On the Similarity of Task Contexts

Walid Maalej and Mathias Ellmann
Department of Informatics, University of Hamburg
Hamburg, Germany
{maalej, ellmann}@informatik.uni-hamburg.de

***Abstract*—Developers work on parallel tasks and switch between them due to interruptions and dependencies. For each task, developers interact with artifacts that constitute the task context. The more dissimilar tasks are, the more time is needed for switches to restore the contexts and adjust the mindset. Organizing tasks by their similarity can increase the efficiency of task switches. Moreover, knowing similar tasks of other developers might also save coordination effort. This paper studies the concept of tasks similarity based on the similarity of task contexts. We define task context as set or vector of the artifacts used in the task and apply popular similarity measures to compare the sets or vectors corresponding to the tasks. Instead of only counting the artifacts in the intersection and union of two contexts, we scale the artifacts with their relevance to the task using the degree-of-interest model. In a field study, a simulation, and an experiment, we investigated the accuracy of the different context similarity models and compared them to the accuracy of text-based similarity using the task descriptions.**

I. Introduction

A task is an atomic and well-defined work assignment for an individual or a team [5]. In software projects, tasks typically describe what developers should do and might include additional information such as the work priority, the resources needed, or the component affected. Tasks can be defined, e.g., in issue trackers, in project backlogs, or in Todo lists.

Both individual developers and development teams use tasks for organizing their work [2]. Developers use task lists to manage their personal productivity. They keep track of what is done, how much is still to do, and what should be done next. Development teams break down the work into manageable tasks, distribute the tasks among the members, and use them to synchronize and coordinate the work [2].

Identifying similar tasks in the task lists of developers or in the task repository of a project would be beneficial to the individual developers and for to the teams. Previous studies have shown that task switches are frequent [6], [8] and expensive [9]. With every task switch, developers have to decide what to do next, adjust their mindset to the new task [8], and locate relevant artifacts to work on [5]. It seems reasonable that the more similar two tasks are, the more efficient and less time consuming a task switch will be. This assumption corresponds to the first principle of Allen's Getting Things Done [1]: Select what to do next based on similar tasks that require the same context. In addition, automatically identifying similar tasks in a large distributed project can help teams to identify (a) which changes they should be aware of and (b) with whom they should synchronize to find hidden dependencies, avoid conflicts, and share knowledge [2].

In this paper, we study the concept of *task similarity* based on the similarity of the task contexts. The main idea is simple: two tasks are similar if "their most relevant artifacts" are similar. To identify the most relevant artifacts (e.g., a class, a method, or a tool) for a task, we observe developer's interaction with the tasks and use a popular artifact relevance model called degree of interest (DOI). To calculate the context similarity, we extend well-known set- and vector- similarity models.

The remainder of the paper is structured as follows. Section II introduces context relevance along with DOI. Section III describes how set-based similarity (Jaccard) and the vector-based similarly (Cosine) and text-based (Latent Semantic Indexing) can be used to calculate context and task similarity. Section IV and Section V report respectively on the methodology and the results of our empirical investigation to compare the different similarity models. Finally, Section VI discusses the findings, the limitations, and conclude the paper.

II. Context Relevance

When working on their tasks developers interact with tools. They might edit source code, debug it, run tests, or send a clarification email. Developers "consume and produce" artifacts of different types, including requirements, models, source code, test cases, or emails. Each artifact contains information that is useful for a particular task. Developers might modify the artifacts. They might also read documents, reference APIs, use tools, or search information. All interactions and the concerned artifacts constitute the context of the tasks.

Definition 1. A task *context* consists of all *interactions* and the concerned *artifacts* a developer performs in order to work on a task or achieve a particular goal [7].

Definition 2. Context relevance is a model which quantifies the importance of each artifact to the context compared to other artifacts. The more relevant an artifact is, the better it describes the context and the more likely it contributes to accomplish the task (inspired by [4]).

Kersten and Murphy [5] introduced a model called **degree-of-interest** (DOI) for calculating the relevance of each artifact used in a task. The DOI value is based on the frequency and decay of interactions with an artifact. The frequency denotes the number of interaction events referring to an artifact. The decay is proportional to the position in the event stream of the first interaction with the artifact [5]. Additionally, each interaction has a scaling factor assigned depending on the interaction type. Over time, if the artifact is not selected or edited, its DOI value decays. Each selection also leads to decaying the interest values of other artifacts. At any point in time, the interest values of the artifacts reflect their relevance ranking to a particular task [5]. Mylyn implements the DOI and is today one of the most recognized task management tools for the Eclipse IDE.

