r7k

Refactoring - Team Strategies

Michael Feathers
R7K Research & Conveyance

A series of *small* steps, each of which changes a program's internal structure without changing its external behavior.

A series of *small* steps, each of which changes a program's internal structure without changing its external behavior.

The goal is to make systems more maintainable over time

A series of *small* steps, each of which changes a program's internal structure without changing its external behavior.

The goal is to make systems more maintainable over time

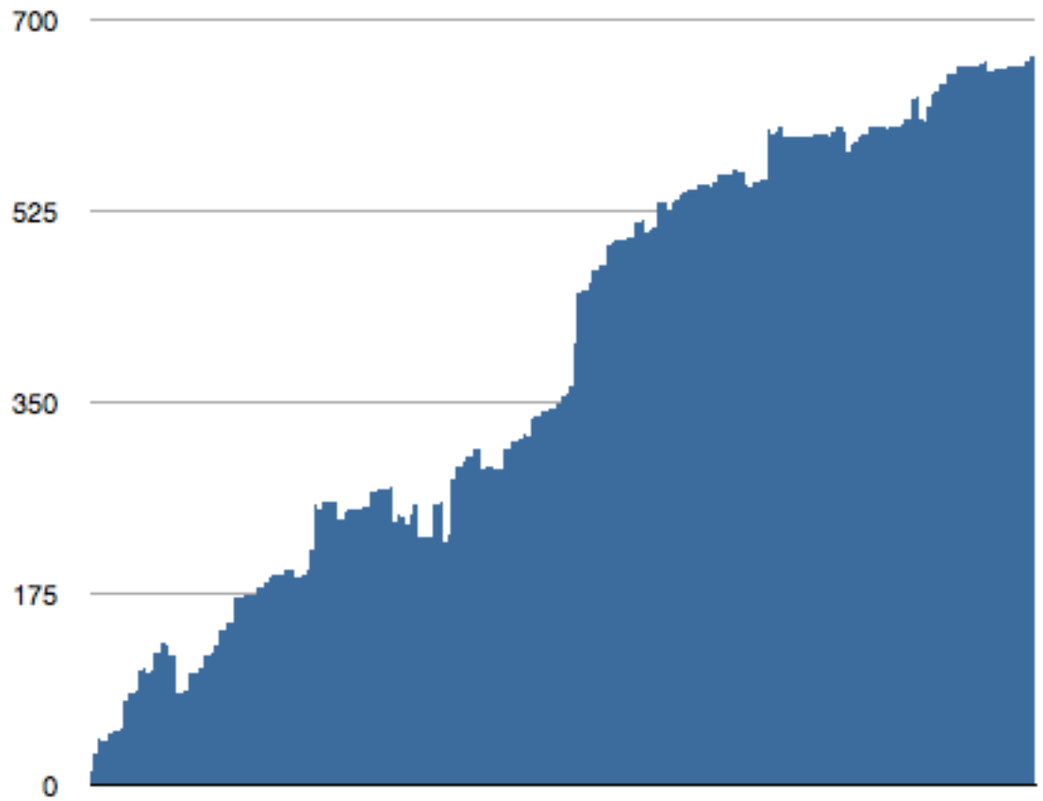
In the large, this is a team activity

Behavioral Economics and Code

Is it easier to add code to an existing method than to create a new method?

