing exactly one *ModelElement*. A *CreateDeleteOperation* creates or deletes a model element from the model. An *AttributeOperation* changes a simple attribute such as an integer or string. *SingleReferenceOperation* and *MultiReferenceOperation* change a link to another model element with a multiplicity of one or greater than one respectively.



**Figure 2: Operation taxonomy (UML class diagram)**

Using operation-based change tracking to generate notifications results in three major benefits:

- (1) Complexity in time: Since the operation-based approach does not require diffing prior to generating notifications, the cost for generating notifications is linear in the *number of changes*. For other approaches the time complexity is up to  $O(n^2)$  in the *number of model elements*. [7, 20] In bigger projects with a big number of model elements expensive diffing is a limiting factor for generating notifications in real time.
- (2) Accuracy: Text-based diffing on graph-like models is widely known to perform poorly and it would be more than difficult to implement a notification provider based on it. Other approaches for model-based diffing such as EMF Compare [29] exhibit problems in detecting composite changes such as a delete or refactoring as being composite. This results in many changes being reported instead of one change. Also both approaches suffer from a problem where two changes partially mask one another [22].
- (3) Time order preservation: The recorded operations are ordered in time and therefore resemble the order in which the users performed the respective changes. To understand an activity performed by someone else the timing is important information. With approaches other than operation-based or change-based tracking, the order is lost in diffing however and cannot be recovered.

#### **4 A Traceability-Based Notification Strategy**

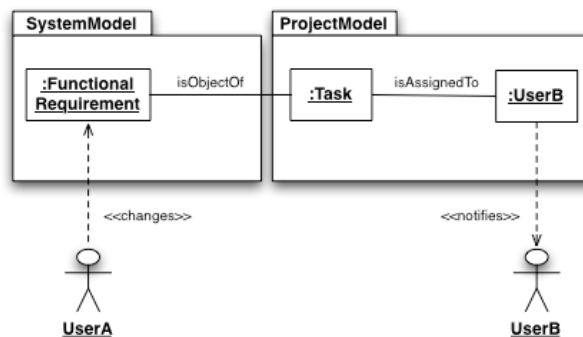
Notifying users about changes helps users to be aware of changes. However it is prohibitive because of the sheer amount of such notifications to provide the user with a notification about any change in a given project. This problem is even increased in large scale-projects and projects with make extensive use of models. Therefore it is necessary to provide context-sensitive change notifications that are customized for every single user.



**Figure 3: Notifications and Providers (UML class diagram)**

Figure 3 shows notifications and their notification providers. A *NotificationProvider* generates notifications based on a certain algorithm and strategy. Each *Notification* is targeted at a specific *User* and describes a change on a certain *ModelElement*. In the following, we will describe how our traceability-based notification provider as a subclass of *NotificationProvider* works in detail.

As outlined in section 3, the unified model can be used to represent tasks and their assignment to users. The unified model also provides information on the objective of a task, that is the object the work item is to be executed on or is concerned with. A typical example is an implementation task being associated with a functional requirement. Combining the information from these two associations allows us to track which user is working on which part of the system model and therefore the system (see Figure 4). In the example, 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.

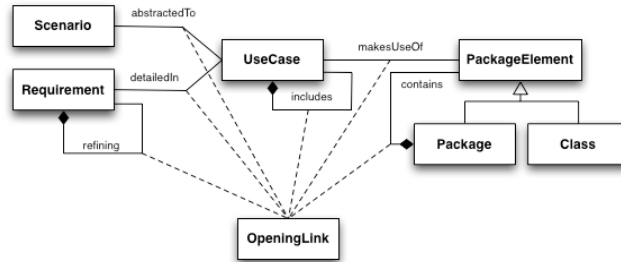


**Figure 4: Notification workflow (custom UML diagram)**

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. For example, a change on a use case that is detailing a functional requirement should lead to a notification for users working on the functional requirement.

