Is design now a subset of refactoring?

Asked 15 years, 11 months ago Modified 15 years, 9 months ago

Viewed 549 times

6

Looking at the cool new principles of software development:

- Agile
- You Ain't Gonna Need It
- Less As A Competitive Advantage
- Behaviour-Driven Development
- The Evils Of Premature Optimization

The New Way seems to be to dive in and write what you need to achieve the first iteration of scope objectives, and refactor as necessary to have elegant solutions. Your codebase grows incrementally, and never has a big planning/hierarchical design stage. That, to me, suggests that software design (worthy though it is) has been subsumed into refactoring, because that's where the elegant code comes from, not the incremental additions to functionality.

Am I wrong?

language-agnostic refactoring

Share

Improve this question

Follow

edited Mar 14, 2009 at 18:11

community wiki
Phil H

Did you mean "You Ain't Gonna Need It" to link somewhere else than "Agile Unified Process"? – Jonik Jan 23, 2009 at 12:25

9 Answers

Sorted by: Highest score (default)

▲

6

▼

Well the trouble with refactoring is that you need to know good design before you jump head first in. BDD/TDD are supposed to make the design emerge but without other factors, such as Domain knowledge and technical competence you will end up in trouble.

Share Improve this answer

Follow

answered Jan 23, 2009 at 11:16

community wiki
Johnno Nolan

6

I'd say that doing it the way you describe is a recipe for disaster, I would still recommend to do the overall design up front. Of course during the project you will need to change the design, it is never set in stone, flexibility is a must! (That's where the refactoring has to come in). I would also recommend to do the more detailed design for a module just before you start coding it.

But a solid general design is worth its weight in gold: It gives all developers a common base from which to start, a common idea or perhaps vision, of the goal.

Without that everybody will do as he/she thinks best, with the result that everybody does things in a different way. And suddenly you have to refactor a lot, just to align everybody to what has emerged as the apps architecture. The resulting code is ... not very elegant.

Share Improve this answer Follow

answered Jan 23, 2009 at 11:25

community wiki
Treb

4

Wrong? Partially.

"Your codebase grows incrementally, and never has a big planning/hierarchical design stage."

Correct.

"That, to me, suggests that software design (worthy though it is) has been subsumed into refactoring..."

Not quite correct.

There's a huge gulf between Big Design Up Front (BDUF) and a more Agile design approach.

BDUF dictates that all design is completed before any coding. This is still popular (just read an RFP yesterday which absolutely required all design be reviewed by the customer before any coding could begin. Sigh.)

Agile suggests that perhaps *all* design isn't a helpful goal. You need to do *enough* design that TDD will work. You can't, for example, start TDD until you have a working infrastructure that allows someone to write tests and incrementally evolve a solution knowing that there won't be a weird production deployment problem to solve.

Design is still king. Agile Design is better than monolithic design.

A consequence of Agile Design is YAGNI, DRY and Less-is-More. These don't replace design, they're a consequence of how you prioritize and do design.

BDD and TDD are ways to structure your time so you have focus on what people need, what they do and what really matters. TDD, in particular, focuses on testable behaviors of the software. Not zero-value nuance, but actual behavior.

Premature optimization is interesting, but unrelated. Even Agile teams can run down a low-value rat-hole pursuing a nuance or optimization that doesn't add any value. Premature optimization is a habit of overthinking (== "hand wringing") a technology choice without facts about actual performance.

Agile is supposed to help you focus on the big picture: **What actual people will actually do with the actual software** and avoid technology rat-holes.

It doesn't replace engineering. It refocuses it.

