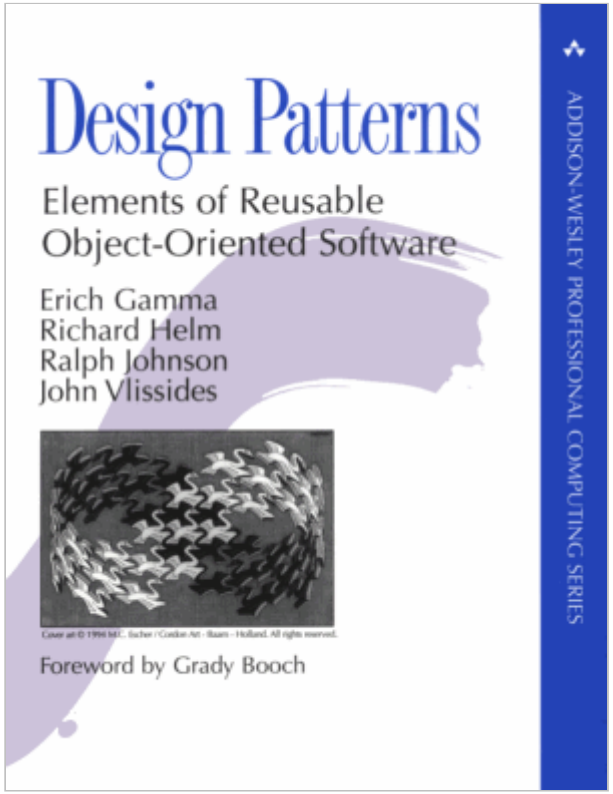
CODING HORROR

programming and human factors

02 Jul 2007

Rethinking Design Patterns

Many developers consider the book Design Patterns a classic.



RESOURCES

So what's a design pattern?

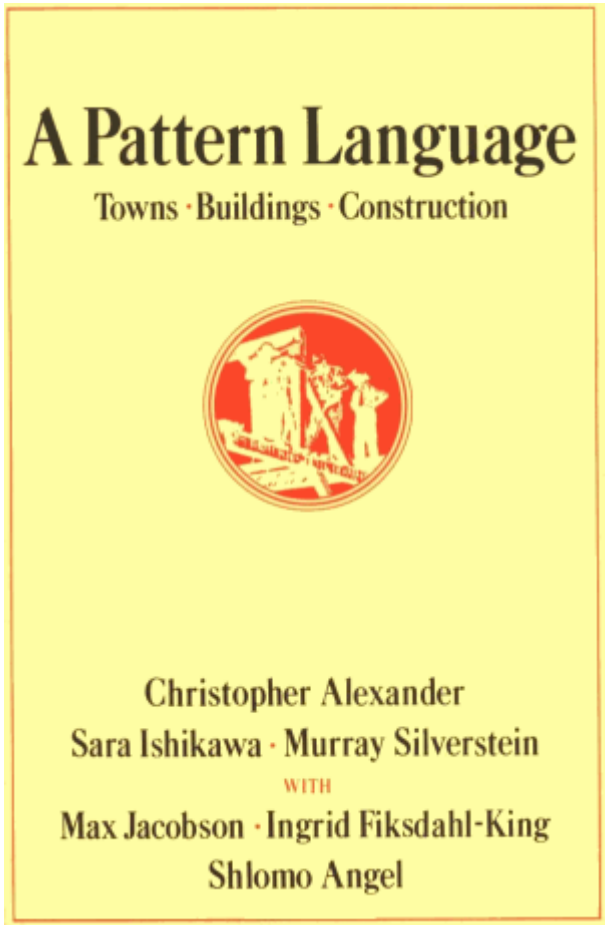
> A design pattern systematically names, motivates, and explains **a general design that addresses a recurring design problem** in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a **general arrangement of objects and classes** that solve the problem. The solution is customized and implemented to solve the problem in a particular context.

It's certainly worthwhile for every programmer to read Design Patterns at least once, if only to learn the shared vocabulary of common patterns. But I have two specific issues with the book:

1. Design patterns are a form of complexity. As with all complexity, I'd rather see developers focus on simpler solutions *before* going straight to a complex recipe of design patterns.
2. If you find yourself frequently writing a bunch of boilerplate design pattern code to deal with a "recurring design problem", that's *not* good engineering--it's a sign that your language is fundamentally broken.

In his presentation "Design Patterns" Aren't, Mark Dominus says **the "Design Patterns" solution is to turn the programmer into a fancy macro processor**. I don't want to put words in Mark's mouth, but I think he agrees with at least one of my criticisms.

But Dominus also digs deeper into the source material than most. He cites Christopher Alexander's book A Pattern Language, which was the inspiration for Design Patterns.