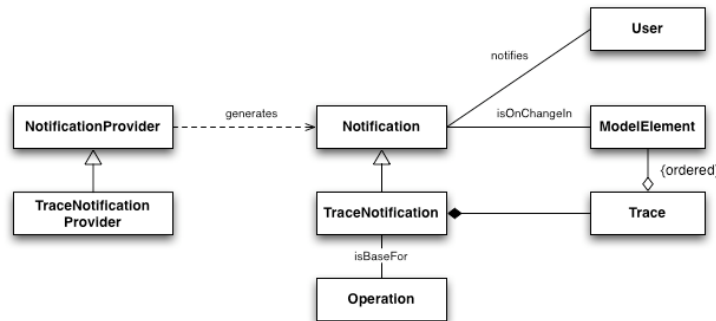
UNICASE provides a concept called *opening link taxonomy* [13] (see Figure 5), which determines which links require such change propagation. The opening link taxonomy allows us to find all model elements that are influencing a given model element in a way that the model element in question cannot be 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.



**Figure 5: Opening link taxonomy (UML class diagram)**

In the example from Figure 4, the opening link taxonomy defines the link between functional requirement and the detailing use case as so-called “opening link”. The trace-based notification strategy follows these links. As a consequence, a change by *UserA* on the use case would lead to a notification of *UserB* who is working on the functional requirement detailed by the use case. Starting at the task, the change notification strategy follows a trace of the length two. Therefore we will call this type of change notification trace-based change notification of the length two. We will compare different lengths later in the evaluation in chapter 5.

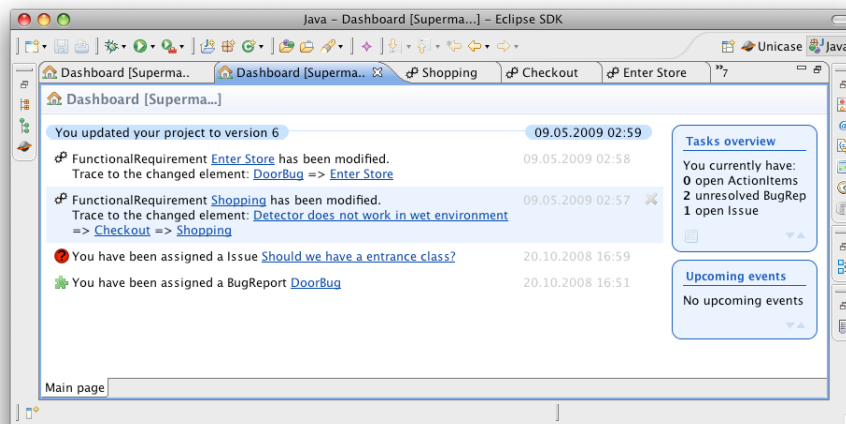


**Figure 6: TraceNotificationProvider (UML class diagram)**

Figure 6 shows a class diagram of our notification provider. The *TraceNotificationProvider* generates notifications only for changes on model elements a user is working on. As mentioned earlier, the work items are associated with model elements of the system model that are the target for the work to be done. For one specific work item assigned to a user, we can trace to all model elements that are reachable via opening links starting from these associated model elements. By following all opening links, we obtain all model elements that are (directly or indirectly) involved in the task. In other words we obtain all model elements a user is

working in on directly and indirectly. For any of these candidate model elements we have to determine whether they have been changed or not. Therefore the provider will process all operations that occurred to determine whether one of the model elements in question has been changed. The provider will generate a notification for all operations on these model elements. For any of these *TraceNotifications*, at least one trace from the changed element to a work item of a user exists. This *Trace* is an ordered aggregation of ModelElements. The user therefore can be provided not only with an exact description of the operation and where it occurred but also with a trace why he or she was notified about this change.

In an application context the notifications are generated whenever the user receives operations from a central repository to synchronize the local model instance. From the list of given operations a set of notifications is derived and presented to the user in a list oriented view. Instead of reviewing the incoming operations to understand the changes that occurred to the model, the user can rely on the notifications to be aware of relevant changes. Figure 7 shows a screenshot of the UNICASE Dashboard implementation. It displays four notifications; the two topmost notifications are *TraceNotifications*.