III. Context Similarity

Definition 3. Context similarity is a model which quantifies how similar two contexts and the corresponding tasks are. Two tasks are similar, if a developer would work on them in parallel or after each other, because the effort needed to switch focus and restore new information and tools is small.

Comparing the similarity of two task contexts is equivalent to comparing the similarity of two artifact collections. There are established measures for comparing the similarity of sets and vectors, which we discuss in the following.

A. Set-Based Similarity: Jaccard

Perhaps the most well-known measures for calculating set similarity is the **Jaccard** Index, which is defined as the size of the intersection of two sets divided by the size of the sets' union. When applied to task contexts, Jaccard is the number of artifacts in the intersection of two tasks divided by the number of artifacts in the union of these tasks.

Jaccard only considers whether an element is present in the set or not (i.e., an artifact is used in the task). It treats all artifacts equally, ignoring their relevance to the task. Unlike in information retrieval, where e.g., all characters in a string have the same relevance for a similarity calculation, in the task similarity comparison, it is crucial to use the relevance of the single artifacts. A task context might include a lot of "noise artifacts", which might be identified in the intersection of various tasks but does not increase their similarity. We extend Jaccard with the DOI relevance model discussed above. We define the *Relevance-aware Jaccard* of two tasks A and B:

$$\mathrm{DOIJac(A,B)}= \frac{\displaystyle\sum_{\mathrm{x}\in|\mathrm{A}\cap\mathrm{B}|} \mathrm{DOI(x,A)} + \mathrm{DOI(x,B)}}{\sum_{\mathrm{x}\in|\mathrm{A}|} \mathrm{DOI(x,A)} + \sum_{\mathrm{x}\in|\mathrm{B}|} \mathrm{DOI(x,B)}}$$

B. Vector-Based Similarity: Cosine

The vector-based **Cosine Similarity** is a popular measure to compare documents in text mining. It calculates the similarities of two vectors by measuring the cosine of the angle between them. This corresponds to the Euclidean product of the vectors divided by the product of the vector's magnitude.

In text mining, the vectors A and B usually correspond to the term frequency vectors of the documents TF-IDF, which reflect how important a term is to a document in a corpus. Transferred to task context comparison, this would mean that the vectors show the artifact frequencies, e.g., how often an artifact is used in a task divided by the total number of the artifact usages in all observed tasks. When applied to task comparison the relevance calculation with such a model is static. It does not consider the aspect of time and the meaning of the interaction. We also extend the Cosine Similarity using the relevance of each artifact as a single value in the vector. The vector positions correspond to all artifacts in both tasks. If an artifact is not used in a task, its relevance is 0. The other relevance values can be calculated using DOI.

$$\mathrm{DOICos(A,B)}= \frac{\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{DOI(x_i,A)} * \mathrm{DOI(x_i,B)}}{\sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}} (\mathrm{DOI(x_i,A)})^2} * \sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}} (\mathrm{DOI(x_i,B)})^2}}$$

C. LSI: Text-Based Similarity

We can also compare the similarity of tasks by comparing their textual descriptions. To do this, several text-based similarity algorithms can be used. One popular approach is the Latent Semantic Indexing (LSI) [3], a statistical model that analyzes the relationships between a set of documents and the terms they contain by producing a set of concepts (also called topics) related to the documents and terms. LSI assumes that terms that are close in meaning will occur in similar pieces of text. To remove all English stop words from the task description, we used the Natural Language Toolkit NLTK. To created the corpus and calculate the task similarity we used the python library Gensim. LSI requires defining the number of topics that should be extracted. For this we choose 300 which is recommended for mid-sized documents [3].

IV. Empirical Investigation

The main goal of our study was to gain insights into which context similarity model works in which situations. We focused on the following specific questions:

1) How do the different models, context a text based model, perform compared to each others?
2) Which additional factors are important for calculating the similarity of tasks and their contexts?

