

Refactoring and Evaluating Existing Code

Position paper for the workshop on Software Archeology, OOPLSA 2001

Klaus Marquardt
Käthe-Kollwitz-Weg 14, D-23558 Lübeck, Germany
Email: marquardt@acm.org

Copyright © by Klaus Marquardt

Experience: Refactoring to multiple parallel users

The system was written by two subsequent developers during 4 years in a change-on-demand development style, and consisted of about 50000 NLOC written in Clipper 5. My task was to change the system design from a single-user system to allow parallel usage by about 6 users, and to add several features. Besides the source code, the user and some future users were available for questions.

At first I did not even try to understand the system, instead I used `grep` to find all calls that accessed the database. Some were obviously used together, so I refactored the code to group them together. The names were chosen quite well so I could relate most call sequences to a known use case. All not-so-obvious calls I resolved by a few debugging sessions after I had learned more about the supported use cases. After that I began to factor the non-DB related code out of these call sequences, for example I removed all print commands between DB read statements and filled memory structures instead. After that I introduced a transaction concept that allowed me to hold read and write locks on the database for a minimal time, and I had a defined work sequence for each use case. This effort took about three weeks (spread over 2 months), but the system wasn't that large after all and I felt pretty comfortable with it in the end.

Experience: Evaluating for reuse

The system was written by a team of about 15 developers during 4 years for a proprietary embedded system, and consisted of about 200000 NLOC written in C and C++. My task was to evaluate that code for a possible reuse of design ideas and source code. Besides the code, three of the initial developers were available for questions.

The first thing for me was to contact the initial developers and ask them for an overview of the system structure. After a quick overview of the static structure, we identified three major workflow sequences that would crosscut the system and exhibit most of its properties. I sat together with a hands-on developer and stepped through the systems' dynamic call sequences of these workflows. For the most relevant design decisions we discovered during this session, I asked for design alternatives and reasons. Each of these walk-through sessions ended with a reflection on the static structure. Without understanding the system entirely, I managed to get a feeling for the appropriateness of the design, and the level of coupling between different system parts. After two days I was able to give management an informed proposal about reuse of the design ideas and potential reuse of specific system parts.

Position statement

I wish I had a tool that shows me the way through a system. OK, more specifically: the flow of control through a system when executing some major workflow sequence. This control path should be recorded while the system runs, and displayed graphically. I need to have two particular important features. Call sequences that are used often must be drawn with a thicker line or just more lines, just as a person equipped with a pencil would do. More important, selected call sequences must be omissible or expandable in the graphics in order to hide frequent but less interesting calls, e.g. to utility classes.