Refactoring Yourself As A Developer

Ryan Taylor

2009.04.28

Refactoring Yourself

What is refactoring?

To improve readability while preserving meaning.

Refactoring Yourself

Example:

Today in the afternoon I went to the park and I went to the grocery store.

Refactored:

I went to the park and grocery store this afternoon.

Refactoring Yourself

In the context of this presentation:

To improve yourself while preserving your role as a developer.

Refactoring Yourself

改善

Kaizen

Japanese - Continuously making small improvements.

kai = change
zen = good

Refactoring Yourself

Are you a better developer now than you were...

one year ago?

six months ago?

As A Developer

Software development is both an art and an engineering discipline.

Technology will change, but craftsmanship is timeless.

Your Knowledge Portfolio

"An investment in knowledge always pays the best interest." -Benjamin Franklin

Your Knowledge Portfolio

Our knowledge of current technology is an expiring asset.

The value diminishes over time (like tickets to the Superbowl, etc).

Your Knowledge Portfolio

- Know your strengths
- Identify your weaknesses and faults
- Never stop learning
- Always challenge yourself on new projects
- Be diverse

Be Passionate About Your Craft

"Any fool can write code that a computer can understand. Good programmers write code that humans understand." -Martin Fowler

Be Passionate About Your Craft

- Take pride in how your code looks
- Consistency and readability are hugely important
- Avoid "religious wars" about code formatting, etc
- Sign your work

Be Passionate About Your Craft

Bad:

```
if (example.a == ONE_THING && example.b == SOMETHING_ELSE && example.c == WHATEVER)
{
      // Do something.
}
```

Be Passionate About Your Craft

Good:

```
/**
 * Determines whether or not a given <code>Example</code> is valid
 * by assessing the values of <code>a</code>, <code>b</code>, and
 * <code>c</code>.
 *
 * @return A value of <code>true</code> if the conditions are met,
 * otherwise <code>false</code>.
 *
 * @see #ONE_THING
 * @see #SOMETHING_ELSE
 * @see #WHATEVER
 */
public function isExampleValid(example:Example):Boolean
{
      var isA:Boolean = (example.a == ONE_THING);
      var isB:Boolean = (example.b == SOMETHING_ELSE);
      var isC:Boolean = (example.c == WHATEVER);

      return (isA && isB && isC);
}


if (isExampleValid(example))
{
      // Do something.
}
```

Be Passionate About Your Craft

- Write self-documenting, well-commented code
- Settle on and use a common set of standards

http://opensource.adobe.com/wiki/display/flexsdk/Coding+Conventions

Simplicity

"Everything should be made as simple as possible, but not simpler." -Alfred Einstein

Simplicity

- Complexity is easy to achieve
- Keeping a design simple can be surprisingly difficult

Simplicity

- Only design for "real" requirements
- Never add functionality before it is scheduled
- Only 10% of extra stuff will ever get used, so you are wasting 90% of your time

Simplicity

- A function should carryout a single, specific task
- In general, a function larger than 15 lines may be doing too much

Simplicity

- A class should have only one purpose
- In general, a class larger than 300 lines may be doing too much

Avoiding Software Entropy

What is software entropy?

Defines the measure of disorder in software systems.

Avoiding Software Entropy

A few laws have been suggested:

- A computer program that is used will be modified
- When a program is modified, its complexity will increase

Avoiding Software Entropy

Complexity leads to:

- Software that is more difficult to maintain
- An increase in the cost of ownership

Avoiding Software Entropy

Solutions:

- Fixing Broken Windows
- Code Refactoring

Fixing Broken Windows

- Crack down on the little stuff to prevent the big stuff
- Take action to prevent further damage and show that you are on top of things

Code Refactoring

Improving code without changing its overall result.

Code Refactoring

Refactoring is not:

- Adding new features
- Fixing bugs

Code Refactoring

Refactor early, refactor often.

- As requirements begin to change: refactor
- After a code review: refactor

Code Refactoring

Targets for refactoring:

- Duplication
- Poor architecture
- Outdated knowledge/logic
- Performance

Code Refactoring

How to explain/justify to your boss or client?

Surgery analogy!

- Catch it early = simple surgery
- Catch it late = dangerous surgery and costly

Code Refactoring

Tips:

- Keep a public TODO list of things that need to be refactored
- Take small steps and test often (unit tests and test harnesses recommended)

Code Refactoring

As any good dentist will tell you:

If it hurts now, it is going to hurt even worse later.

You Can't Write Perfect Software

- Great software today > "perfect" software tomorrow
- Knowing when to stop is important

Communication

The best ideas are worthless if they can't be communicated.

Communication

In general:

- Use email so there is a paper trail (avoid hallway conversations)
- Get back to people. "I'll get back to you later." is better than nothing at all
- Keep your audience in mind

Communication

With your development team:

- Assign a platform lead
- Create a wiki to track wireframes, specifications, and issues
- Leverage a version control system
- Have regular status meetings

Communication

Don’t refrain from asking questions:

- Why are we doing it this way?
- Is there a better or easier way?
- What problems could result from this?

Estimating

- Huge part of software development
- Helps prevent surprises down the road by forcing you to think about things up front

Estimating

When a project manager asks for an estimate:

"I'll get back to you."

Estimating

Best place to start:

Ask someone who has already done it.

Estimating

Don't let your ego screw you over.

- Give good, honest estimates
- Don't try to impress others with a low number and then drop the ball

Estimating

- Pad to be safe
- Take the amount of time that you think it will take you and multiply that by 2

Estimating

- Unit of measurement is important in expressing accuracy (125 days = 25 weeks = 6 months)
- Explicitly declare assumptions
- Record your estimates and compare to actual time spent after the project is completed

Take Responsibility

- Admit your mistakes
- Offer options, not excuses

Don't Be An Asshole

- Your skills won't get you far if you're an asshole
- Know your role within a team
- Don't make it personal
- Don't prescribe your glasses for someone else

Seek Feedback

- The people that work closely with you are the best judges of your abilities
- Request peer reviews
- Seek out people you trust and tell them to be honest

Seek Feedback

- Take action based on the feedback that you receive
- Your critics are never 100% wrong
- Debate is healthy if both sides are open to change

Work/Life Balance

- Balance your professional and personal life
- The best developers find a way to sustain balance and avoid burnout

Questions?

Thank You!

Ryan Taylor
ryan@boostworthy.com

www.boostworthy.com
www.boostworthy.com/blog

Twitter: @boostworthy
AIM: boostworthy
GTalk: rtaylor11@gmail.com