Do you design/sketch/draw a development solution first and then develop it? If so how? [closed]

Asked 16 years, 2 months ago Modified 11 years, 6 months ago

Viewed 1k times

7

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 work a lot with decision makers looking to use technology better in their businesses. I have found that **a picture is worth a thousand words** and prototyping a system in a diagram of some sorts always lends a lot to a discussion. I have used Visio, UML (somewhat), Mind Maps, Flow Charts and Mocked Up WinForms to start the vision for these sponsors to ensure that everyone is on the same page. I always seem to be looking for that common process can be used to knit the business vision to the development process so that we all end up at the

same end, "**Something functional that solves a problem**".

I am looking for suggestions or Cliff notes on how to approach the design process such that it works for applications that may only take a week to develop, but can also be used to encompass larger projects as well.

I know that this delves into the area of UML, but I have found that I have a hard time finding a guide to appropriately use the various diagram types, let alone help business users understand the diagrams and relate to them.

What do you use to capture a vision of a system/application to then present to the sponsors of a project? (all before you write a single line of code)...

uml **system**

Share

Improve this question

Follow

edited Oct 1, 2008 at 2:16

Blorgbeard

103k ● 50 ● 235 ● 276

asked Oct 1, 2008 at 2:06

KodeTitan

913 ● 1 ● 9 ● 23

10 Answers

Sorted by: Highest score (default)

Paper or whiteboard!

For the lone deveoper, I'd recommend paper. At least at first, eventually you may want to formalize it with UML, but I don't think its necessary.

For a group of developers that work together (physically), I'd recommend a whiteboard. That way its visible for everyone and everyone can improve and contribute. Again, you may want to formalize at this point, but I still don't think its neces

When I first started doing OOP (or designing an algorithm), I'd do it all in my head while coding. But after doing some reasonable complex projects I definitely see the benefit in drawing out the interactions between different objects in a system.

I do projects by myself, so I use lots of index cards for designing classes and paper for their interactions. The point is it needs to be easy to change. I've played with Dia, a diagram editor, for doing UML, but making changes is too difficult. I want to be able to be able to make quick changes, so I can figure out what works best.

Its worth mentioning that TDD and doing "spike"[1] projects can help when designing a system, too.

[1] From Extreme Programming Adventures in C#, page 8:

> "Spike" is an Extreme Programming term meaning "experiement." We use the word because we think of a spike as a quick, almost brute-force experiment aimed at learning just one thing. Think of drving a big nail through a board.

Share Improve this answer Follow

edited Oct 1, 2008 at 2:29

answered Oct 1, 2008 at 2:18

Mark A. Nicolosi

85.4k ● 11 ● 46 ● 46

2

For small or very bounded tasks, I think developers almost universally agree that any sort of diagram is an unnecessary step.

However when dealing with a larger, more complicated system, especially when two or more processes have to interact or complex business logic is needed, Process Activity Diagrams can be extremely useful. We use fairly pure agile methods in development and find these are almost the only type of diagrams we use. It is amazing how much you can optimize a high level design just by having all the big pieces in front of you an connecting them with flow lines. I can't stress enough how important it is to tailor the diagram to your problem, not the other way around, so while the link gives a good starting point,

simply add what makes sense to represent your system/problem.

As for storage, whiteboard can be great for brainstorming and when the idea is still being refined, but I'd argue that electronic and a wiki is better once the idea is taking a fairly final shape (OmniGraffle is the king of diagramming if you are lucky enough to be able to use a Mac at work:) . Having an area where you dump all these diagrams can be extremely helpful for someone new to get a quick grasp on an overall piece of the system without having to dig through code. Also, because activity diagrams represent larger logic blocks, there is not the issue of always having to keep them up to date. When you make a large change to a system, then yes, but hopefully likely using the existing diagram to plan the change anyway.

Share Improve this answer

Follow

answered Oct 1, 2008 at 2:33

Peter

29.8k ● 22 ● 91 ● 126

1

Read up on Kruchten's 4+1 Views.

Here's a way you can proceed.

1. Collect use cases into a use case diagram. This will show actors and use cases. The diagram doesn't start with a lot of details.
2. Prioritize the use cases to focus on high value use cases.