Share Improve this answer Follow

answered Jan 23, 2009 at 12:14

community wiki
S.Lott

2

✓

Reminds me of the following quote:

> The goal is clean code that works. [...] First we'll solve the "that works" part of the problem. Then we'll solve the "clean code" part. This is the opposite of architecture-driven development, where you solve "clean code" first, then scramble around trying to integrate into the design the things you learn as you solve the "that works" problem. (Kent Beck, "TDD by Example")

Share Improve this answer Follow

answered Jan 23, 2009 at 12:15

community wiki
Fabian Steeg

▲
2
▼

There's nothing in the agile manifesto, which says that you're not allowed to think before you act. Of course you can be agile and still design up front. Architecture is best designed, so before you start coding/refactoring, you should have some ideas as to how your application should be constructed.

The point is you don't have to complete each and every step before moving on. Do as much design as you need to get started.

When you have code, you can refactor as needed, but changing the fundamental architecture through refactoring becomes hard if you start from a simple dummy application every time.

Share Improve this answer Follow

answered Jan 23, 2009 at 11:44

community wiki
Brian Rasmussen

2

The nature of design is changing, I'd say the new way is to think before coding, but just about what will be implemented in the next iteration. See "Is design dead".

Design for today, code for tomorrow.

Share Improve this answer Follow

answered Jan 23, 2009 at 12:47

community wiki
philant

2

Depends what you're designing. If it's a complex algorithm thats going to be the next video compression standard, you can iteratate and refactor 'till the cows come home and it isn't going to get any faster. The perfomance comes from design.

Similarly, if you are writing a very large application, that will grow through regular releases, you need to put in place an architecture that will support growth, and this will be by design. You can go down a long road by jumping head first into code just to discover a dead end.

> Am I wrong?

IMHO, pretty much.

Edit: The reason I make many design decisions early in the process rather than mid flow is this can often be the cheapest time to do so. For example, if we start writing an application using platform dependent technology early on, it may be a very expensive decision to reverse. If we take time to consider the platforms we want to support before starting to code, it is much cheaper. We can't and won't get everything right first time, but this does not relieve us of the duty of exploring all important design choices up front. Every tried refactoring a MS MDI program to MVC? I have, and wouldn't recommend it ;)

Share Improve this answer Follow

edited Jan 23, 2009 at 13:21

community wiki
3 revs
Shane MacLaughlin

Isn't design a part of refactoring? Just larger scale? – Phil H Jan 23, 2009 at 12:34

You'd have to define what you mean by refactoring. My understanding of refactoring is iteratively improving code quality without affecting its functionality. Iteratively improving design to improve functionality before starting to code is something very different, often much cheaper and quicker. – SmacL Jan 23, 2009 at 13:10

New? No. I was reading about agile ten years ago. And agile is just the crystalization of ideas that have been

1

around for longer than that. It's hardly new, it just hasn't diffused everywhere yet.

As for your view of design, I think there's still a place for an overarching idea and some up front design. It's the waterfall notion that you can't make a move before requirements and design are 100% complete that has been discredited everywhere but in the large firms that still cling to it.

Share Improve this answer Follow

answered Jan 23, 2009 at 11:08

community wiki
duffymo

0

Let's see if we can get a definition. I'm going to suggest that there may be a book we could reference. How about [the one by Martin Fowler]?

"Refactoring: Improving the Design of Existing Code"

Now let's take as an example the TDD mantra:

until done do

- Red (we wrote a test and it failed because there was nothing that could pass it)
- Green (we designed and wrote some code and the test now succeeds)

- Refactor (we need to integrate the design we did with everything else)

end

I know the Agile books tend to just say "write enough code to pass the test", but there's design implicit in that statement. By necessity. Choosing a variable name is a design decision. Not a big one, but a decision nonetheless. naming a function or method is a slightly bigger one, and so on up the food chain.

There's nothing in Agile that can ensure you always make *good* design decisions. Nothing in waterfall or any other process either. Agile does assert that you can't figure it all out upfront and tries - with some success in my experience - to give you a set of tools to help you make better decisions throughout the whole exercise.

Share Improve this answer Follow

answered Jan 23, 2009 at 12:43

community wiki
Mike Woodhouse