
Type of paper: Survey

Correlation to the real-world setting: Refactoring is a part of many commercial IDEs. Agile methods like XP are widely used and advocate refactoring. However, open issues remain:

1. Identifying parts of the software that will benefit from refactoring.
2. Ensuring that the refactoring doesn’t break the software.
4. Deciding on the optimal amount of automation in the refactoring process.
5. Avoiding conflicts when refactoring frameworks and product line software.

What is the hypothesis:

1. Duplicated code is one of several possible “bad smells” that should trigger refactoring.
2. Formal proofs of equivalence won’t scale to handle interesting refactorings, we need relaxed formalisms (same method calls in same order, etc) and testing.
3. Language independent refactorings are the solution to keeping multiple representations in sync.
4. The developer must be involved in refactoring, since he or she carries essential information not available in the software.
5. Evolution conflicts can be resolved with logic metaprogramming.

Validation: Persuasion through the citation list. It’s a survey after all.

Result: There is a lot of work left to do.

Do you believe the result, and why? I’m persuaded on points 1, 2, and 4. 2 is beyond my area of expertise, but I know enough about what happens to Java code to realize that tests could break even after a conservative refactoring.

I think that the agile methods have the right idea with respect to 3: don’t keep multiple representations of the program. I don’t know what logic metaprogramming is, so I’m not persuaded by 5. Wouldn’t a history of prior refactorings help?