Best Practices - Design before coding [closed]

Asked 16 years ago Modified 13 years, 1 month ago Viewed 8k times

45

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 6 years ago.

Improve this question

I'm curious How do you people think ? (I mean a way of thinking) about design architecture of your Libraries, Systems, Frameworks, etc. before start coding it.

I recently find my self feeling pain in what I've done, and practically every time I want to start everything from scratch..

I do design before, painting some schemes on the paper and imagine how it will work, but maybe I do it in a wrong way ?

For example how do you decide what Interfaces you will need, or how everything will be connected in a best way ?

(I had a problem some day ago, my friend asked me a library what I've done some time ago, and instead of giving him just one file, I had to give him about 3-4 files, and that's because they're connected in some way.. but not in the right one I think :) so it was my mistake in design..)

architecture

Share

Improve this question

Follow

edited Nov 4, 2011 at 11:59

Mat

207k ● 41 ● 400 ● 415

asked Dec 17, 2008 at 8:34

Lukas Šalkauskas

14.3k ● 20 ● 63 ● 77

About this 3-4 files, thats perfectly normal. You didn't expect to do a deliverable lib did you? That's when you refactor and improve the layout of the lib. – OscarRyz Dec 17, 2008 at 8:47

Amazingly, it's taken me a good chunk of forever to find this question. I do the exact same thing. – new123456 Sep 27, 2011 at 21:34

11 Answers

Sorted by: Highest score (default)

I usually do enough analysis of the problem domain on paper/white board to get a good enough understanding of the problem domain to start writing code. I rarely draw implementation or class diagrams on paper. A key technique I've found to achieve better design is to not get too attached to the code you write. If I don't like it, I delete, rename, move and shuffle it around until it expresses a good enough solution to what I'm trying to solve. Sounds easy? Not at all! But with good "coding" tools, actually writing the code is not the major effort. Write some, refactor, delete, write again...

Good design almost never start out good. **It evolves to good.** Accepting this makes it easier to work in small steps without getting frustrated why the design isn't "perfect". In order for this process to work you have to posses good design skills though. The point being, even excellent designers don't get it right the first time.

Many times, I thought I understood the problem domain when I started, but I didn't. I then go back to the white board, talk to someone or read up on the problem domain if I realize I don't understand it well enough. I then go back to the code.

It is a very iterative processes.

An interesting question to ask when dealing with how programmers think, is how they developed their way of thinking. Personally, my way of thinking has evolved over the years, but a few events have had profound influence

on the way I develop software. The most important among them have been to design software with people who are expert designers. Nothing has influenced me more than spending iterations with great designers. Another event that has, and still do, affect the way I think is going back and look at software I wrote some time back.

Share Improve this answer Follow

edited Dec 20, 2008 at 20:01

answered Dec 17, 2008 at 11:16

Kim Major

3,711 ● 1 ● 24 ● 20

6

For an object oriented approach, I find it is generally a good idea to step away from the user interface for a bit, and focus on what entities (objects) will exist in the system, and what properties and actions are appropriate.

Drawing on a whiteboard or large piece of paper, using different colors to identify various characteristics is also a good idea. Post-it notes are also a nice way to visualise your model.

Even if I spend a lot of time to think through a design very carefully, I ALWAYS end up changing it as I go. So it's good keep in mind that your design WILL change as you make a decision about how to document your design.

Share Improve this answer Follow

edited Dec 17, 2008 at 8:45

answered Dec 17, 2008 at 8:39

Ola Eldøy

5,938 ● 7 ● 52 ● 84

5

I mostly start with a box of cards, and write down the concepts of the domain I want to model. Sometimes I use a mindmap for that.

Take a look at the stakeholders, and what they want to accomplish. It is important to start there, because it allows you to prioritize correctly (i.e. not on the most interesting technical part, but on the part with the most business value)

The thinking about design is mostly written down in tests. Written before the code to implement it. Start with the stakeholders end-goals, work from there backwards to the beginning (Time-inversion). That ensures that you concentrate on what, and less on how. Interaction between objects is more important to get right than object attributes.

The other part is mostly on the white board. A compact digital camera is standard equipment these days.

Doing a framework before you have three implementations that need it is a bad practice. If you have

to, be prepared for significant interface (and implementation) change.