To answer these questions, we calculated three studies: a field study with professional developers, a simulation with data from the Bugzilla repository of the Eclipse community, and an experiment with students including an online survey.

A. Field Study with Professionals

In the field study, we collected context data of development tasks from real projects. We also collected the subjective assessments of the developers for the similarity of these tasks and compared the subjective assessments with the values predicted by the models described above.

We asked subjects to define at least 20 tasks that they plan to process in the next one or two weeks. The tasks should be defined either in Mylyn or in an issue tracker and imported via a Mylyn connector to Eclipse. Each time subjects start working on a task, they should activate it in the "Task List" view in Eclipse. This enables the tracking the context data. Subjects should work on their tasks as usual, including interruptions, definition of new tasks, issues, etc. Subjects should maintain a task similarity sheet, in which they specify for each task the first, second, and third most similar tasks. After each work day subjects should maintain this sheet and compare the task similarity. We collected the interaction histories from all subjects together with the similarity assessments. Overall we recruited 13 subjects. Nine of them submitted useful data and worked between September 2011 and August 2012. All subjects had at least two years of work experience and were familiar with Mylyn. For the data analysis, we created a tool that simulates the task activations in Mylyn and calculates for every active task a list of the similar tasks.

B. Simulation with Bugzilla Data

The second study consisted of a simulation with task data collected from the Bugzilla repository of the Eclipse community.It was hard to identify an open repository with the needed data, including the tasks, their interaction data (context), and information about the tasks similarity. The Eclipse Bugzilla repository partly includes this data. This repository is used to track bugs and other development tasks for the Eclipse products. Some product teams consistently use Mylyn and attach their interaction data to the task descriptions to allow others to reproduce the task context. Moreover, Bugzilla provides a field to link two kinds of related tasks: *"depends on"* and *"blocks"*. We assume that these related tasks include similar tasks.

We downloaded and crawled the task data from Bugzilla using the Python library BeautifulSoup. We first downloaded all tasks that have Mylyn Context data attached. This resulted in 6650 unique tasks. In this list, there were only 2605 tasks with related tasks. We then filtered only tasks and the related tasks which have mylyn-context data attached. This step led to 679 eligible tasks which have 928 related tasks. The related tasks included 430 unique "depends on" tasks and 292 unique "blocks" tasks. The sample included the task ids, the task descriptions, the task metadata (status, comments, creation time, etc.), the ids of the related tasks, and the context data describing the interaction of the developers.

C. Experiment with Students

To answer the second research question, we conducted an experiment with 152 software engineering students at the Technische Universität München in end of 2011. The study included two phases. First, we defined real development tasks and let the students work on them. Second, we asked the students to assess the similarity of these tasks and explain their decisions. We then compared the similarity assessments of the students with the similarity which we foresee for the tasks based on our models for relevance and similarity.

The students were randomly grouped into 37 teams of 4-5 members and randomly assigned to one of two projects. The first project, IM, was an instant messenger. The second, POLL, was a poll tool to vote for course-related questions and evaluate the course sessionsStudents had to implement the systems as client-server applications. For IM we defined four mandatory tasks and for POLL three. We did not give any constraints on how and when to implement the tasks, but we had assumption on their similarity. After finishing the tasks, we asked the students to rank the similarity of the predefined tasks and explain their rationales using an online questionnaire. We then compared the questionnaire results with our assumptions.

V. Empirical Results

A. Field Study with Professionals

The 9 subjects of the field study worked on a total of 55 tasks and identified 111 tasks as similar. From the 111 tasks, 64 tasks were unique. We first applied the similarity models on the tasks of each subject separately (within-subjects analysis) and searched for the similar tasks over all collected data (between-subjects analysis).

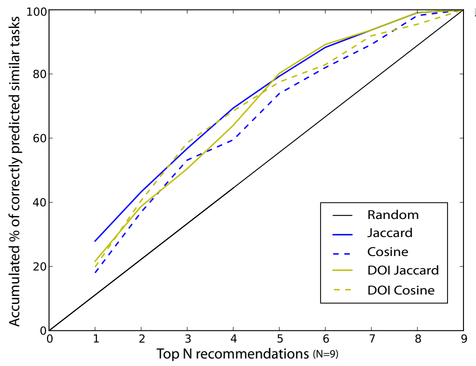
Figure 1a show the hit ratios of the different similarity models until the position N=9 by analyzing every subject individually. Figure 1b shows the prediction ability of the models if we search for similar tasks over the data of all subjects. The Normal Jaccard predicted all tasks until the 22nd position. Interestingly the cosine algorithms are worse then before. The results show that with a high number of artifacts (or vectors) its very hard for the cosine vector algorithm to distinguish between similar vectors (or task).

