How do you begin designing a large system? [closed]

Asked 16 years, 4 months ago Modified 12 years, 8 months ago
Viewed 5k times

10

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

It's been mentioned to me that I'll be the sole developer behind a large new system. Among other things I'll be designing a UI and database schema.

I'm sure I'll receive some guidance, but I'd like to be able to knock their socks off. What can I do in the meantime to prepare, and what will I need to keep in mind when I sit down at my computer with the spec?

A few things to keep in mind: I'm a college student at my first real programming job. I'll be using Java. We already have SCM set up with automated testing, etc...so tools are not an issue.

java oop architecture

Share

Improve this question

Follow

edited Apr 6, 2012 at 9:29

Brett Veenstra

48.4k ● 18 ● 71 ● 86

asked Aug 19, 2008 at 4:26

Jeff

3,262 ● 3 ● 24 ● 14

10 Answers

Sorted by: Highest score (default)

11

Do you know much about OOP? If so, look into Spring and Hibernate to keep your implementation clean and orthogonal. If you get that, you should find TDD a good way to keep your design compact and lean, especially since you have "automated testing" up and running.

UPDATE: Looking at the first slew of answers, I couldn't disagree more. Particularly in the Java space, you should find plenty of mentors/resources on working out your application with Objects, **not a database-centric approach**. Database design is typically the first step for Microsoft folks (which I do daily, but am in a recovery

program, er, Alt.Net). If you keep the focus on what you need to deliver to a customer and let your ORM figure out how to persist your objects, your design should be better.

Share Improve this answer Follow

edited Jun 23, 2011 at 19:31

answered Aug 19, 2008 at 4:32

Brett Veenstra
48.4k ● 18 ● 71 ● 86

+1 Let JPA handle the data layer, concentrate on the java implementation which is compilable, testable and comprehensible by your editor – klonq Apr 6, 2012 at 9:32

5

This sounds very much like my first job. Straight out of university, I was asked to design the database and business logic layer, while other people would take care of the UI. Meanwhile the boss was looking over my shoulder, unwilling to let go of what used to be his baby and was now mine, and poking his finger in it. Three years later, developers were fleeing the company and we were still X months away from actually selling anything.

The big mistake was in being too ambitious. If this is your first job, you *will* make mistakes and you *will* need to change how things work long after you've written them. We had all sorts of features that made the system more complicated than it needed to be, both on the database level and in the API that it presented to other developers.

In the end, the whole thing was just far too complicated to support all at once and just died.

So my advice:

1. If you're not sure about taking on such a big job single-handed, don't. Tell your employers, and get them to find or hire somebody for you to work with who can help you out. If people need to be added to the project, then it should be done near the start rather than after stuff starts going wrong.
2. Think very carefully about what the product is for, and to boil it down to the **simplest** set of requirements you can think of. If the people giving you the spec aren't technical, try to see past what they've written to what will actually work and make money. Talk to customers and salespeople, and understand the market.
3. There's no shame in admitting you're wrong. If it turns out that the entire system needs to be rewritten, because you made some mistake in your first version, then it's better to admit this as soon as possible so you can get to it. Correspondingly, don't try to make an architecture that can anticipate every possible contingency in your first version, because you don't know what every contingency is and will just get it wrong. Write once with an eye to throwing away and starting again - you may not have to, the first version may be fine, but admit it if you do.

Share Improve this answer

Follow

answered Aug 19, 2008 at 10:34

Marcus Downing

10.1k ● 12 ● 65 ● 86

3

I also disagree about starting with the database. The DB is simply an artifact of how your business objects are persisted. I don't know of an equivalent in Java, but .Net has stellar tools such as SubSonic that allow your DB design to stay fluid as you iterate through your business objects design. I'd say first and foremost (even before deciding on what technologies to introduce) focus on the process and identify your nouns and verbs ... then build out from those abstractions. Hey, it really does work in the "real world", just like OOP 101 taught you!

Share Improve this answer

Follow

answered Aug 19, 2008 at 6:08

xanadont

7,604 ● 6 ● 38 ● 51

2

Before you start coding, plan out your database schema - everything else will flow from that. Getting the database reasonably correct early on will save you time and headaches later.