See:

- http://www.robvens.nl/en/articles/computers/29-time-inversion-pattern
- http://www.robvens.nl/en/articles/computers/28-active-passive-pattern

Share Improve this answer Follow

edited Dec 17, 2008 at 8:58

answered Dec 17, 2008 at 8:41

Stephan Eggermont
15.9k ● 1 ● 40 ● 65

reflektis.com/blog/time-inversion-pattern and reflektis.com/blog/active-passive-pattern for who (like me) was not able to find the linked articles ;) – Kamafeather May 10, 2019 at 19:06

Thanks. Links sometimes go stale after a decade... – Stephan Eggermont May 16, 2019 at 11:59

4

It has to be a balance.

If you try to design everything up front with lots of pictures on a white board, then you'll probably miss some details when it actually comes round to coding it up.

If you start hacking away at some small part of the system, you'll probably lose sight of the "big picture" and end up with a poor design.

As you're looking at libraries, you want them to be re-usable as much as possible. So for the initial design, think of the most general cases you can *among the intended uses of your library you already know about* -- don't worry too much at this stage about hypothetical future uses that might well never happen. Code up that design with unit tests and refactor as you learn more and find problems with the design.

Try to put as little specific knowledge about the library's users into the library itself. Then, with luck you will end up with something re-usable and won't automatically want to start from scratch next time.

Share Improve this answer

Follow

answered Dec 17, 2008 at 8:53

Paul Stephenson

69.3k ● 9 ● 52 ● 51

Yes. Balanced it should be! – kuhajeyan Nov 13, 2018 at 10:11

4

I usually dedicate about 2 - 4 hours to do the design of my app and then write it down in a notebook.

Then I start coding, and each time I thing get complicated (because something is not in the right place) I refactor. Move one class to another package, or extract method,

etc. etc. When the "design" feels right then I can move on.

This way I avoid "analysis paralysis" that use to happen to me quite a lot. Many times when I over designed upfront I ended up not using some artifacts.

Thus I took this more "agile" approach.

Sketch up the design very quickly and "refine" (by refactoring) in the run.

Of course this is possible for small apps (2 - 4 weeks long)

I would suggest you to take a read to this article: Is design dead from Martin Fowler. It is a little lengthy but worth reading.

EDIT

Additional link (slightly offtopic) How to design a good API and why it matters by Joshua Bloch. Talk about the relevance of design when you have an audience for your API. Summary, start as private as possible and the grow from there.

Share Improve this answer Follow

edited Dec 17, 2008 at 8:56

answered Dec 17, 2008 at 8:43

OscarRyz

I follow a very loose version of Rational Unified Process.

First start of with a vision document which states clearly what you're trying to do, who your are doing it for and maybe some key details about the proposed methods/algorithms. It doesn't need to be fancy and could even be a one liner for a simple system "A system for predicting winning lottery numbers, for personal use only, based on latest research from Hogwarts school.", for a more complex system it could be about five pages, but no more. This is not the design but more like a wish list.

Then do some use cases/stories. These should be plain natural language text descriptions of all interactions with the external world. Minimalism is the key here; the purpose of use cases is to identify all the required functionality and only the required functionality. Also I find this is where most of the creativity occurs.

A use case might start of as:

```
User presses the "magic" button.
System displays next weeks winning number.
User writes down number.
User buys lottery ticket.
Lottery draws winning number.
User claims money
lottery pays up.
```

After much working around it ends up as:-

```
User presses "magic" button
System selects next weeks numbers.
System logs on to lottery system.
System enters winning numbers.
Lottery selects winning numbers.
Lottery transfers winnings to users account.
User spends money.
```

Once you have done your use cases you can fire up your development environment and the various classes and interactions will just fall into place.

Share Improve this answer

Follow

edited Oct 20, 2011 at 1:20

Iterator

20.6k ● 12 ● 78 ● 113

answered Dec 17, 2008 at 9:07

James Anderson

27.4k ● 7 ● 54 ● 80

These are not use cases: what stakeholder goal is (partly) accomplished? -1 – Stephan Eggermont Dec 17, 2008 at 9:11

1 Space is a bit limited for doing the full use case "stories" might have been a better term. The main point was to illustrate how drasticlally a design can change when specifing the use cases and how little these changes cost. – James Anderson Dec 17, 2008 at 11:14

▲

3

Although not an entire answer to your question, I sometimes find the easiest way to get into gear for a project as it were is to find a small piece of isolated

functionality to get started on and work on that whilst also thinking about the bigger picture.