**Figure 7: Screenshot of the UNICASE Dashboard**

## 5 Evaluation

To evaluate our approach we conducted a case study where users manually rated generated change notification according to their personal relevance. To allow for comparison with other approaches, we implemented the most common ones to provide change notifications in a case study. Based on related work, we selected the following approaches for comparison:

- Modifier-based notifications [e.g. 11]: 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 2) at a major European airport. The objective of the DOLLI2 project was integrating facility management and telemetry data into the tracking and locating infrastructure developed in the previous project, together with expanding the 3D visualization on desktop computers as well as porting it to mobile devices. More than 20 developers worked on the project for about five months. All modeling was performed in the UNICASE tool [3]. This resulted in a comprehensive project model consisting of about 1000 model elements and a history of over 600 versions.

For the traceability-based notification provider as well as for the three other notification providers, we generated notifications for the DOLLI2 project data post mortem. To evaluate the quality of the notifications generated with the different providers we interviewed five participants of the project about their opinion on 12 of their notifications randomly selected for each provider.

Figure 8 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

**Figure 8: 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 on the second place as the creation of an artifact usually implies some kind of ownership. As the modifier-based provider also includes creator-based notification, but shows a significantly higher number of notifications it was expected to be rated on the third place.

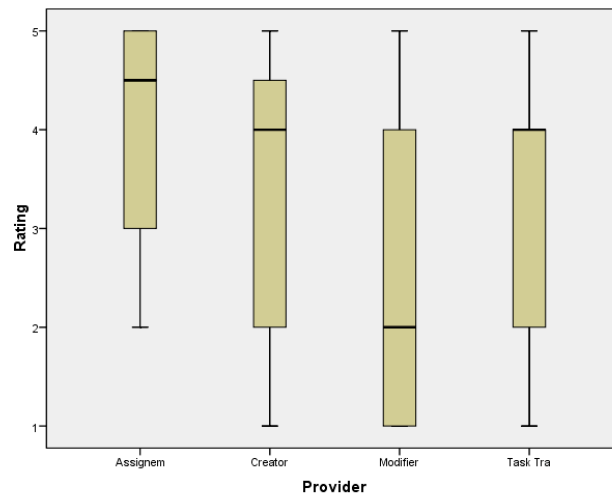
The table shows that all other providers generate up to 1000 times more notifications for the same data. In a practical appliance the number of change notification providers shall be as low as feasible in order not to overwhelm the notified users. As the

traceability-based notification provider has a low number of total notifications on one hand, but on the other hand creates notifications which are not covered by other providers (because of the traces between system and project model), we claim it to be useful if we can show that it is better rated than at least one of the existing notification providers.

Before presenting the results of our statistical analysis regarding the difference in the user ratings depending on the respective notification provider, we want to assure that the partially subjective ratings given by the interview based on five users show an acceptable variance. For this purpose, we consider the standard errors of the mean rating values for each user, more precisely, their empirical estimates. The mean rating value  $m=3.16$  of the whole sample is compared with the means of the ratings grouped by the respective user. The resulting empirical variance of

$$V = \frac{1}{k-1} \sum_{i=1}^k (m_i - m)^2 = 0.13, k = 1..5$$

as well as the standard errors  $\sigma_1$  to  $\sigma_5 = (0.199, 0.197, 0.223, 0.255, 0.203)$  of the user-grouped ratings are small enough to assume an unbiased interview result.



**Figure 9: Boxplot of the user rating for each provider**

Our hypothesis was that the TaskTrace provider results in a significantly higher rating than the Modifier provider, since we measured the mean ratings  $m_{\text{Modifier}} = 2.53$  and  $m_{\text{TaskTrace}} = 3.19$  based on  $n_1=60$  and  $n_2=47$  items in each category, respectively. Figure 9 shows a boxplot of the user rating for each provider.