Next, we calculate for each task in the sample similar tasks based on the similarity of the descriptions. Overall, developers worked on 64 unique tasks that have a summary, for example "Find and eliminate Bugs". 61 tasks have a unique text description. Figure 1b shows that the text-based similarity is not as accurate as the context based similarity calculation. One possible reason is that most task descriptions in the sample were very short. Overall the shorted task description included 12 and the longest 89 characters. This finding is consistent with previous studies which showed that developers often encounter difficulties to write a meaningful description for their tasks [6].

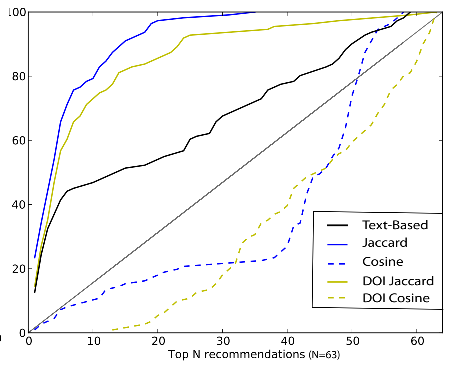
B. Simulation with Bugzilla Data

1) Context Similarity Models: The results of the **hit ratios** are summarized in Figure 1c. Overall, this time there is a clear difference between the performances of the similarity models. While the Jaccard and relevance-aware Jaccard had a good performance in predicting similar tasks, the results from Cosine and relevance-aware Cosine were worse than for a random recommendation. The Jaccard model performed best and was able to predict about 60% of the similar tasks up to the 20th. position. We think that this is a positive result since the search space is much bigger in this study. Recommending back 20 tasks to the developers, which others are working on and which they should be aware of for coordination reasons is acceptable. This prediction accuracy can be reached if developers are using context data or if the summary and product component name are available in the task description. We think that the accuracy of the two cosine models (pure cosine and DOI-Cosine) was very low since cosine is meant to compare the similarity of vectors with ordered elements. In the Bugzilla study related tasks are not ordered.

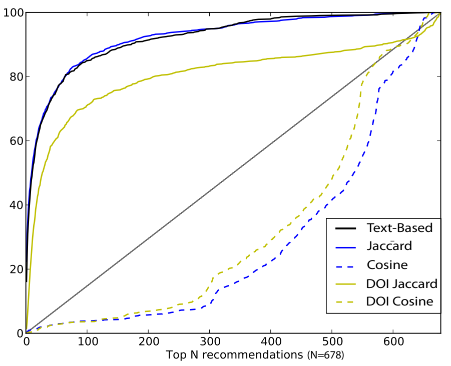
The pure Jaccard performed better than the context aware Jaccard. In the Bugzilla dataset, the relevance of the artifacts seems to rather have a secondary importance for the similarity. This sound reasonable, in particular for "blocks" tasks. Two tasks might block each other even if they share artifacts which are not relevant for both tasks (e.g., one API call in common).



(a) Field study (between subjects analysis)



(b) Field study (within subjects analysis)



(c) Bugzilla study

Fig. 1: Hit ratios of the different similarity models in the field study and the bugzilla study

We also ran a differentiated analysis for the "depends on" vs. "blocks" tasks. We found that the hit ratios for the "blocks" are better than the "depends on" relationships. The Jaccard curves have a sharp increase for the top N recommendations between N=5 and N=80. The last part of the curses has the same shape. The Jaccard algorithms are able to predict until the 20 position 74% of the "blocks" and 64% of the "depends on" dependencies. The text-based similarity has a slightly worse accuracy: 68% "blocks" task and 63 % "depends on" tasks.