Dominus summarizes the book thusly:

> Suppose you want to design a college campus. You must delegate some of the design to the students and professors, otherwise the Physics building

> won't work well for the physics people. No architect knows enough about about what physics people need to do it all themselves. But you can't delegate the design of *every* room to its occupants, because then you'll get a giant pile of rubble.
>
> How can you distribute responsibility for design through all levels of a large hierarchy, while still maintaining consistency and harmony of overall design? This is the architectural design problem Alexander is trying to solve, but it's also a fundamental problem of computer systems development.

That's the key insight that drives both books. Unfortunately, Dominus believes that the Gang-of-Four version obstructs Alexander's message, replacing actual thought and insight with a plodding, mindless, cut-and-paste code generation template mentality:

> The point of the talk is this: The "design patterns" you get from the Gang-of-Four book are not the same as the idea of "design patterns" that are put forward in Alexander's books. People say they are, but they aren't the same thing. Alexander's idea is a really good one, one that programmers might find useful. But very few programmers are going to find out about it, because they think they already know what it is. But they actually know this other idea which happens to go by the same name.
>
> **So (once again) we need to take a fresh look at Christopher Alexander**. Forget what I said about the damn iterator pattern, already.

I know it's not technically a software development book, but consider this advice from Don Park:

> If Eclipse is a boomtown which countless developers and companies continue to pour into, it now looks like LA, tiny downtown surrounded by endless expanse of suburban neighborhoods indistinguishable from each other other than by their names. Although one of the key pioneers behind Eclipse is Eric Gamma, one of the four authors of the infamous Design Patterns book, I feel that not enough attention is being paid to the original concepts that inspired the book, concepts captured in books by Christopher Alexander.

You could read Design Patterns like any number of other software developers before you. But I humbly suggest that you should go deeper and read A Pattern Language, too, because ideas are more important than code.

NEXT
The Technology Backlash

PREVIOUS
Avoiding Walled Gardens on the Internet

Written by Jeff Atwood

Indoor enthusiast. Co-founder of Stack Overflow and Discourse. Disclaimer: I have no idea what I'm talking about. Find me here: http://twitter.com/codinghorror

Continue Discussion

83 replies

Charles Jul '07

Thank you Jeff. Finally someone mentions that the Emperor has no clothes! I have always been floored by the devotion of my otherwise sensible brethren to strict adherence to the GoF commandments. While they force their code into this pattern and that pattern they typically miss the fact that they are more often than not INCREASING complexity rather than following Fowler's first rule of KISS. Other than that they seem like genuinely intelligent people. Go figure.

Reed Jul '07

I read through Design Patterns once, just to get some ideas for future use. Maybe I'll refer back to it if someone references it in documentation or conversation, or if I vaguely recall reading something there, but it shouldn't be used as a hard manual for how to do everything.

Eoin_Prout Jul '07

Design Patterns are now part of software development Dogma,
any questioning of them will be met with cries of heresy.

Well done Jeff!

Parveen Jul '07

I particularly enjoyed /The Timeless Way of Building/ by Alexander. It attempts to describe the concept the QWAN.
http://en.wikipedia.org/wiki/Quality_without_a_name

The concept of QWAN is much more important than a pattern language. A lot of the code that I read that uses design patterns severely lacks QWAN. Code with QWAN can be looked at and you can say, "hey I get how that works" and not require some name for the pattern.

Marcos_Meli Jul '07

Excellent post =)

I think about this, in the same way I do in some other points

“The OVERUSE is worst thank NO USE it at all”

I hate the overuse of XML, DesignPatters (useless factories, and a lot of things like that), XML configuration, Ajax, and the list go on

Is important to understand and use all these things, but more important is how and where to use them.

The clear example to me is Java vs .NET both use patters but I think that the .NET approach is a lot clever.

In Jave the explicityness of patters (what a word :P) is the key, in .NET you can do simple things with little code or one line, with Jave you need to instanciate a lot of objects, or know about much more clases.

I love “Framework Design Guidelines” from the CLR core team, is an amazing book where patterns are important but be rationale is more 😛

Best Regards

rmf Jul '07

I agree wholeheartedly. Certainly there are some patterns that are incredibly useful in software engineering, but I find many people focus too much on the usage of a pattern instead of its usefulness. It’s like patterns are many Sr. Developer/Architect’s way of overcompensating for weak OO design skills. Roughly 80% of all pattern usage I see is what I call ‘contrived complexity’. “Oh this uses the Visitor Pattern” the architect confidently chirps. “Yeah, but why? Can’t you just use standard, boring old, solid OO design to do that?”.

What’s worse is that this ‘Pattern Overcompensation’ leaks into the Jr. Developers habits too. I can’t tell you how many times I’ve had a developer on my current project say “we should use a factory pattern here”, when I’m certain they have no idea how empty that statement is.