3. Write narratives. If you want, you can draw Activity diagrams.

The above is completely non-technical. UML imposes a few rules on the shapes to be used, but other than that, you can depict things in end-user terminology.

1. You can do noun analysis, drawing Class diagrams to illustrate the entities and relationships. At first, these will be in user terminology. No geeky technical content.
2. You can expand the activity diagrams or add sequence diagrams to show the processing model. This will start with end-user, non-technical depictions of processing.
3. You can iterate through the class and activity diagrams to move from analysis to design. At some point, you will have moved out of analysis and into engineering mode. Users may not want to see all of these pictures.
4. You can draw component diagrams for the development view and deployment diagrams for the physical view. These will also iterate as your conception of the system expands and refines.

answered Oct 1, 2008 at 2:23

S.Lott

391k ● 82 ● 517 ● 788

1

When designing an application (I mainly create web applications, but this does apply to others), I typically create user stories to determine exactly what the end user really needs. These form the typical "business requirements".

After the user stories are nailed down, I create flow charts to lay out the paths that people will take when using the app.

After that step (which sometimes gets an approval process) I create interface sketches (pen/pencil & graph paper), and begin the layout of the databases. This, and the next step are usually the most time consuming process.

The next step is taking the sketches and turn them into cleaned up wireframes. I use OmniGraffle for this step -- it's light years ahead of Visio.

After this, you may want to do typical UML diagrams, or other object layouts / functionality organization, but the business people aren't going to care so much about that kind of detail :)

Share Improve this answer

Follow

answered Oct 1, 2008 at 2:25

pixel

5,298 ● 8 ● 37 ● 33

1

When I'm putting together a design, I'm concerned with conveying the ideas cleanly and easily to the audience. That audience is made up of (typically) different folks with different backgrounds. What I don't want to do is get into "teaching mode" for a particular design model. If I have to spend considerable time telling my customer what the arrow with the solid head means and how it is different from the one that is hollow or what a square means versus a circle, I'm not making progress - at least not the progress I want to.

If it is reasonably informal, I'll sketch it out on a whiteboard or on some paper - block and simple arrows at most. The point of the rough design at this point is to make sure we're on the same page. It will vary by customer though.

If it is more formal, I might pull out a UML tool and put together some diagrams, but mostly my customers don't write software and are probably only marginally interesting in the innards. We keep it at the "bubble and line" level and might put together some bulleted lists where clarification is needed. My customer don't want to see class diagrams or anything like that, typically.

If we need to show some GUI interaction, I'll throw together some simple window prototypes in Visual Studio - it is quick and easy. I've found that the customer can relate to that pretty easily.

In a nutshell, I produce simple drawings (in some format) that can communicate the design to the interested parties and stake holders. I make sure I know what I want it to do and more importantly - what THEY NEED it to do, and talk to that. It typically doesn't get into the weeds because folks get lost there and I don't find it time well spent to diagram everything to the nth degree. Ultimately, if the customer and I (and all other parties concerned) are on the same page after talking through the design, I'm a happy guy.

Share Improve this answer

Follow

answered Oct 1, 2008 at 2:33

itsmatt

31.4k ● 11 ● 102 ● 165

1

I'm an Agile guy, so I tend to not put a lot of time into diagramming. There are certainly times when sketching something on a white board or a napkin will help ensure that you understand a particular problem or requirement, but nothing really beats getting working software in front of a customer so they can see how it works. If you are in a situation where your customers would accept frequent iterations and demos over up front design, I say go for it. It's even better if they are okay to get early feedback in the form of passing unit or integration tests (something like Fit works well here).

I generally dislike prototypes, because far too often the prototype becomes the final product. I had the misfortune of working on a project which was basically extending a

commercial offering which turned out to be a "proof of concept" that got packaged and sold. The project was canceled after over 1000 defects were logged against the core application (not counting any enhancements or customizations we were currently working on).