2) Text Similarity vs. Context Similarity: We found that the text-based similarity model performed as good as the pure Jaccard model. This shows that it is possible to identify similar (or related) tasks by comparing the similarity of their textual descriptions. As a task description we used the summary, product, and component names from the bug report. We found that adding the metadata about the product and component names (affected by the task) significantly increases the prediction accuracy of the text-based similarity model. By using only the summary as an input, the text-based similarity model discovered 65% of the tasks until the first 100th position. By adding the metadata (product and component names) the model discovered 85% of the tasks.

We also created an evaluation dataset with the text-similarity as a golden standard. We calculated the text-based similarity between the 679 unique tasks in the original dataset. We then selected the task pairs with a similarity score of 0.8 or higher, which means that the tasks have at least 80% of text similarity. This resulted in 257 unique tasks with 390 similar tasks. Finally, we examined whether the models can predict similar tasks in this dataset. The overall results were consistent with those based on the "depends on" and "blocks" relationships. Also in this case the vector-based approaches had a very low accuracy.

C. Experiment with Students

In the experiment we checked: Hypothesis H_0: Our task similarity definition based on context similarity is intuitive for developers. Hypothesis H_1: Developers understand task similarity in a different way. Overall, 127 students (84 %) answered the questionnaire. A full matching of similarity means that a student ranks all tasks exactly in the expected order. More than half of the ranks exactly match with the expected positions. A Chi Squared test rejected H_1 with 95% confidence.

Students ranked the tasks similarity and were asked to explain the reason for their decision (open question). We manually analyzed the justifications, in an iterative, independent, peer-conducted process, focusing on additional similarity drivers. 92 of the 127 submitted questionnaires included justifications.

Functionality & architecture indicate task similarity. 55 participants considered the similarity of the functionality as indicator for tasks similarity. Even though we defined two tasks as similar, if they use the same tools and artifacts, the functionality to be implemented played the most important role in the justifications. One student wrote: "task 3 is most similar because most code has to run on the client side...". Students often compared tasks on a conceptual level, relating the design, architecture, or workflow of the desired solution.

Different technologies let tasks appear dissimilar. 32 participants claimed that if the technologies (e.g., framework or programming language) involved in the tasks are different, both tasks were considered as completely dissimilar. In this case tools and artifacts involved in the tasks seem to recede into the background. One student stated: "working on [internal behavior] is totally different than working on gui stuff". One other wrote: "tasks 1 and 2 comprise editing java code [...] using a database backend on the server. Task 3, by contrast, mainly involves modifying cascading style sheets which are exclusively relevant for the client."

Participants think about tools instead of artifacts. We counted 21 participants who compared tasks based on the tools they used, but very rarely referring to concrete artifacts used in the tools. Instead of explaining differences between tasks based on different artifacts (e.g., index.html or Message.java), they explained the differences by the tools used (HTML editor vs. Java editor). One wrote: "my most important tool [for this task] was Adobe Fireworks. It is dissimilar by definition".

Participants think about artifact types instead of artifact instances. Instead of explaining differences between tasks based on concrete artifacts (e.g., index.html, main.css, or Message.java), 18 participants rather see the differences based on the artifact types (HTML, CSS, or Java documents). Concrete file names seem to play a subordinate role. One student wrote: “for implementing functionalities we mostly used source code editor and java files, for beautifying an application we need to change the css file”.

VI. Discussion and Conclusion

Comparing the results of the Bugzilla study with the field study, we observed that the Jaccard models work very well for both situations: similar tasks of a single developer and similar tasks of various team members. It also seems that, the larger the search space is, the more accurate the Jaccard-based models are. The cosine models seem to work better for predicting similar tasks for a single developer, which were ordered correctly as most similar, second most similar, etc. We also found that the relevance-aware models (both Jaccard and Cosine) are slightly better for comparing the tasks of an individual developers. On the other hand, the pure Jaccard performed better for comparing tasks of different developers. To better understand the reasons, a factor analysis based on a larger and a more diverse sample will be needed. Does the size of context data play any role in comparing the context? Can we develop more accurate context relevance models? Are there other approaches for measuring context similarity?

