Wed 2 May 2007

How to Write an Effective Design Document

Posted by Scott Hackett under Programming
[56] Comments

Day by day, programmers are able to get more done in less time. With today's high level languages, development environments, tools and the "rapid application development" mindset, both programmers and managers have become accustomed to extremely fast development cycles. Programmers are now more inclined to jump directly into development, fearing that every hour they are not writing code will result in an hour of overtime the weekend before the deadline.

The process of designing before coding is becoming outdated. Documenting designs is becoming even rarer. Many developers have never written a design document and cringe at the idea of doing so. Many who are required to, typically generate a lot of interaction diagrams and class diagrams, which don't often express the developer's thought process during the design phase. This article will discuss how to do write an effective design document concisely with no special tools, and without needing to know UML. It will also discuss why a well written design document is one of the most valuable tools a developer can have when entering a new project.

Why write a design document?

A design document is a way for you to communicate to others what your design decisions are and why your decisions are good decisions. Don't worry if your design is not UML compliant and don't worry if you didn't use a special modeling tool to create it. The biggest factor that determines if your design document is good is whether or not it clearly explains your intentions.

This presents a problem, however. In order to convey design decisions, you have to consider the audience that you are writing for. A peer developer will understand why a well-crafted class abstraction is a good design, however your manager will probably not. Because your peer developers and your manager have different concepts of what makes a design good, there is a need for two design documents; one for peer developers and one for managers. Each document serves a different and equally valuable purpose as you begin your project development.

If this seems like too much work, it's not. This article will show you how to do this through documentation reuse.

What makes a good design?

A design will typically be considered good if it fulfills the requirements in a meaningful way. If any aspect of the design cannot be justified, then it is probably worth reevaluating. Many programmers try to incorporate design patterns into their work, and they often add unnecessary complexity. You should be able to list at least one compelling reason, related to the requirements, for why a design decision was made. That reason must then be documented. If you can't come up with a clear reason for a design decision, then it is probably not adding value.

Diagrams are a great tool for visualizing your design, but they cannot convey the motivation behind your design decisions. That is why it is so important to let diagrams supplement your design document, not be your design document.

In addition, it is also extremely important to document any benefits that result from a design decision. By doing so, others who read your document will understand what value they can gain. Likewise, any associated risks must also be documented. More often than not, other programmers have faced the same risks and may have helpful pointers or solutions that you may not have thought about. By listing these items, you also get others to think about what the potential risks could be as well. Teammates will often be able to see potential pitfalls that you didn't see when you created your design. It is much easier to rearrange some boxes in a diagram than it is to

rewrite hundreds of lines of code when an assumption fails or when you hit an unforeseen snare during coding. A good design document minimizes unexpected complications by addressing them before the code is written.

Finally, a document will provide you, your manager and your team with a common vocabulary for talking about the project. A design document can be a powerful tool for a manager because it gives them a view into the project that they don't normally have the technical expertise to see. By listing the benefits you give your manager tangible items that describe why your design is sound. By documenting the risks of your design before development, you pass the responsibility of that risk to your manager, which is where it belongs.

Lastly, the design document is a written contract between you, your manager and your team. When you document your assumptions, decisions, risks, etc, you give others a chance to say, "Yes, this is exactly what I expect." Once your document passes that stage, it becomes a baseline for limiting changes in project scope. Obviously, requirements are going to change sometimes, but with a baseline document you have the power to say that no change in scope is due to a misunderstanding of the requirements.

Writing for a Peer Developer

The goal of a peer developer design document is to make sure that your ideas are valid and that your approach works with what others are doing. When developers don't communicate their plans, disaster is sure to strike when modules or classes begin to interact. The following items describe a general guideline for writing this type of document:

Section 1 – State the purpose of your project/sub-system: In this section, write a few paragraphs that describe what the project or sub-system does. What is the problem it is trying to solve? Why does it need to exist? Who will use it? By answering these questions, you establish the scope of your design. If you find it hard to write a few paragraphs in this section, then you probably don't understand the domain as much as you should. If you can't fit your description within a few paragraphs, then perhaps the scope is too large. Use this section as a tool to verify that the scope of your design is reasonable.

Section 2 – Define the high level entities in your design: High level entities are objects, or groups of objects, that constitute major constructs of your design. Good examples of entities are a data access layer, a controller object, a set of business objects, etc… Figure 1 shows an example of a .

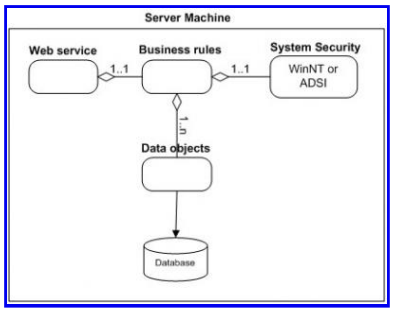

Figure 1 (click to see full size)

In this section, explain in a few sentences what each entity does. The descriptions don't have to be verbose, just enough to explain what each block's purpose is. Be sure to describe your reasoning for defining the entities in your diagram and what their roles are.

Section 3 – For each entity, define the low level design: This section is where your objects and object relationships are defined. For each object (or set of objects) define the following:

Usage

Describe in a paragraph how the object is used and what function it serves. If an object will interface with an external object or system, it is a good idea to show the interface for the object. Most importantly, you must again describe your thought process for defining the object as you did. List the benefits and risks. If an object provides an encapsulation, describe in a sentence why the encapsulation adds value. Use your descriptions to give meaning to the diagrams. They don't have to be verbose, just enough to get the point across.