The negative correlation  $\rho(L,R)=-0.127$  (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 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  $n_1 = n_2 = 47$  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. However, a significant difference in the rating between Modifier and TaskTrace provider cannot be assumed at the stronger 1% level, since  $d_{47} = 0.2341 < c(\alpha = 1\%; n) = 1.63 / n^{1/2} = 0.238$ .

To verify the positive result delivered by the test on the 5% level, we used the standard parametric test based on the Student-t-distribution. Assuming a normal distribution of both mean values, we applied the t-test for independent samples to check the hypothesis of equal mean values. The 95%-confidence interval for the difference of the mean values  $m_{\text{Modifier}} - m_{\text{TaskTrace}} = -0.66$  is  $[-1.18, -0.137]$ . The t-test returned a T-value of -2.50, which means that it is below the critical value  $-c(k=100, \alpha=5\%) = -1.66$  for  $n_1 + n_2 - 2 = 105 \approx k$  degrees of freedom (rounded off) and thus the null-hypothesis has to be rejected on the 5% level of significance.

Summing up these statistical results, we have showed that our traceability-based notification provider performed better than the modifier-based notification provider. Regarding the significant lower number of created notifications we claim our traceability-based notification provider to be useful in practical appliance. Only if the notifications are limited to a small enough set they can help the user to understand the changes relevant for his or her work. If the users are flooded with notifications it leads to them ignoring notifications entirely.

## 6 Conclusion and Future Work

We have shown that notifications based on the rich traceability of a unified model can help to increase change awareness in software development projects. The traceability-based notification provider generates notifications, which are not covered by existing approaches. We compared the approach and showed that the traceability-based change notification performs better than the modifier-based notification. As modifier-based change notification is currently in use in various software projects, we claim our traceability-based change notification provider to be useful in practice as well.

The contribution of our paper is a notification approach that relies on the traceability between project model and system model. Furthermore the combination with operation-based change tracking allows for effective generation of high quality notifications.

Our next step will be to enable the user to dynamically evaluate the notifications provided to him. The feedback of the user can directly lead to a new calibration of the notification provider. For example a user who often creates artifacts for other project participants would rate the creator-based notifications of these artifacts lower, and would not get creator-based notifications for artifacts he is not interested in. We believe that only by supplying a number of different user-configurable notification providers, users can be effectively notified about changes in a project without being flooded with irrelevant notifications. The provider presented in this paper is a first step towards this vision.

## References

1. Abrahamsson, P., Moser, R., Pedrycz, W., Sillitti, A., and Succi, G. Effort Prediction in Iterative Software Development Processes -- Incremental Versus Global Prediction Models. *Empirical Software Engineering and Measurement*, 2007. ESEM 2007. First International Symposium on, (2007), 344-353.
2. Biehl, J.T., Czerwinski, M., Smith, G., and Robertson, G.G. FASTDash: a visual dashboard for fostering awareness in software teams. *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM (2007), 1313-1322.
3. Bruegge, B., Creighton, O., Helming, J., and Kögel, M. Unibase - an Ecosystem for Unified Software Engineering Research Tools. *Third IEEE International Conference on Global Software Engineering, ICGSE 2007*, (2008).
4. Bruegge, B., Dutoit, A.H., and Wolf, S. Sysiphus: Enabling informal collaboration in global software development. *Global Software Engineering, 2006. ICGSE '06. International Conference on*, (2006), 139-148.
5. Carroll, J.M., Neale, D.C., Isenhour, P.L., Rosson, M.B., and McCrickard, D.S. Notification and awareness: synchronizing task-oriented collaborative activity. *International Journal of Human-Computer Studies* 58, 5 (2003), 605-632.
6. Cheng, L., Hupfer, S., Ross, S., and Patterson, J. Jazzing up Eclipse with collaborative tools. *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, ACM (2003), 45-49.
7. Cobéna, G., Abdesslem, T., and Hinnach, Y. A comparative study of XML diff tools. *Technical Report, INRIA*, 2004.
8. Cubranic, D. and Murphy, G. Hipikat: recommending pertinent software development artifacts. *Software Engineering, 2003. Proceedings. 25th International Conference on*, (2003), 408-418.
9. Curtis, B., Krasner, H., and Iscoe, N. A field study of the software design process for large systems. *Commun. ACM* 31, 11 (1988), 1268-1287.
10. Damian, D., Chisan, J., Allen, P., and Corrie, B. Awareness meets requirements management: awareness needs in global software development. *Proc. of the Int'l Workshop on Global Software Development, International Conference on Software Engineering (ICSE 2003)*, (2003).
11. De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G. Enhancing an artefact management system with traceability recovery features. *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*, (2004), 306-315.
12. Estublier, J. and Garcia, S. Process model and awareness in SCM. *Proceedings of the 12th international workshop on Software configuration management*, ACM (2005), 59-74.
13. Helming, J. Integrating Software Lifecycle Models into a uniform Software Engineering Model. *Software Engineering Conference - Workshop Proceedings, Gesellschaft für Informatik* (2008), 157-164.
14. Helming, J., Koegel, M., and Naughton, H. Towards Traceability from Project Management to System Models. *Proceedings of the 31st International Conference on Software Engineering*, (2009).
15. Holmes, R. and Walker, R.J. Promoting developer-specific awareness. *Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*, ACM (2008), 61-64.
16. Kögel, M. Towards software configuration management for unified models. *Proceedings of the 2008 international workshop on Comparison and versioning of software models*, ACM (2008), 19-24.