Using the text descriptions of the tasks as indicator for the task similarity seems to work as well as using the task contexts. One major advantage of the context-based approach is that it does not need the task descriptions. Previous studies have confirmed that a large number of developers’ tasks are not described and are rather implicit in the head developers [6], [8] On the other hand, one disadvantage of the context-based approach is that the developers need to start working on a task and to make enough "progress" in order to collect the context. We think that a combination of the context-based (interaction-based) and text-based approach is needed for a similarity model to be applicable in practice. The correct management of the task context seems to be non-trivial for developers. Our results show that only very few developers (about 5% in the experiment) correctly activate and deactivate the tasks in Mylyn. We expected far more since students were introduced to the tools and their participation willingness was high. This shows overall the difficulty of task activation

We think that task context includes additional dimensions which we did not consider in this paper. Our evaluation results show that developers (at least beginners) think, that the similarity of task contexts depends on other factors that the interactions, artifacts, and tools. For instance, the similarity of functionality to be implemented, of architecture and technology used might be strong indicators for context similarity.

Both relevance and similarity are two well-studied concepts in statistics, machine learning, and information retrieval. Some of the assumptions of this work are based on these results such as Jaccard or Cosine. In software engineering authors have studied the relevance and similarity. For instance, Parnin and Goerg [9] looked at various ways to compute file relevance for a task and used (including recency and frequency). There are quite a few works that compute artifact similarity using text similarity approaches or topology [10]. In this work we focused on a specific view of task similarity based on comparing the developers’ interaction histories when working on the task.

As for every study there are several limitations to the validity of our results. In the field study, we used a relatively small sample (55 tasks) to evaluate of the context models. The external validity of the Bugzilla study is rather high, since the data represented different task types of more than 30 developers from 10 different products (i.e., teams). The relationship between the tasks collected were specified as “depends on” and “blocks”, while the relationship which we aimed at evaluating was “similar”. To mitigate this threat we compared the context similarity models against the similarity predicted by text similarity algorithm and found supportive results. As for the experiment, the number of tasks assessed by students was rather small. A larger number of tasks would have complicated the study leading to difficult and probably imprecise assessments by the students. For the field study, the assessments concerned a larger number of tasks up to 10 tasks.

Overall the results are appealing as we were able to correctly predict similar task contexts in over 80% of the studied tasks. We think that our work gives insights on how to design a recommender of similar tasks for both individuals and teams.

References

[1] David Allen. *Getting Things Done. The Art of Stress-Free Productivity*. Penguin, New York, New York, USA, 2001.

[2] Kelly Blincoe, Giuseppe Valetto, and Daniela Damian. Do all task dependencies require coordination? the role of task properties in identifying critical coordination needs in software projects. In *Proceedings of 9th ESEC/FSE*, pages 213–223. ACM, 2013.

[3] Scott C. Deerwester, Susan T Dumais, and Thomas K. et al. Landauer. Indexing by latent semantic analysis. *Journal of the American Society for Information Science and Technology*, 41(6):391–407, 1990.

[4] B. Hjorland and F. Sejer Christensen. Work tasks and socio-cognitive relevance: a specific example. *Journal of the American Society for Information Science and Technology*, 53(11):960–965, 2002.

[5] Mik Kersten and Gail C. Murphy. Using task context to improve programmer productivity. In *Proceedings of SIGSOFT ’06/FSE-14*, pages 1–11. ACM Press, 2006.

[6] Walid Maalej. Task-First or Context-First? Tool Integration Revisited. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering - ASE ’09*, pages 344–355, Washington, DC, USA, November 2009. IEEE Computer Society.

[7] Walid Maalej, Thomas Fritz, and Romain Robbes. *Recommendation Systems in Software Engineering*, chapter Collecting and Processing Interaction Data for Recommendation Systems, pages 173–197. Springer, 2014.

[8] Gloria Mark, Victor M. González, and Justin Harris. No Task Left Behind? Examining the Nature of Fragmented Work. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI ’05*, pages 321–330, New York, NY, USA, 2005. ACM Press.

[9] Chris Parnin and Carsten Gorg. Building usage contexts during program comprehension. In *Proceedings of the 14th IEEE International Conference on Program Comprehension*, ICPC ’06, pages 13–22, Washington, DC, USA, 2006. IEEE Computer Society.

[10] Giriprasad Sridhara, Emily Hill, Lori Pollock, and K. Vijay-Shanker. Identifying word relations in software: A comparative study of semantic similarity tools. In *Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension*, ICPC ’08, pages 123–132, Washington, DC, USA, 2008. IEEE Computer Society.