Keebler Jul '07

Who are there architects that I always hear about that blindly apply the Gospel of the Design Patterns instead of thinking? I’ve not seen them, and the GoF book doesn’t seem to advocate

it as far as I can tell.

I've read Design Patterns more than once, and the only way I've ever used them or seen them used is as a verbal/texual/conceptual short-hand for a code technique. "We'll handle the specific handling there with a stragegy class" or "Well, lets use a factory to keep the object-generation code from mucking things up" or "sort of a flyweight pattern thing to keep these from eating up memory"… just gives a label to things we'll be doing anyway.

I think of it like music theory. You don't write a song by cobbling together theoretical rules. However, you can look at a song and think, "Ah, it's a secondary dominant thing here… I could jazz it a little by substituting…" Also, quite useful when describing to other musicians in a way they'll understand. You write what you want to write, and use the pre-named patterns for conceptual abstraction, to draw parallels with existing stuff, and to communicate to others.

I'm also quite familiar with the "design patterns are usually a flaw in the language/framework" argument. A valid point. I'd also like to note that I also have to keep wearing shoes because nobody has gotten around to carpeting the sidewalks. I don't like C++ or Java; they force me to bend backwards to do things that are simple in other languages, but I rarely have the luxury of choosing whatever tools I like on a project I'm getting paid for, so the non-necessity of certain patterns and idioms may be in more modern languages is not that relevant most of the time.

Grant Jul '07

Have you been talking to Alex Miller by off chance?

http://tech.puredanger.com

codinghorror Jul '07

Who are these architects that I always hear about that blindly apply the Gospel of the Design Patterns instead of thinking? I've not seen them, and the GoF book doesn't seem to advocate it as far as I can tell.

Indeed. Ever read "The Bible", or "The Quran"? Strangely, those books don't advocate a lot of the things you'll see people doing in the name of them, either.

someone Jul '07

Someone Jul '07

You aren't actually supposed to program with design pattens in mind. You are supposed to program with the knowledge design patterns gives you and not think in terms of I'm going to use strategy here, or visitor there, that leads to muck. Designing around a particular pattern is an anti-pattern. Design patterns ae discovered from existing code not written down from scratch. It is the knowledge the pattern holds that is important not the pattern itself.

DaveS Jul '07

The good thing about design patterns is that they provide a shared vocabulary for some concepts.
The bad thing about a shared vocabulary (not necessarily design patterns, mind you) is that you're well on your way to providing more evidence for the Sapir-Whorf hypothesis (http://en.wikipedia.org/wiki/Sapir-Whorf_hypothesis) by knowing them.
There's a right way and a wrong way to use patterns - if you start to see patterns emerging as you're building your system, great. If, during the Big Design Up Front, you hand down a mandate from the heavens that "Object Bar will implement the Singleton pattern", you're doing it wrong. Put down that Golden Hammer, kind architect.
Form should follow function rather than attempt to inform it.

John_Radke Jul '07

Jeff - surely this has been suggested to you before, but I strongly recommend you check out Head First Design Patterns. It's more of a teaching book, whereas the GoF book should really be considered more of a reference. HFDP teaches good OO design principles, and then sort of reinforces them by introducing design patterns as common solutions which follow those principles.

I think the real problem with GoF is how it's applied. It's not meant to be an instruction manual. You should already be familiar with OO design and the common problems associated with it before you even start thinking about patterns.

Another problem with GoF is how obtuse it is. HF gives you tangible, if impractical, examples (QuackBehavior on a Mallard which inherits from IDuck, or somesuch) whereas GoF seems to talk about accounts and transactions and other boring stuff that only makes sense to programmers who've written accounting software.

codinghorror Jul '07

I strongly recommend you check out Head First Design Patterns.

It's linked in the above article, too, but…

http://www.codinghorror.com/blog/archives/000380.html

Jim Jul '07

As Keebler said, some (IMHO a large measure) of the value of the GoF book, is that it facilitates communication. Two other examples are Martin Fowler's PoEAA and Refactoring, both of which don't have much deep metaphysical insight, but do provide a common vocabulary for the things we do. This not only helps communication, but can be valuable to clear one's own thinking.

Oh wait –

It's certainly worthwhile for every programmer to read Design Patterns at least once, if only to learn the shared vocabulary of common patterns.

– we seem to agree. Never mind

Martin Jul '07

I subscribe to the idea that for every design pattern there is a language in which you do not need to use that design pattern, because the language allows you to express your intention directly.

However more often than not the choice of programming language is driven by other factors. Sometimes you need to use Java because it is your companies standard. Sometimes you need to use C or C++ because no other language gives you the same amount of control of the machine internals.

In those cases you want to use design patterns to implement those recurring problems that are not directly expressable in this language.

