How do I plan an enterprise level web application?

Asked 16 years, 3 months ago Modified 9 years ago Viewed 4k times

10

I'm at a point in my freelance career where I've developed several web applications for small to medium sized businesses that support things such as project management, booking/reservations, and email management.

I like the work but find that eventually my applications get to a point where the overhear for maintenance is very high. I look back at code I wrote 6 months ago and find I have to spend a while just relearning how I originally coded it before I can make a fix or feature additions. I do try to practice using frameworks (I've used Zend Framework before, and am considering Django for my next project)

What techniques or strategies do you use to plan out an application that is capable of handling a lot of users without breaking and still keeping the code clean enough to maintain easily? If anyone has any books or articles they could recommend, that would be greatly appreciated as well.

frameworks

Share

Improve this question

Follow

edited Nov 29, 2015 at 22:57

Wai Ha Lee

8,795 ● 97 ● 59 ● 94

asked Sep 19, 2008 at 23:06

Andy Baird

6,159 ● 4 ● 45 ● 64

6 Answers

Sorted by: Highest score (default)

10

Although there are certainly good articles on that topic, none of them is a substitute of real-world experience.

Maintainability is nothing you can plan straight ahead, except on very small projects. It is something you need to take care of during the whole project. In fact, creating loads of classes and infrastructure code in advance can produce code which is even harder to understand than naive spaghetti code.

So my advise is to **clean up your existing projects**, by continuously refactoring them. Look at the parts which were a pain to change, and strive for simpler solutions that are easier to understand and to adjust. If the code is even too bad for that, consider rewriting it from scratch.

Don't start new projects and expect them to succeed, just because your read some more articles or used a new framework. Instead, identify the failures of your *existing* projects and fix their *specific* problems. Whenever you

need to change your code, ask yourself how to restructure it to support similar changes in the future. This is what you need to do anyway, because there *will* be similar changes in the future.

By doing those refactorings you'll stumble across various *specific questions* you can ask and read articles about. That way you'll learn more than by just asking general questions and reading general articles about maintenance and frameworks.

Start cleaning up your code **today**. Don't defer it to your future projects.

(The same is true for documentation. Everyone's first docs were very bad. After several months they turn out to be too verbose and filled with unimportant stuff. So complement the documentation with solutions to the problems you *really* had, because chances are good that next year you'll be confronted with a similar problem. Those experiences will improve your writing style more than any "how to write good" style guide.)

Share Improve this answer

Follow

answered Sep 19, 2008 at 23:48

vog

25.5k ● 11 ● 63 ● 81

4

I'd honestly recommend looking at Martin Fowlers Patterns of Enterprise Application Architecture. It discusses a lot of ways to make your application more

organized and maintainable. In addition, I would recommend using unit testing to give you better comprehension of your code. Kent Beck's book on Test Driven Development is a great resource for learning how to address change to your code through unit tests.

Share Improve this answer

Follow

answered Sep 19, 2008 at 23:18

Michael Brown

9,143 ● 1 ● 32 ● 37

3

To improve the maintainability you could:

- If you are the sole developer then adopt a coding style and stick to it. That will give you confidence later when navigating through your own code about things you could have possibly done and the things that you absolutely wouldn't. Being confident where to look and what to look for and what not to look for will save you a lot of time.
- Always take time to bring documentation up to date. Include the task into development plan; include that time into the plan as part any of change or new feature.
- Keep documentation balanced: some high level diagrams, meaningful comments. Best comments tell that cannot be read from the code itself. Like business reasons or "whys" behind certain chunks of code.

- Include into the plan the effort to keep code structure, folder names, namespaces, object, variable and routine names up to date and reflective of what they actually do. This will go a long way in improving maintainability. Always call a spade "spade". Avoid large chunks of code, structure it by means available within your language of choice, give chunks meaningful names.
- Low coupling and high coherency. Make sure you up to date with techniques of achieving these: design by contract, dependency injection, aspects, design patterns etc.
- From task management point of view you should estimate more time and charge higher rate for non-continuous pieces of work. Do not hesitate to make customer aware that you need extra time to do small non-continuous changes spread over time as opposed to bigger continuous projects and ongoing maintenance since the administration and analysis overhead is greater (you need to manage and analyse each change including impact on the existing system separately). One benefit your customer is going to get is greater life expectancy of the system. The other is accurate documentation that will preserve their option to seek someone else's help should they decide to do so. Both protect customer investment and are strong selling points.
- Use source control if you don't do that already

- Keep a detailed log of everything done for the customer plus any important communication (a simple computer or paper based CMS). Refresh your memory before each assignment.
- Keep a log of issues left open, ideas, suggestions per customer; again refresh your memory before beginning an assignment.
- Plan ahead how the post-implementation support is going to be conducted, discuss with the customer. Make your systems are easy to maintain. Plan for parameterisation, monitoring tools, in-build sanity checks. Sell post-implementation support to customer as part of the initial contract.
- Expand by hiring, even if you need someone just to provide that post-implementation support, do the admin bits.

Recommended reading:

- "Code Complete" by Steve Mcconnell
- Anything on design patterns are included into the list of recommended reading.

edited Oct 2, 2008 at 12:37

answered Oct 2, 2008 at 12:20

Vlad Gudim
23.5k ● 16 ● 71 ● 92

ACK and NACK. ACK, "Code Complete" is really a great book. NACK, not all books about design patters are worth reading. For instance, "Core J2EE Patterns" consists mostly of overhyped blabla. But the GoF book is great. – vog Mar 14, 2009 at 21:24

2

The most important advice I can give having helped grow an old web application into an extremely high available, high demand web application is to encapsulate *everything.* - in particular

1. Use good MVC principles and frameworks to separate your view layer from your business logic and data model.
2. Use a robust persistance layer to not couple your business logic to your data model
3. Plan for statelessness and asynchronous behaviour.

Here is an excellent article on how eBay tackles these problems http://www.infoq.com/articles/ebay-scalability-best-practices

Share Improve this answer Follow

answered Sep 19, 2008 at 23:28

Peter

29.8k ● 22 ● 91 ● 126

1. Use a framework / MVC system. The more organised and centralized your code is the better.

1

2. Try using Memcache. PHP has a built in extension for it, it takes about ten minutes to set up and another twenty to put in your application. You can cache whatever you want to it - I cache all my database records in it - for every application. It does wanders.
3. I would recommend using a source control system such as Subversion if you aren't already.

Share Improve this answer

Follow

answered Dec 24, 2008 at 0:34

Jamie Rumbelow

5,085 ● 2 ● 34 ● 42

-4

You should consider maybe using SharePoint. It's an environment that is already designed to do all you have mentioned, and has many other features you maybe haven't thought about (but maybe you will need in the future :-))

Here's some information from the official site.
There are 2 different SharePoint environments you can use: Windows Sharepoint Services (WSS) or Microsoft Office Sharepoint Server (MOSS). WSS is free and ships with Windows Server 2003, while MOSS isn't free, but has much more features and covers almost all you enterprise's needs.

Share Improve this answer

Follow

edited Sep 19, 2008 at 23:19

answered Sep 19, 2008 at 23:11

Aidenn
559 ● 1 ● 4 ● 12

I think he wants to learn how to improve his code using a framework he already knows. – Michael Brown Sep 19, 2008 at 23:20