That way you don't get too hung up on every minute detail and are being productive and giving yourself breathing space to see what you need to do with clarity.

Like I say, not an answer.

Share Improve this answer

Follow

answered Dec 17, 2008 at 8:39

navitronic

563 ● 3 ● 15

2

The problem is when you begin a new project, it tends to be new (isn't that obvious now). And in general one doesn't understand new stuff off the bat, unless your a very specialized consultant doing the exact same thing over and over again, which kinda sounds freaky...

Because you're inevitably new to the system your making, you're design and code aren't going to be perfect the first time around, so you reiterate the design & refactor the code, until its ready.

Share Improve this answer

Follow

answered Dec 17, 2008 at 8:42

Robert Gould

69.7k ● 61 ● 191 ● 275

I prefer a bottom-up design. If there is not an application framework in place, I tend to assembly or build it first. A

1

good application framework is portable and application-agnostic, largely addressing cross cutting concerns. It will normally contain things such as logging, exception handling, validation helpers, extension methods, etc.

From there, I like to go with a largely DDD (Domain Driven Design) approach. Interviewing business users, if needed, to construct the domain model. Once the domain model, and subsequent DAL (Data Access Layer) is taken care of, I move to the business logic layer (BLL).

I personally try to stay as removed from front-end implementation as possible, since it is defintely not a strong point of mine. Definitely one reason that I love unit testing is that I can focus on core functionality, and be able to test that functionality without jumping into premature interface decisions.

Share Improve this answer

Follow

answered Dec 17, 2008 at 16:33

Joseph Ferris

12.7k ● 3 ● 48 ● 73

0

Open question. There will be nearly as much answers as posters. 1) have a look at many software engeneering books. Some argue with good design the rest is a snap. That's a straigh lie 2) See how intrusive diverse Frameworks are, You better have to use them the intended way otherwise you better implement the stuff again for your needs 3) Design would need a constantly change as any writing. You see the sentences but you feel they do not fit fully. So you rewrite it. So every design

must take into account that things are different as they seem as you have written the design.

Regards Friedrich

Share Improve this answer

Follow

answered Dec 17, 2008 at 8:40

Friedrich

5,996 ● 1 ● 29 ● 46

Could you rephrase this to make it clearer? – Draemon Dec 18, 2008 at 5:24

0

When doing any major projects like game development I tend to try and carefully structure my code and make non-functional dummy functions and classes right from the start. From the Main code file I split everything into classes within other modules and sub-modules, if necessary. For example; My main code page would consist of nothing more than includes and calls to class initializers for my main modules. Here's an example of the init.py for a game I'm making in Python right now;

from commonheaders import *

```
if arguements.debug:
    debug.init()

try:
    graphics.init()
    sound.init()
    physics.init()
    input.init()
    loop.start("scene1")
```

```
except Exception, e:
    print "Error!"
    print e
```

It's very minimal and easy to understand, but what I like most about it is that it gives very defined separation between the the various main code aspects. If I get frustrated with something in the graphics class I can just go work in the sound class or something else to get other stuff done and keep graphics out of my mind for awhile until the clouds pass. On all class init() calls there is optional arguements, such as widht/height for graphics, but I usually leave them out of there and either set them in the commonheaders.py file or through command line switches.

I try not to let any of my modules get larger than about 100 lines before thinking about what I can cut out and place into it's own module. It makes everything much cleaner and it's much easier to find what you are looking for when it's split up like that. I've worked with various open source projects before and seen single modules in the tens of thousands of lines before...there was so much redundant code that could easily have been offloaded to helper functions. I use comments where necessary, but I try to keep my code structured in such a way that you don't really even need to read the comments.

I really like having the non-functional framework built from the start as it makes it very easy to see what still needs to be done and what can be improved in terms of structure. It's better than rushing off to write thousands of lines of

codes only to figure out that there was a more elegant and functional way of doing it that would require a total restructuring of the code.

Share Improve this answer

Follow

answered Dec 17, 2008 at 10:10

Stephen Belanger

6,341 ● 12 ● 47 ● 49