Effects of Explicit Feature Traceability on Program Comprehension

Jacob Krüger[1], Gül Çalıklı[2], Thorsten Berger[2], Thomas Leich[3], Gunter Saake[1]

Abstract: This abstract is based on our paper with the homonymous title published at the Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) 2019 [Kr19].

Keywords: program comprehension; feature traceability; software maintenance; separation of concerns

Software developers spend a substantial amount of their effort in software development and maintenance on program comprehension. While numerous artifacts can support program comprehension, for instance, documentations or models, developers mainly focus on understanding the actual source code. In this regard, a particular challenge for developers is to locate the features of their system in the source code, which is needed to perform any update, bug fix or extension on any feature. So, it is crucial to improve the comprehensibility of the source code itself, with various studies showing that even small improvements can have strong effects. One particular technique for improving program comprehension are explicit feature traces in the source code to facilitate feature location. Mainly, feature traces are instantiated using one or both of the following concrete techniques:

- *Annotations* represent a virtual separation of features, marking the begin and end of code that belongs to a feature within a single code base.
- *Decomposition* refers to physically separating features, for example, by implementing features in separate classes or feature modules that are separated from the base code.

While both techniques have been investigated in several studies, which highlight different pros and cons of either technique, the actual effect of feature traces on program comprehension remains unclear and requires more detailed evidence. In particular, most studies solely compare between both techniques, but do not compare them to source code without traces—often assuming that either technique is beneficial to use anyway.

In our paper, we tackled this problem with an empirical study, comprising an experimental and a survey part, in which we compared annotations, decomposition, and pure object-oriented code of the same system. To improve the motivation and gain more reliable insights,

[1] Otto-von-Guericke-University, Magdeburg
[2] Chalmers | University of Gothenburg
[3] Harz University of Applied Sciences & METOP GmbH, Wernigerode & Magdeburg

we personally invited 144 professional developers from various organizations and countries to participate in our online study. We obtained results for 49 of these software developers, analyzing their (1) effectiveness in correctly solving three program-comprehension and three bug-localization tasks, (2) efficiency in terms of time needed to perform these tasks, and (3) subjective perception of explicit feature traces.

Overall, we observed the following effects and perceptions of explicit feature traces:

- Annotations did significantly improve the comprehension of feature interactions compared to object-oriented source code for two tasks. Our participants' responses suggest that annotations allowed them to focus on the correct code parts and identify interactions more easily, conflicting some common believes that annotations hamper program comprehension.
- Decomposition did significantly hamper bug localization compared to object-oriented code for one task. While researchers often argue that physically separated features are easier to comprehend, our participants could not identify the correct interaction between a feature and the base code that caused the bug.
- Concerning the survey responses of our participants, we found that explicit feature traces (i) extend program-comprehension strategies, (ii) do not impair program comprehension, and (iii) are perceived as positive. So, we argue that explicit feature traces, and especially annotations, are a simple, yet effective, technique to improve program comprehension. However, there are open challenges, concerning the granularity of feature traces and their maintenance.

None of the three source-code versions resulted in significant changes on task completion time. Our results confirm most works that compare between annotations and decomposition, yielding no significant differences. However, we also showed that it is problematic to not compare against the baseline of source code without any traces, for which we observed conflicting results compared to common believes in software-engineering research. In summary, our results indicate that lightweight feature traces, such as annotations, provide immediate benefits to developers while developing and maintaining software. Moreover, lightweight annotations do not require extensive training or tooling, making it simpler to introduce them compared to heavyweight traceability tools (e.g., DOORS). Our future work will focus on further analyzing especially the effect of feature annotations, and on developing tooling to support their introduction, maintenance, and analysis.

Bibliography

[Kr19] Krüger, Jacob; Çalıklı, Gül; Berger, Thorsten; Leich, Thomas; Saake, Gunter: Effects of Explicit Feature Traceability on Program Comprehension. In: Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE. ACM, pp. 338–349, 2019.