MartinS Jul '07

Well, I guess this kind of criticism can be applied to almost any book. If you just blindly follow formulas, you are a chimp developer, and we all know there are too many out there.

As you said answering another comment, any book that becomes canonical (like the Bible, the Koran, Fowler's EAAP, or even Code Complete) can lead to lots of people blindly following what "looks" like a recipe, and not getting the real message.

You can surely find building architects applying Alexander's patterns in a literal way, too.

In the end, I agree with you that we -the people presenting and training and writing about stuff- should try to clearly state that Design Patterns are not recipes, but conceptual models to help talk about design. I did many trainings about Design Patterns and I'm a witness that they actually work, but you always find some people thinking about them as templates.

Also this is something where Microsoft didn't help. While I'm generally fond of Patterns Practices, I feel that some of their guidance is not clear enough and many times it drives people to the patterns=templates…

See you.

NickQ Jul '07

Jeff, I just took a class that delved through most of the design patterns in this book and then attempted to put them into practice in different projects. I don't think we looked into any of the coding examples (honestly, Smalltalk and outdated C++ were not helpful at all), but like you said, it's the ideas that count.

For students who are learning about the software industry and how to design software systems, this book is an invaluable tool about how some problems could be solved. It's by no means a "Golden Hammer"…but at the very least it encourages discussion about different methodologies in programming.

JesseM Jul '07

I read Notes On The Synthesis Of Form last year because it was shorter and cheaper than A Pattern Language… I would say it's probably as valuable for time invested, even though some of the ideas are questionable (such as every design constraint simply being a binary value).

Petey Jul '07

Once, a long time ago, I was discussing a movie with my brother and explaining why I thought the symbolism was over the top and unnecessary to put through the screenwriter's intentions. My brother responded with "You ruin movies. Can't you just watch a movie without analyzing it?"

The truth is, no, I can no more unlearn to recognize symbols, both conscious and subconscious, than I can lose the ability to see the color red. I can still find a movie funny, sad, sexy, exciting, or terrible based on its own mertis, but I cannot view it superficially.

A Pattern, like a program, is a conglomeration of symbols. Once you learn to recognize them, you get a deeper understanding of the art of writing code. This doesnt mean that you cant appreciate simplicity, or that programming that doesnt comform to a Design pattern doesnt follow a pattern and hasnt been designed. But you cant help but remember what a Strategy pattern is, for instance, and when a problem presents itself that can be solved easily by a strategy pattern, not think to use it.

Petskull Jul '07

Hey, Jeff-

I've been reading your blog for a while now and I love how all of your articles link to each other to present a complete picture of your point of view.

I think, however, it may be helpful to add (with a question mark; underscores for spaces) the linked article's title after the URL; as in the following:

http://www.codinghorror.com/blog/archives/000380.html?Head_First_Design_Patterns

This may help with some clarity ("Ah, yes! I remember that one!") that arises when you link random phrases to your previous articles.

Regards,
Petskull

Andrew Jul '07

"If you just blindly follow formulas, you are a chimp developer, and we all know there are too many out there."

Actually I want to blindly follow formulas. Software development is so complex and I'm so stupid that if there are techniques that I can apply without thinking then I can use the thinking that I save there on other things that I really need to think deeply about.

This applies to more than design patterns but to algorithms and logic. Why kill your brain trying to reinvent sorting algorithms when someone has already solved your problem before? Why struggle with developing your own coding standards when you can use one of the many standards out there? Why create your own image processing library when you can reuse somebody else's?

Programming is hard. If there's something out there that helps me to cheat and get things done faster and better than would have happened if I reinvented the wheel, then damnit I want to use it. If that makes me mindless or makes people think that I'm stupid then I'm happy with that.

DavidD Jul '07

I definitely agree that developers need absorb ideas (i. e., read books) from other disciplines. But, at 1,216 pages, that's a waste of time, unless you plan to switch careers.

WarpedJ Jul '07

Design patterns sure are a classic. We should write a song about them…

Yeah, they say two thousand patterns patterns party over
Oops out of time
So tonight I'm gonna program like it's 1999
Yeah

As programmers we need to remember that design patterns are solutions to problems, and not implementations to problems.

Shawn_Oster Jul '07

The issue is less with the GoF's "Design Patterns" and more with the actual people reading it. Some people read it looking for a quick fix, hence creating a copy-and-paste template generating programmer while others read it, digest it, think about it and come out with some

good bits.

Personally when I read it so long ago what it really did was give me different ways to view how objects interact, different ways to couple and decouple frameworks, different contexts for more abstract classes, etc. Since most of the examples are in SmallTalk if I remember rightly there really was no chance for me to copy and paste, I had to think about application and approach and while I've never used many patterns I always thought about how to structure things so they were simple and well-organized.