Share Improve this answer

Follow

answered Aug 19, 2008 at 4:31

Kyle Cronin

79k ● 45 ● 151 ● 167

The main thing is being able to abstract the complexity of the system so that you don't get bogged down by it as soon as you start off.

- First read the spec like a story (skimming through it). Don't stop at every requirement to analyze it right there and then. This will allow you to get an overall picture of the system without too many details. At this point you would start identifying the major functional components of the system. Start putting these down (use a mindmap tool if you like).
- Then take each component and start exploding it (and tying each detail with requirements in the spec document). Do this for all components, till you have covered all requirements.
- Now, you should start looking at relationships between the components, and whether there are repetitions of features or functions across the various components (which you can then pull out to create utility components, or such). Around now, you would have a good detailed map of your requirements in your mind.
- NOW, you should think of designing the database, ER diagrams, Class Design, DFDs, deployment, etc.

The problem with doing the last step first is that you can get bogged down in the complexity of your system without really gaining an overall understanding in the first place.

Share Improve this answer

Follow

answered Aug 19, 2008 at 6:03

Vaibhav

11.4k ● 11 ● 53 ● 71

1

I do it the other way around. I find that doing it database-schema-first gets the system stuck in a data-driven-design that is difficult to abstract from persistence. We try to do domain model designs first *and then* base the database schema on those.

And then there's the infrastructure design: the team should settle on conventions on how to structure the program first and foremost. And then we work together to agree first on a design for the common functionality of the system (e.g., things everyone needs like persistence, logging, etc.). This becomes the framework of the system.

We all work on that together first before we split the rest of the functionalities amongst ourselves.

Share Improve this answer

Follow

answered Aug 19, 2008 at 4:39

Jon Limjap

95.3k ● 15 ● 103 ● 153

1

It has been my experience that Java applications (.NET also) that consider the database last are highly likely to perform poorly when placed into a corporate environment. You need to really think about your audience. You didn't say if it was a web app or not. Either

way the infrastructure that you are implementing on is important when considering how you handle your data.

No matter what methodology you consider, how you get and save your data and it's impact on performance should be right up there as one of your #1 priorities.

Share Improve this answer

Follow

answered Aug 19, 2008 at 11:21

bruceatk

5,138 ● 2 ● 27 ● 36

1

I'd suggest thinking about how this application will be used. How will future users work with it? I'm sure you know at least a few things about what this application needs to handle, but my first advice is "think of the user and what he or she needs".

Draw it up on plain paper, thinking of where to section off the code. Remeber not to mix logic with GUI code (common error). This way you will be set to extend your applications reach in the future to servlets and/or applets or whatever platform comes along. Section in layers so that you can respond to large changes faster without rebuilding everything. Layers should not see any other layers than their closest neighbouring layers.

Begin with true core functionallity. All that time consuming fluff (that will make your project 4 weeks late), wont matter much to the wast majority of users. It can be added later once you are sure you can deliver on time.

Btw. Even though this has nothing to do with design I'd just like to say that you won't deliver on time. Make a realistic estimate on time consumption and then double it :-) I assume here that you will not be alone in this project and that people will come and go as the project progresses. You may need to train people midway through the project, people go on holiday / need surgery etc.

Share Improve this answer

Follow

answered Oct 22, 2008 at 15:09

Anders Björklund

0

Split the big system to smaller pieces. And don't think that it's so complex, because it usually isn't. By thinking too complex it just ruins your thoughts and eventually the design. Some point you just realize that you could do the same thing easier, and then you redesign it.

Atleast this has been my major mistake in designing.

Keep it simple!

Share Improve this answer

Follow

answered Aug 19, 2008 at 10:48

Tuoski

1,081 ● 3 ● 15 ● 15

0

I found very insightful ideas about starting a new large project, based on

- common good practices
- Test Driven Development
- and pragmatic approach

in the book [Growing Object-Oriented Software, Guided by Tests](#).

It is still under development, but first 3 chapters may be what You are looking for and IMHO worth reading.

Share Improve this answer

Follow

answered Aug 19, 2008 at 21:31

keiw

1,390 ● 1 ● 9 ● 13