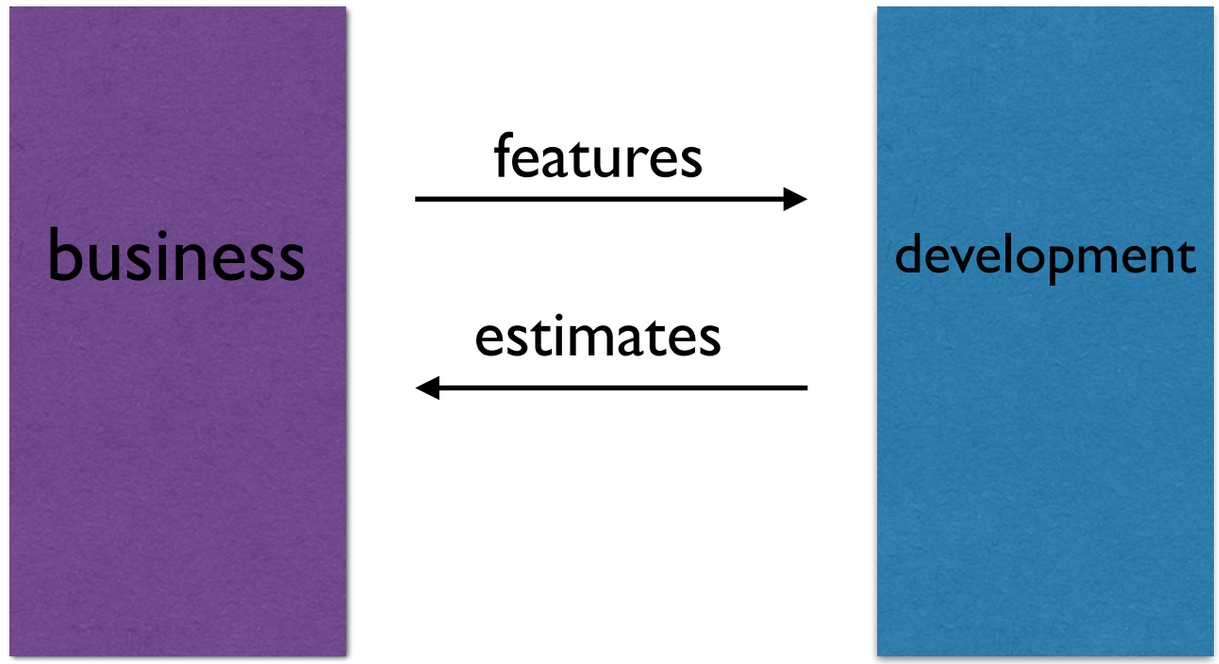
Is it easier to add a method to an existing class than to add a new class?

We should not be surprised by what we see in our code


700
525
350
175
0

business

development


business
features
estimates
development

Joel on Software

The Law of Leaky Abstractions

by Joel Spolsky

Monday, November 11, 2002

There's a key piece of magic in the engineering of the Internet which you rely on every single day. It happens in the TCP protocol, one of the fundamental building blocks of the Internet.

TCP is a way to transmit data that is *reliable*. By this I mean: if you send a message over a network using TCP, it will arrive, and it won't be garbled or corrupted.

Technical Debt - the amount of effort it takes to refactor your code to make it easy to add the next feature *non-invasively*.

```
public void addEvent(Event event) {
    event.added();
    events.add(event);
    sendMail("jacques@spg1.com", "Event Notification", event.toString());
    display.showEvent(event);
}
```

```
public void addEvent(Event event) {
    event.added();
    events.add(event);
    sendMail("jacques@spg1.com", "Event Notification", event.toString());
    eventAuditing.eventReceived(event);
    display.showEvent(event);
}
```

```
public void addEvent(Event event) {
    event.added();
    events.add(event);
    eventProcessors.forEach(p -> p.runOn(event))
}
```

Technical Debt - the amount of effort it takes to refactor your code to make it easy to add the next feature *non-invasively*.

Open/Closed

In object-oriented programming, the **open/closed principle** states "software entities (classes, modules, functions, etc.) should be **open** for extension, but **closed** for modification"; that is, such an entity can allow its behaviour to be extended without modifying its source code.

Practices

Practice

Design Decision Cards

Practice

Design Decision Cards

Maintain cards for each of the design decisions you make that you may consider revisiting someday. Periodically re-estimate them to consider feasibility

Practice

Feature Trend Cards

Practice

Feature Trend Cards

Hypothesize a couple of features that you will never add to your code. Task them and estimate them periodically. See the debt trend for areas they touch.

Practice

Scratch Refactoring

Practice

Scratch Refactoring

Refactor massively in an editor. Emphasize extractions, and moves. Don't worry about compilation. Never check it in. Use the experience to explore

Practice

Suggestive Refactoring

Practice

Suggestive Refactoring

Create small refactoring stories based upon and add them to the backlog

Practice

Split Preparatory Refactorings

Practice

Split Preparatory Refactorings

Highlight refactoring within a team by making it a separate task done by different people. The handoff forces discussion

Practice

Privileged Abstractions

Practice

Privileged Abstractions

Select the abstractions that you consider primary in the system and document them. Have conversations around them

Practice

Limited WIP Refactoring

Practice

Limited WIP Refactoring

Never have more than 1 or 2 large scale refactorings in progress at once. This forces focus and emphasizes completion

Practice

Architectural Mapping

Practice

Architectural Mapping

Diagram the system you are working as it it were the terrain of an old country. Document the dragons. Have a common team vision of the place where the best code resides

Practice

Silent Alarms

Practice

Silent Alarms

Don't have check-in gates. Let people make mistakes.