Configuration

If your object needs any special configuration or initialization, this is a good place to describe it. If not, this section can be left out.

Model

Figure 2 shows an example of a to supplement the System Security entity from figure 1. It is not perfect UML, but has some aspects of UML. Most importantly, it describes the design.

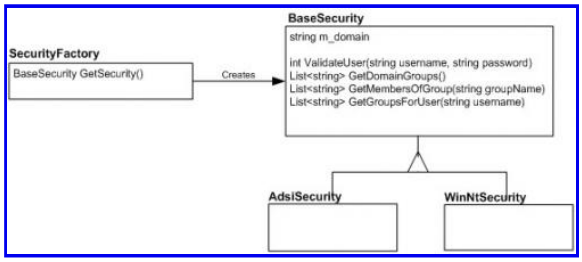

Figure 2 (click to see full size)

Don't worry about perfection in your models, but be sure to describe exactly what is going on in the diagram. Here, two concrete security objects derive from a base security object, and a security factory will create one or the other for a client depending on the security model of the system.

Interaction

This is also a good section for interaction diagrams. An interaction diagram shows how a set of objects or entities communicate with each other to perform a complex task. Figure 3 shows an example of an to show how a user might log in. It uses objects from the various entities shown in figure 1.

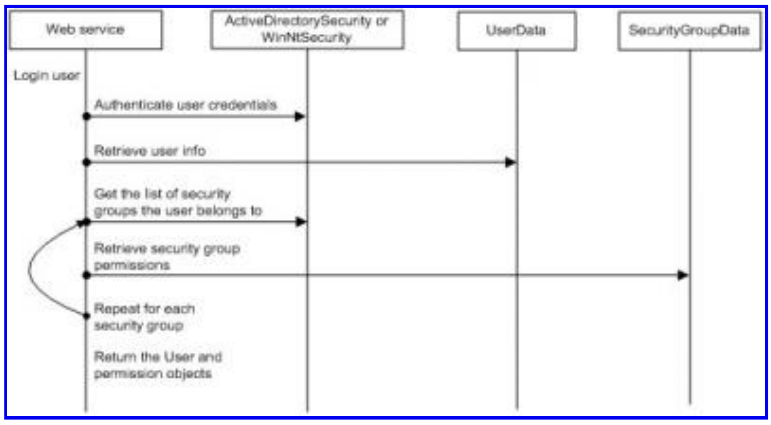

Figure 3 (click to see full size)

Again, this diagram is not perfect UML, but it explains the communication sequence to accomplish a complex task. Interaction diagrams are most useful when you want to diagram how an object in your system will communicate with an object in another subsystem. This type of diagram will let the other developer verify that the interaction is correct.

Section 4 – Benefits, assumptions, risks/issues: In this section, make a list of 5-6 top benefits of the design, a list of **ALL** known risks/issues and a list of ALL assumptions. Some of this may simply be rehashing what you wrote in a previous section of the document. What's important is getting all of these items into one section so that the reader doesn't have to read the whole document to understand what the benefits, risks and assumptions are.

Never remove anything from this section! As risks become non-risks, document that they are now non-risks and why they became non-risks. Never erase them from the document. The same holds true for assumptions. You should be able to look at this section and know instantly what the current risks are to your design.

Writing for a Manager

The goal of a design document for your manager is to make sure that your manager understands what the main entities of the system are, what the benefits are and, most importantly, what the risks are. The document is your chance to show that you understand the requirements and that you have come up with a plan to meet those requirements.

If you have written the peer developer document well, then writing the manager's document is simple, because it is just made up of sections 1, 2 and 4. By dividing the peer developer document up as described previously, the parts which are typically not meaningful to a manager have been contained in a single section of the document which may be removed.

Conclusion

The hardest part of writing a design document has nothing to do with the writing. The difficult part is working through a logical design before you get to coding. Once you have a vision of how the objects and entities are arranged, writing the details is easy. In addition, it should not require anything more than a word processor and a simple shape painting program. The positive difference that spending a week on this task can make is unbelievably rewarding in the end. As the adage goes, "If you fail to plan, then you plan to fail."

« What is a Power Programmer? | Thank you TI-99/4A »

---

56 Responses to " How to Write an Effective Design Document "

Comments:

1. *Ben* says:
   May 10th, 2007 at 5:42 pm

   Thanks for the very interessting article. Your blog has become a everyday-must-read to me.

   What about a nifty "diggit"/"reddit" Button?

   Regards Ben

2. *Scott H* says:
   May 11th, 2007 at 7:50 am

   Thanks Ben! I really enjoy design work and always try to focus my design docs around expressing the decisions I made and why I made them. That's an area that a lot of popular design methodologies tend to forget about. Glad you liked my post, I'm really enjoying having the blog here.

3. *Jason* says:
   May 14th, 2007 at 10:03 am

   Hi Ben,

   Glad you are enjoying the blog. As requested, we now have a nifty "diggit"/"reddit" Button.

   Thanks for the suggestion,
   Jason

4. *Kevin Lynx* says:
   May 25th, 2007 at 12:40 pm

   good articles for a beginner .