I've tried using UML, and found that unless the person looking at the diagrams understands UML, they are of little help. Screen mock-ups are generally not a bad idea, but they only show the side of the application which directly effects the user, so you don't get much mileage for anything that isn't presentation. Oddly enough tools like the Workflow designer in Visual Studio produce pretty clear diagrams that are easy for non-developers to understand, so it makes a good tool for generating core application workflow, if your application is complex enough to benefit from it.

Still, of all the approaches I've used over the years, nothing beats a user getting their hands on something to let you know how well you understand the problem.

Share Improve this answer

Follow

answered Oct 1, 2008 at 2:43

ckramer

9,433 ● 1 ● 26 ● 38

1

I suggest reading Joel's articles on "Painless Functional Specifications". Part 1 is titled "Why Bother?".

We use Mockup Screens at work ("Quick and Easy Screen Prototypes"). It's easy to alter screens and the

scetches make clear that this is only a design.

The mockups are then included in a Word document containing the spec.

Share Improve this answer

Follow

answered Oct 1, 2008 at 5:07

Mackaaij

470 ● 2 ● 9

Btw since MockupScreens 4.70 you can export your mockups directly to Word – IgorJ Jun 18, 2013 at 23:05

1

From *Conceptual Blockbusting: A Guide To Better Ideas* by James L. Adams:

> Intellectual blocks result in an inefficient choice of mental tactics or a shortage of intellectual ammunition. . . . 1. Solving the problem using an incorrect language (verbal, mathematical, visual) -- as in trying to solve a problem mathematically when it can more easily be accomplished visually

(pg. 71, 3rd Edition)

Needless to say, if you choose to use diagrams to capture ideas that may be better captured with mathematics, it's equally bad. The trick, of course, is to find the right language to express both the problem and the solution too. And, of course, it may be appropriate to use more

than one language to express both the problem and the solution.

My point is that you're assuming that diagrams are the best way to go. I'm not sure that I would be willing to make that assumption. You may get a better answer (and the customer may be happier with the result) via some other method of framing requirements and proposed designs.

By the way, Conceptual Blockbusting is highly recommended reading.

Share Improve this answer

Follow

edited Aug 21, 2011 at 15:19

Marek Grzenkowicz

17.3k ● 11 ● 84 ● 117

answered Oct 1, 2008 at 2:20

Onorio Catenacci

15.3k ● 16 ● 84 ● 134

0

The UML advice works well if you're working on a large & risk-averse project with a lot of stakeholders, and with lots of contributors. Even on those projects, it really helps to develop a prototype to show to the decision makers. Usually walking them through the UI and a typical user story is quite sufficient. That said, you must beware that focus upon the UI for decision makers will tend to make them neglect some significant backend issues such as validations, business rules and data integrity. They will

tend to write these issues off as "technical" issues rather than business decisions.

If, on the other hand, you're working on an Agile project where it's possible to make code changes quickly (and rollback mistakes quickly), you may be able to make an evolutionary prototype with all the works. Your application's architecture should be supple and flexible enough to support quick change (e.g. naked objects design pattern or Rails-style MVC). It helps to have a development culture that encourages experimentation, and acknowledges that BDUF is no predictor of working successful software.

Share Improve this answer

Follow

answered Oct 1, 2008 at 2:41

Alan

7,064 ● 6 ● 32 ● 38

0

4+1 views are good only for technical people. And only if they are interested enough. Remember those last dozen times you struggled to discuss use-case diagrams with the customer?

The only thing I found that works with everybody is in fact showing them screens of your application. You said yourself: a picture is worth a thousand words.

Curiously, there are two approaches that worked for me:

1. Present to users a complete user manual (before even development is started), OR

2. Use mockups that doesn't look at all like finished app: Discuss main screens of you app first. When satisfied, proceed discussing mockups but one scenario at a time.

For option (1) you can use whatever you want, it doesn't really matter.

For option (2) it's completely fine to start with pen and paper. But soon you are better off using a specialized mockup tool (as soon as you need to edit, maintain or organize your mockups)

I ended up writing my own mockup tool back in 2005, it became pretty popular: MockupScreens

And here is the most complete list of mockup tools I know of. Many of those are free: http://c2.com/cgi/wiki?GuiPrototypingTools

answered Jun 18, 2013 at 23:20

IgorJ

428 ● 4 ● 8