Investigate the mistakes off-line and see why they happened. Then, intervene

Practice

Scrape the Pan

Practice

Scrape the Pan

Global mutable state binds code in place. Consolidate the state to make it possible pry out particular subsystems, making them independently testable

```
public class XXXFactories {

    public static ResourceFactory resourceFactory = new ResourceFactory() {
        public Resource makeResource(int id) {
            return new Resource(id);
        }
    };

    ...
}
```

```
public class XXXRepositories {
    public final static PartialFillRepository partialFills = new PartialFillRepository();
    …
}

public class PartialFillRepository {
    private Hashtable partials = new Hashtable();

    public PartialFill getPartialFill(int id, String symbol) {
        String key = id + " " + symbol;
        Object partial = partials.get(key);
        if (partial == null)
            throw new InvalidPartialFill(id, symbol);
        return (PartialFill)partial;
    }

    public void resetForTest() {
        partials = new Hashtable();
    }
}
```

Practice

Direction Tagging

Practice

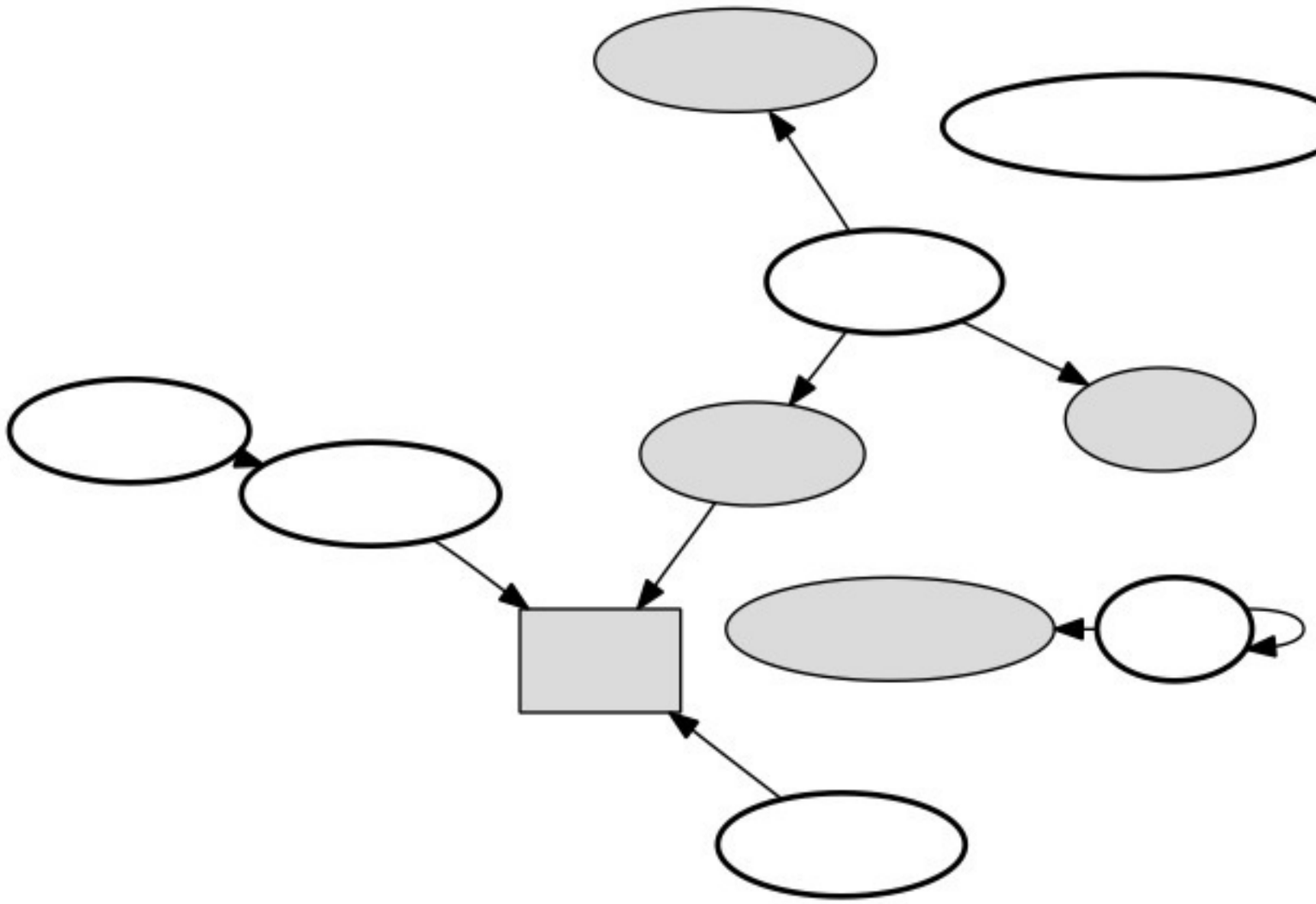
Direction Tagging

Create tags for areas that need work. Make them orthogonal and embed them in the code. They do not go stale as comments do. Tackle then in a limited WIP manner

