
How to Develop Software

A process for developing the right software quickly and efficiently.

Business Process Reengineering

Make a business do the right thing efficiently by

- focussing on its main processes
- eliminating unnecessary work
- eliminating wasted time

“Reengineering the Corporation” by Michael Hammer and James Champy

Process of Software Development

Necessary

- get requirements
- write code
- test code

Unnecessary

- make and fix errors
- build wrong features
- communicate (meetings,
documentation)

What we know works

Incremental development

Testing

Reviewing

If these are good, let's do them
ALL THE TIME!

Extreme Programming

Goals

- Simplicity
- Communication
- Testing
- Aggressiveness

<http://c2.com/cgi/wiki?ExtremeProgramming>

<http://www.xProgramming.com>

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

692

Incremental development

System should always run.

Make changes as quickly as possible/as small as possible.

Maintainers

- add one feature at a time
- make sure they don't break anything

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

693

Tests

Make sure changes don't break anything.

Document features.

Measure of completeness.

Write test for new feature first, then
implement the feature. Stop when tests
work.

Reviews

Design is done with CRC cards.

Entire group reviews it as it is done.

All code is written in pairs.

All code is reviewed as it is entered.

Process

Plan

Test

Program

Refactor

Optimize

Planning

Specify system by a set of “stories”/use cases.

Developers estimate time to implement each story.

Customer ranks stories by importance.

Customer orders stories and puts them into iterations that should not take more than two or three weeks to implement.

Replan after every iteration.

User stories for WikiWorks

Have “save” button at top of screen so you don’t have to scroll when you edit

Have button that lets you put in your e-mail address so you are notified when page changes

[wiki-VisualWorks:Ralph Johnson] generates a link to page “Ralph Johnson” on the other wiki

[isbn:0-13-318387-4] will generate a link to

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

698

User stories

Let the business people make business decisions and technical people make technical decisions.

Ensure good communication between business people and developers.

Let business people know how things are going - provide measure of completeness.

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

699

User stories and developers

Make sure you can write test case for each user story.

Divide user story into “engineering tasks.”

Implement one task at a time.

- Write test case
- Get it to work
- Refactor

User stories and management

Estimate total time of project by adding up times for all stories.

Feel free to add stories, but that adds time.

Feel free to change priority of stories.

Have developers reestimate stories every few months.

Compare predicted time for story with actual time.

Testing

Unit tests vs. functional tests.

Unit tests should always work.

Write tests first.

Convert a user story into a set of test cases.

Change program until all tests work.

Tests should be small enough that you can do several in one day.

Programming

Do the simplest thing that could possibly work.

Follow standard coding conventions.

Program in pairs.

Do the Simplest Thing that Could Possibly Work

Reuse vs. building your own

Just implement the current test.

Think of at least two ways you could
implement this, and choose the simplest
one.

Follow Standard Coding Conventions

Follow a standard like “Smalltalk Best
Practice Patterns” by Kent Beck.

Use standard classes.

Use standard patterns.

Write code to be as clear as possible.

- Choose names carefully.
- Write comments when necessary.
- Rewrite code if it is not clear.

Program in Pairs

Increases communication, improves quality.

Pick pairs based on skill mix.

- Tests - customer and developer
- Adding a feature to complex subsystem
 - - feature expert, subsystem expert

Switch drivers.

Both people must understand and be

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

706

Refactoring

The system should be as simple as possible.

Eliminate all duplicate code.

Adding code can make it more complex than necessary.

Refactor after adding code.

If it is easier to refactor before you add code, refactor before you add code.

Object-oriented Programming and Design - Copyright 1998 by Ralph E. Johnson

707

Optimizing

Measure your program's performance and fix bottlenecks.

Usually it is easier to optimize a well-factored program.

It is always easier to optimize a program that is easy to understand.

Documentation

What is needed:

- User documentation?
- High-level design documentation?
- Detailed programmer documentation?

Write as little documentation as possible.

- Communicate orally instead
- Use program as design documentation

Documentation

~~Class comments for every class.~~

Method comments for a few methods.

- Abstract methods in abstract classes
- Methods with bad names
- Overly complex methods
- Algorithm references, citations, descriptions of facts that reader should know
- High-level design documentation

Design

You need an overall architecture.

Prototype your architecture before you start quickly adding features.

Select a “system metaphor”.

Use light-weight design process like CRC cards to do design each new piece.

Most design occurs during refactoring.

Patterns

Analysis patterns - patterns about the problem

Design patterns - occur during refactoring

Coding patterns

Conclusion

Design occurs all the time, not just up-front.

Rewrite your program as you learn.

Program reflects current knowledge.

Don't separate design and code.

This can be a very disciplined approach, it is just different from typical up-front design approach.