Everyone learns differently and some people you have to "force" to actually think because otherwise they'll just mimic. There are others that can learn from looking at well written source code, turning the implemented ideas into lessons. Personally I'm in the latter category which is why I got a lot out of "Design Patterns" and when I did read "A Pattern Language" afterwards I felt it dragged on, labouring over concepts that I had already inferred from "Design Patterns". Then again I never had "Design Patterns" next to my computer, trying to use it like some code template handbook, instead I'd read it before bed, on road trips or just on my deck with a nice cigar.

I don't think either is better, just different, which is why it's important that there's many different views and approaches to teaching the same basic thing.

Martin_Cron Jul '07

I don't know if it's that the Gang of Four didn't "get" Christopher Alexander, or if it's that almost nobody "gets" the Gang of Four. I actually think the second scenario is more likely, especially as the book is so dense that it's nearly unreadable. For example, the patterns are in alphabetical order instead of increasing order of usefulness, etc. Other authors have done a much better job of making the (good) core concepts more accessible.

The design patterns gurus I've worked with have always said that the pattern is in the encapsulation of the forces in a problem, and aren't boilerplate solutions. The diagrams in the book aren't
the patterns", they are just one example of a way to implement the patterns. If your implementation is different, that's great.

If a pattern-oriented design is more complex, you're probably misusing the pattern. You have to use a more sophisticated measure of complexity than just "number of classes", though.

It's a shame that the whole design patterns mindset is so easily misunderstood and incorrectly

applied. I've found knowledge of patterns has really improved the quality of my code and my ability to communicate design to other developers.

codinghorror Jul '07

I definitely agree that developers need absorb ideas (i. e., read books) from other disciplines. But, at 1,216 pages, that's a waste of time, unless you plan to switch careers.

Use the Amazon "search inside" function to browse A Pattern Language. I don't think it's the type of book you read cover-to-cover.

But you got my intent: absorb the deeper ideas and think for yourself about how these might (or might not) apply to software development.

JD111 Jul '07

I'm not sure what's worse: A junior developer who slavishly adheres to design patterns or, as was my recent experience, a junior developer who *shares* the sentiments of your blog post and has concluded that design patterns are for ninnies, so why bother with them?

To me, design patterns represent frequently elegant solutions to common problems. I'd much prefer a junior developer working under me to slavishly grasp and adhere to a design pattern than to strike out on his own and decide he's got a "better" solution to a common problem than the four numbskulls who wrote the book!

buggyfunbunny Jul '07

not mentioned yet, so Holub's book on design patterns is an antidote for mindless doting. and he's an iconoclast of The First Water.

tub Jul '07

I have read A Pattern Language and I have to agree with Mark Dominus's sentiments that Design Patterns just doesn't match up to A Pattern Language. Where A Pattern Language has 250 short patterns, Design Patterns has about 20 huge ones. Design Patterns tends to treat them only at one level, and in too much detail. A Pattern Language looks at patterns over a much larger scale and in more general terms.

I think a cs book that did follow in the same vain as A Pattern Language would cover things from a high level, such as patterns for deployment, down to parsers, and different patterns used there. But in all would remain fairly general. It's probably too much for a single book to cover though.

As far the Design Patterns book goes as a Object Oriented learning mechanism it does a reasonable job at that. Though I think if a developer is at the stage where they could read and grok Design Patterns, they should read "Refactoring" by Martin Fowler first. I'd almost say that it's more true to the spirit of the original patterns book than Design Patterns is.

Whatever Jul '07

I don't see the problem.

If I have something I need to do, I simply file a bug report with the compiler vendor demanding they add the relevant feature to the language, then in less than a year I purchase the upgrade with the feature that solves my problem without me needing to do anything.

What's wrong with that? All software development problems without exception are flaws in languages, after all. Hell, it only took half an hour to redesign perl with all required features that have been discovered in computing, so how long can it take to add a new feature to an existing language. Clearly having a way to solve a known problem using existing compilers and language features is pathetically stupid when you can just upgrade your language with new features.

Well, ok, the language reference now requires three large shipping containers because there's a whole lot of problems that have needed solved and the number of language features has gotten a little larger than it started out, but the electronic version can be searched easily enough.

Mitch Jul '07

If you rethink design patterns, maybe you are ready for refactoring

"Refactoring: Improving the Design of Existing Code"

a href="http://www.amazon.com/Refactoring-Improving-Design-Existing-Code/dp/0201485672"http://www.amazon.com/Refactoring-Improving-Design-Existing-

Code/dp/0201485672/a

Cheers!

Telos Jul '07

"If I have something I need to do, I simply file a bug report with the compiler vendor demanding they add the relevant feature to the language, then in less than a year I purchase the upgrade with the feature that solves my problem without me needing to do anything.

What's wrong with that?"

