5895 Software Design

Adam Burry, P.Eng.
aburry@ieee.org
2015-03-30

There is a charmingly adolescent quality to the agile literature: I am sooooo unique! Nobody before me understood what life is about! My folks are sooooo, like, 20-th century!

Bertrand Meyer, 2014

Hype is the plague on the house of software.

Robert Glass, 2003

Evidence-Based Practice

5 Steps

1. Ask answerable questions
2. Collect evidence
3. Critically appraise evidence
4. Integrate evidence into decision making
5. Reflect on results

The bad news is that, in our opinion, we will never find the philosopher’s stone. We will never find a process that allows us to design software in a perfectly rational way. The good news is that we can fake it.

David Parnas and Paul Clements, 1986

We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others.

David Parnas, 1972

Design Activities

Usual suspects

1. Architecture
2. High-level design
3. Low-level design
4. Detailed design
5. Prototyping
6. Development

Counter-intuitive

1. Construction
2. Coding
3. Maintenance

Malleability

Things that change

1. Business rules
2. Hardware dependencies
3. Input and output
4. Nonstandard language features
5. Complex algorithms
6. Status variables

You wanted a banana but what you got was a gorilla holding the banana and the entire jungle.

Joe Armstrong, 2009

Recommended Reading

Bertrand Meyer, “Agile! The Good ,the Hype and the Ugly”, Springer, 2014.

Pierre Bourque, “SWEBOK V3.0: Guide to the Software Engineering Body of Knowledge”, IEEE, 2014.

Andy Oram and Greg Wilson, “Making Software: What Really Works, and Why We Believe It”, O’Reilly Media, 2011.

Bertrand Meyer, “Touch of Class: Learning to Program Well with Objects and Contracts”, Springer, 2009.

Steve McConnell, “Software Estimation: Demystifying the Black Art”, Microsoft Press, 2006.

Scott Meyers, “Effective C++: 55 Specific Ways to Improve your Programs and Designs”, 3rd ed., Addison-Wesley, 2005.

Steve McConnell, “Code Complete: A Practical Handbook of Software Construction”, 2nd ed., Microsoft Press, 2004.

Tore Dyba, “Evidence-based Software Engineering for Practitioners”, IEEE Software, 2004.

Robert Glass, “Fact and Fallacies of Software Engineering”, Addison-Wesley, 2003.

Brian Kernighan, “The Practice of Programming”, Addison-Wesley, 1999.

Steve McConnell, “After the Gold Rush: Creating a True Profession of Software Engineering”, Microsoft Press, 1999.

John Lakos, “Large-Scale C++ Software Design”, Addison-Wesley, 1996.

David Parnas, “A Rational Design Process: How and Why to Fake It.”, IEEE Transactions on Software Engineering, Feb 1986.

David Parnas, “On the Criteria to Be Used in Decomposing Systems into Modules”, Communications of the ACM, Dec 1972.

Textbook: Agile

1. Continuous Integration
2. Test-Driven Development
3. Refactoring
4. Time Boxing
5. User Stories
6. ~~Pair Programming~~
7. ~~Embedded Customer~~
8. ~~Open Workspace~~

Textbook: UML

Core diagrams

1. Class
2. Sequence
3. Activity
4. State Machine

Other notations

1. Data Flow Diagram
2. Tables
3. SysML
4. Ad-hoc

Textbook: Patterns

The choice of programming language is important because it influences one’s point of view. Our patterns assume Smalltalk/C++ level language features, and that choice determines what can and cannot be implemented easily. If we assumed procedural languages, we might have included design patterns called “Inheritance,” “Encapsulation,” and “Polymorphism.” Similarly, some of our patterns are supported directly by the less common object-oriented languages. CLOS has multi-methods, for example, which lessen the need for a pattern such as Visitor.