

Facts and Fallacies of Software Engineering

A summary of Robert Glass' 2003 seminal work.

Facts and Fallacies of Software Engineering



Robert L. Glass
Foreword by Alan M. Davis

People

- The most important factor in software work is the quality of the programmers.
- The best programmers are up to 28 times better than the worst programmers.
- Adding people to a late project makes it later.
- The working environment has a profound impact on productivity and quality.

Tools and Techniques

- Hype (about tools and technology) is a plague on the house of software.
- New tools and techniques cause an initial loss of productivity / quality.
- Software developers talk a lot about tools, but seldom use them.

Estimation

- One of the two most common causes of runaway projects is poor estimation.
- Software estimation usually occurs at the wrong time.
- Software estimation is usually done by the wrong people.
- Software estimates are rarely corrected as the project proceeds.
- It is not surprising that software estimates are bad. But we live and die by them anyway!
- There is a disconnect between software management and their programmers.
- The answer to a feasibility study is almost always "yes".

Reuse

- Reuse-in-the-small is a solved problem.
- Reuse-in-the-large remains a mostly unsolved problem.
- Reuse-in-the-large works best in families of related systems.
- Reusable components are three times as hard to build and should be tried out in three different settings.
- Modification of reused code is particularly error-prone.
- Design pattern reuse is one solution to the problems of code reuse.

Requirements

- One of the two most common causes of runaway projects is unstable requirements.
- Requirements errors are the most expensive to fix during production.
- Missing requirements are the hardest requirements errors to correct.

Design

- Explicit requirements 'explode' as implicit requirements for a solution evolve.
- There is seldom one best design solution to a software problem.
- Design is a complex, iterative process. Initial design solutions are usually wrong and certainly not optimal.

Coding

- Designer 'primitives' rarely match programmer 'primitives'.
- COBOL is a very bad language, but all the others are so much worse.

Error removal

- Error removal is the most time-consuming phase of the lifecycle.

Testing

- Software is usually tested at best to the 55 to 60 percent coverage level.
- 100 percent test coverage is still far from enough.
- Test tools are essential, but rarely used.
- Test automation rarely is. Most testing activities cannot be automated.
- Programmer-created, built-in debug code is an important supplement to testing tools.

Reviews and Inspections

- Rigorous inspections can remove up to 90 percent of errors before the first test case is run.
- Rigorous inspections should not replace testing.
- Post-delivery reviews, postmortems, and retrospectives are important and seldom performed.
- Reviews are both technical and sociological, and both factors must be accommodated.

Maintenance

- Maintenance typically consumes 40 to 80 percent of software costs. It is probably the most important software lifecycle phase.
- Enhancements represent roughly 60 percent of maintenance costs.
- Maintenance is a solution-- not a problem.
- Understanding the existing product is the most difficult maintenance task.
- Better methods lead to more maintenance, not less.

Quality

- Quality is a collection of attributes.
- Quality is not user satisfaction, meeting requirements, achieving cost and schedule, or reliability.

Reliability

- There are errors that most programmers tend to make.
- Errors tend to cluster.
- There is no single best approach to software error removal.
- Residual errors will always persist. The goal should be to minimize or eliminate severe errors.

Efficiency

- Efficiency stems more from good design than good coding.
- High-order language code can be about 90 percent as efficient as comparable assembler code.
- There are tradeoffs between optimizing for time and optimizing for space.

Research

- Many researchers advocate rather than investigate.

Ten fallacies

1. You can't manage what you can't measure.
2. You can manage quality into a software product.
3. Programming can and should be egoless.
4. Tools and techniques: one size fits all.
5. Software needs more methodologies.

Ten fallacies

6. To estimate cost and schedule, first estimate lines of code.
7. Random test input is a good way to optimize testing.
8. "Given enough eyeballs, all bugs are shallow".
9. The way to predict future maintenance costs and to make product replacement decisions is to look at past cost data.
10. You teach people how to program by showing them how to *write* programs.