Your manager is willing to wait a year for a project?

Allied Jul '07

"If I have something I need to do, I simply file a bug report with the compiler vendor demanding they add the relevant feature to the language, then in less than a year I purchase the upgrade with the feature that solves my problem without me needing to do anything.

What's wrong with that?"

I HOPE you're joking.

KartikA Jul '07

Further reading on Christopher Alexander: Richard Gabriel's "Patterns of Software": http://www.dreamsongs.com/Files/PatternsOfSoftware.pdf

More recently, Steve Yegge had something to say about QWAN: http://steve-yegge.blogspot.com/2007/01/pinocchio-problem.html

Gertjan Jul '07

I agree to a certain extend; the Design Patterns book basically contains the recipes for several recurring problems in software development. The key is to go a little bit higher in abstraction level and think about the idea of recognizing a recurring pattern while you are developing software. And that is a powerful concept. But, yes, not everyone will interpret the book like that

that.

On my own blog, I wrote an article about the vocabulary of software design, which more or less reflects the level of abstraction to recognize design and patterns in software architectures:

http://www.code-muse.com/blog/?p=4

Alexander's book is now surely high on my wishlist of books to read!

EricM Jul '07

For this one time I have to slightly disagree with you Jeff, and while overusage of Design Patterns is a clear sign of a clueless architect that should probably find a different line of work I think Design Patterns are an essential tool. If you use them often you are doing something wrong? I don't like that flat out blanket statement, it just doesn't hold up, I mean perhaps to a UI developer thats true, but a back end developer or an architect it's definitely not the case. Time and time again I have used patterns but I think most people miss the point of patterns, you don't have to follow a pattern to its concise design. I see them more as a blue print that you mold to your needs, change this, change that, until it works for you, don't fit a square peg into a round hole just for the sake of using the pattern.

More often than not most developers who shy away from patterns in my experience have done so because they failed to adapt them to their needs or they simply used the wrong pattern for the wrong thing. I have actually seen a UI developer use the factory pattern for a UI control?!

Patterns are a necessity to information reuse, if you build a bridge you don't start from scratch do you? Every single bridge in the world uses reinforced concrete, I mean I hate analogies like this in software where you compare software architecture to real world architecture but it gets the point across…

I do agree that its good to understand your lines, don't use patterns for the sake of using them, for god sake this happens so often its quit annoying. I have been on projects where the lead developer would hand everyone a design pattern book and say read this, this is how we are going to design our application. Which left all the junior developers with a "Huh?" look on their face all confused… They are there to aid you in a specific common problem…

I do agree with you about th cut-paste mentality with design patterns, DONT DO IT, but should developers shy away from patterns? WHY? It's a tool in their arsenal and they should use it

when necessary!

M1167 Jul '07

People keep bashing on GoF for conflicting reasons: it's too simple, requiring no thought; it's too complicated, the patterns are huge; it's too low level, bogged by implementations instead of abstractions; it's too abstract, not considering real world implementation issues; and so on…

The truth is that even the best books are not for everyone. Some will call Fred Brooks' book largely irrelevant. "Code Complete" is huge and contains a lot of advice that is just a matter of style and not quality. "Effective C++" is too simple, and only shows toy examples and not real world examples. The list goes on.
Yet most of us agree that these books are very important, and many would agree that they are among the best available. GoF (and other books) are very easy targets. So is C++ (in a different way, perhaps).
People like bashing them, but the truth is that they enabled breakthroughs in real world software development, and are still very useful tools on the road to quality.

On another, related note:
Steve Yegge also doesn't like design patterns, and believes like others here that they point to deficiency in the language.

"The problem is, about 1/3 to 1/2 of them were basically cover-ups for deficiencies in C++ that don't exist in other languages. Although I'm not a huge Perl fan anymore, I have to admit the Perl community caught on to this first (or at least funniest). They pointed out that many of these so-called patterns were actually an implementation of Functional Programming in C++."

http://steve.yegge.googlepages.com/singleton-considered-stupid

That's fine. Maybe we should all use LISP, or at least Haskell. C++ really sucks, or whatever. Man, overload resolution is a real hack.

In the meanwhile, I have real work to do.
I really need to get that image display UI for that digital camera working. I don't know of any LISP implementation that runs well enough on a digital camera. LISP is simply not an option (yet?). So I am gonna do it in C++. I might use a design pattern here or there, even (GASP!) a Visitor. Because I recognize that patterns are simply that: recurring OO design solutions to recurring problems.

When you are holding that digital camera and browsing through your hundreds of photos in lightning speed, organizing them, and resorting them, you will be glad I used C++, and employed all tools and methods available to me. You won't know it, of course. Your software will simply work. It will work very well: fast, easy, no crashes. But it will do so because I considered the design carefully, coded with discipline, and tested as best as I could. The design patterns book is just another tool.