Practice

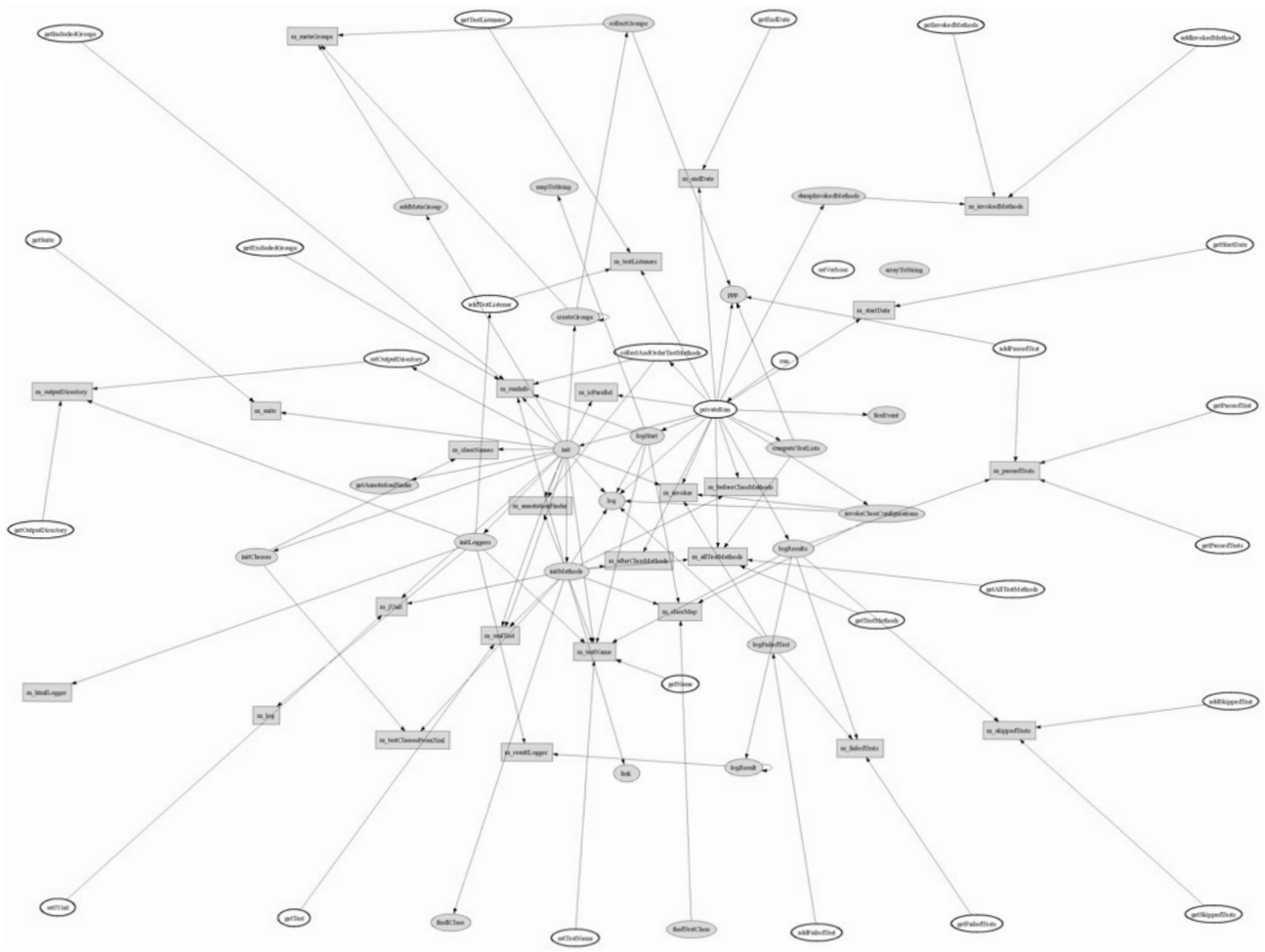
Transparent Design Quality

Transparent Design Quality



Practice

Transparent Design Quality



http://r7krecon.com