6.883: Program Analysis

Assignment 1

Due: Tuesday, September 13 at 11am

In 1–2 pages, list several difficulties that you often encounter during software development — design, implementation, documentation, testing, maintenance, or otherwise. For each such difficulty, discuss the underlying cause. Is there some way that tools could help?

Below are some examples (you should come up with your own). Your list need not be nearly this long, but each item should be more substantial. Discuss the problem: motivate why people should care about it, explain why it is not easy to solve or work around (and speculate on why no one has solved it to date), and brainstorm some possible solutions.

Submit this assignment to the lecturer electronically (at `mernst@csail.mit.edu`), in a single file (text, HTML, PDF, or Ghostscript-viewable PostScript). This will facilitate sharing them with the class.

- In dynamically-typed languages, it is easy to apply illegal operations to data, and for such errors to be latent for long periods until some input exposes them. Even if easily reproducible, they might be exposed only after other long-running computations, making them difficult to detect. This sort of error is particularly frequent when accessing deeply-nested data structures: it is easy to forget one level of dereference.
- Interning values (replacing them by a canonical version) saves space, since only one version needs to be stored, and also saves time, since equality testing can be performed via pointer comparisons. Failure to properly intern can be extremely hard to track down, however.
- Reusing code requires an understanding of its behavior. Most code is not documented, however. This lack of specifications and documentation makes it difficult to use the code and difficult to know its assumptions or operational parameters (the values over which it has been tested and is known to work).
- Large components are often more worth reusing than small ones. However, they are also more likely to make assumptions (such as that they control execution of the program, that there is no need for thread safety, or that batch processing is acceptable) that may not be acceptable to an integrater who wishes to use these large components.
- Test suites are crucial to eliminating bugs and improving confidence in programs, but are difficult and tedious to create. For example, it may be difficult to create valid inputs to a program, because there are implicit constraints on the input. Given an arbitrary input to the program, it can be difficult to determine whether the program operated correctly.
- Large, comprehensive test suites tend to take a long time to execute. That discourages developers from using them as frequently as they ought to, or slows them down waiting for tests to complete when they could be moving on to other tasks.
- Some program analyses produce so much output that it is difficult or time-consuming to separate the useful information from spurious reports.
- "Maintenance" activities (modifying a program) tend to degrade its structure at both the macro (modules) and micro (specific functions) levels.
- It is hard to build new analyses for existing languages (such as Java); lots of grunge work is required, even though building a front end is no longer considered interesting research. A related problem is that real infrastructures are difficult to use, and it is difficult to integrate new tools with them.

Avoid discussing implementation annoyances unless you can identify an underlying principle. (Example: "Windows (or Unix) lacks this feature that Unix (or Windows) has," or "My favorite tool does not support x.") Avoid mentioning difficulties in performing tasks that don't matter. (Example: "I can't determine the cyclomatic complexity of my Tcl code." No one cares about cyclomatic complexity any more.) Avoid problems that are extremely minor or that can be solved easily. (Example: "I often fail to balance delimiters before attempting to compile a file." This costs only a few moments of compile time, and a trivial editor extensions can detect it even earlier.)

While some people might be able to knock off an answer to this assignment in a few minutes, it will reward careful thought about interesting problems and issues. Please introspect deeply and thoughtfully about program development and maintenance. Doing so will help you in this class, and beyond.