DaveR Jul '07

I think your argument has a rather painful fallacy in it…The problem isn't Design Patterns, the problem is the quality of the engineer/architect that tries to use them. If I apply your logic in the construction arena, then we must surely blame the failure of buildings and structures on the slavish use of AutoCAD and poorly designed hammers. In other words, the tools are rarely the issue–be they patterns or languages.

It still comes down to the fact that good engineers and architects know how to use their tools well and that poor ones, even when given the best 'chisels and screwdrivers', can't program their way out of a wet paper bag. Plain and simple. 15 years in the trenches has yet to teach me differently.

ChristopherH Jul '07

The problem isn't even design patterns related.

I think the problem is newbie programmers or those that seem to still have the behavior of wanting to blindly implement this cool-new-thing ™ they learned… and implement it all over the place.

Lacking sight of the big picture and the true point is also lacking for these people.

We've all been guilty of facination of the cool-new-thing, yet over time we learn better what to do with this information. Especially after you've been maintaining a codebase for years, you pick up a few things 😉

Maybe it's part of the learning process? I don't know. At least comment the code that you're patterning all over the place 😃

ps - I also am a fan of Martin Fowler's Refactoring book and the Head First Design Patterns

book (in fact I'm a fan of all of Kathy Sierra's books/Head First).

Stuart_Gray Jul '07

The GOF book was the first pattern book I read, and I found it to be a terrible introduction to the world of patterns. I can (now) appreciate the patterns it contains and it's role in creating a wider awareness of patterns in software, but it presents a very narrow perspective of what patterns are all about.

I agree with Parveen about The Timeless Way of Building by Christopher Alexander. It the book that gave me an "aha!" moment about the nature of patterns and is great if you want to understand the theory behind what makes them work patterns.

"A Pattern Language" is a catalog of architectural patterns based on the method described in Timeless Way i.e. it's a demonstration of how to apply the theory in practice. It's a great example of patterns and a pattern language in practice, but Timeless Way contains the real nuggets.

For a broader perspective, I found the Pattern Languages of program design series (PLoP) from hillside.net to be quite informative - they really opened my eyes to the different types of problems to which patterns can be applied, not just software design architecture.

FrankW Jul '07

Patterns aren't things you *implement* (in the solution space). They are things you *recognize* (in the problem space).

DavidO Jul '07

I would certainly agree with everyone about the misuse of patterns. Which is why I think that reading "Object-Oriented Design Heuristics" by Arthur Riel is so helpful. While patterns tell you what to do, heuristics tell you why - or at least make you weight the factors for that decision.

CptBongue Jul '07

Sorry, but I too have to disagree

When designing, patterns may seem cumbersome and overkill, but once your applications have grown considerably, patterns make your application more maintainable as they allow you to get a “Feel” of how they work, and before looking at the code, you often know what library and what chunk of code is broken, without even having to dive in or debug. I’ve even experienced opportunities where patterns, combined with naming conventions allow you to say: “I’m looking for this method name, in this class file, in this library or namespace”, or “This method is called this or the other, so it does exactly this, and nothing else”.

Granted, patterns aren’t the silver bullet. If you know beforehand to what extent your application or library will grow, then you should consider more of a rapid development approach, if you don’t, put in the extra effort.

T_E_D154 Jul '07

For this one time I have to slightly disagree with you Jeff

Funny. This is the one time I find myself in total agreement with Jeff. Devotion to design patterns seems to be a sure indicator of programming mediocrity. The worst part of this is that an unthinking pattern applier often misses the one requirement detail that makes their chosen pattern totally inappropriate (without some kind of modification). Worse are the ones who like to rectify this issue by layering on yet another pattern…

The fact of the matter is that software development is hard and requires real thought. You can’t solve all problems with a finite set of some “n” cookie cutters.

Lest I come off as a total luddite, I rather like the refactoring book. There’s a fun book on “anti-patterns” that I liked a lot too.

Mark_L Jul '07

Don’t apply patterns as dogma! Consider the patterns, understand the patterns and apply as necessary to solve a problem.

There is way much prose about this topic.

Miguel_Ping Jul '07

Patterns are a map, a guideline.
Code is the terrain, sometimes it’s rocky

Code is the terrain, sometimes it's rocky.
It's nice to have a map.
Don't always trust the map, trust the terrain instead.

That's my haiku for design patterns

MaximK Jul '07

I think Alexander's newest work "Nature of Order" not only sheds light on "A pattern language", but also has the potential to have a better impact in the way we build software than "Design Patterns" has in general. If you have not read the series, I suggest you read it as it is a much more mature collection of ideas than "A pattern language" was. How and if it will impact the way we right software is yet to be seen. But fun to think about.