17. Kwan, I., Damian, D., and Storey, M. Visualizing a Requirements-centred Social Network to Maintain Awareness Within Development Teams. *Requirements Engineering Visualization, 2006. REV '06. First International Workshop on, (2006)*, 7.
18. LaToza, T.D., Venolia, G., and DeLine, R. Maintaining mental models: a study of developer work habits. *Proceedings of the 28th international conference on Software engineering, ACM (2006)*, 492-501.
19. Letkeman, K. Comparing and merging UML models in IBM Rational Software Architect: Part 3. IBM Rational, 2005.
20. Lindholm, T., Kangasharju, J., and Tarkoma, S. Fast and simple XML tree differencing by sequence alignment. *Proceedings of the 2006 ACM symposium on Document engineering, ACM (2006)*, 75-84.
21. Luqi. A graph model for software evolution. *Software Engineering, IEEE Transactions on* 16, 8 (1990), 917-927.
22. Papadopoulou, S., Ignat, C., Oster, G., and Norrie, M. Increasing Awareness in Collaborative Authoring through Edit Profiling. *Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on, (2006)*, 1-9.
23. Papadopoulou, S. and Norrie, M. How a structured document model can support awareness in collaborative authoring. *Collaborative Computing: Networking, Applications and Worksharing, 2007. CollaborateCom 2007. International Conference on, (2007)*, 117-126.
24. Sarma, A., Noroozi, Z., and van der Hoek, A. Palantir: raising awareness among configuration management workspaces. *Software Engineering, 2003. Proceedings. 25th International Conference on, (2003)*, 444-454.
25. Sinha, V., Sengupta, B., and Chandra, S. Enabling Collaboration in Distributed Requirements Management. *IEEE Softw.* 23, 5 (2006), 52-61.
26. Storey, M.D., Čubranić, D., and German, D.M. On the use of visualization to support awareness of human activities in software development: a survey and a framework. *Proceedings of the 2005 ACM symposium on Software visualization, ACM (2005)*, 193-202.
27. Tam, J., McCaffrey, L., Maurer, F., and Greenberg, S. Change awareness in software engineering using two dimensional graphical design and development tools. Alberta, Canada, Department of Computer Science, University of Calgary, (2000).
28. Tam, J. and Greenberg, S. A framework for asynchronous change awareness in collaborative documents and workspaces. *Int. J. Hum.-Comput. Stud.* 64, 7 (2006), 583-598.
29. Toulmé, A. and Inc, I. Presentation of EMF compare utility. *Eclipse Modeling Symposium, (2006)*.