Brad Jul '07

I know 2 things for sure:

1. The more I learn the less I know. I become ever more cognizant of things I should know that I don't have time to learn.
2. This must be the most confusing time for programmers. OO is *the* proper way to develop systems. This is established by years of practice… no wait, maybe OO can encourage bloat and complexity, FP is way better… wait, who the heck knows? One things for sure, design patterns result in better code… wait, maybe they really only expose an underlying problem. Well, I do know that I should make code flexible and extensible… wait, that has been the source of many bloated, complex systems too.

Janitorial arts - it's the only way to go.

codinghorror Jul '07

T.E.D., what book do you mean? Is it this book?

http://www.amazon.com/AntiPatterns-Refactoring-Software-Architectures-Projects/dp/0471197130/

ChrisM Jul '07

Design Patterns is, like a number of other books are, more like a combination dictionary and thesaurus, and largely ought to be used like one. I'd never try to write a novel without a thesaurus and/or dictionary handy, but I certainly wouldn't expect them to instruct me on writing great prose.

It's unfortunate that so few people understand the patterns (indeed, Patterns) that they're working with, but taking away their "instruction manual" will not force them to rethink their working habits; They'll just find another manual.

What I find strange is that Jeff and others like him will work to convince us that we live in an age of infinte monkeys, but will then turn around and blame Shakespeare for their existance.

T_E_D155 Jul '07

what book do you mean? Is it this book?

a href="http://www.amazon.com/AntiPatterns-Refactoring-Software-Architectures-Projects/dp/0471197130/"http://www.amazon.com/AntiPatterns-Refactoring-Software-Architectures-Projects/dp/0471197130//a

Yup, that's the one.

Jim Jul '07

Misuse of patterns - and posters here using words like "dogma" and "commandments" - is not the fault of the patterns but the practitioners.

The GoF and other pattern books present a handful of solutions to a handful of problems. To appreciate them and use them well, or to make up your own solutions to new problems, you really need to understand the principals and values that led somebody to think there was a problem and why they think the pattern solution is good. If you're focused on separation of concerns, isolating sites of change, managing dependencies and so on, the patterns give some solid hints and examples. You can judge whether the problem is likely to cause you so much pain that you need the solution … or not. GoF goes down much better with something like Robert Martin's Patterns, Principals and Practices. Without that understanding, one can wind up pushing "dogma" and "commandments" without reason.

Gene Jul '07

I use design patterns to 1) solve particular problems in the code “trenches” and 2) help myself design code by the concept of orthogonality DRY Principle (refer to The Pragmatic Programmer). I don’t do it as an exercise of faith. Nor do I do it without trying other solutions.

Also, the I agree with the comment about junior programmers using design patterns. Let the JPs code JP stuff.

Another thought: Perhaps the term “Patterns” is incorrect here. Perhaps “Constructs” would be better, because “patterns” has the connotation of being mindlessly, guilelessly repetitive and reactive. Maybe “Templates” would be even better.

SteveS Jul '07

I read a Pattern language cover to cover. It is a very worthwhile read.

However, more generally: I find it is always worthwhile to go to the source inspiration for any ‘classic’ text. Norman’s The Design of Everyday Things has a fantasic bibiography in the back.

codinghorror Jul '07

Jeff and others like him will work to convince us that we live in an age of infinte monkeys

We do, in fact, live in an age of infinite monkeys:
http://www.codinghorror.com/blog/archives/000864.html

But not all monkeys are created equal.

I find it is always worthwhile to go to the source inspiration for any ‘classic’ text. Norman’s The Design of Everyday Things has a fantasic bibiography in the back

Shh! You’re giving away all my blogging secrets! 🙂

Tim_Dudra Jul '07

Jeff wrote:
But I have two specific issues with the book: 1. Design patterns are a form of complexity. As with all complexity, I’d rather see developers focus on simpler solutions before going straight to a complex recipe of design patterns.

Design patterns are only a form of complexity when viewed at a code level. When viewed in the abstract, they are a source of simplicity because they provide the ability to understand the forest without knowing the details of every tree. The problem with “simpler [code] solutions” is that the code is the only way to understand what is going on and that breaks communication of ideas because those ideas can only be communicated at the code level. This makes it harder for the rest of the team, QA, Program/Product Management, Other Development Teams, … to understand what is going on inside the box.

It is true that rabid adherence to every gory detail of the patterns in Design Patterns can lead to overly complex code but, as others have said, these patterns are to be adjusted to the needs of the task at hand. If the full power of the Abstract Factory or Factory Method is not needed then simplify them down to the core idea which is “place construction code in a separate method so it doesn’t clutter up the mainline logic”. If you aren’t bridging two different inheritance hierarchies, don’t use the bridge. If the target API is close to the needs of your project, use a facade to wrap it - don’t bother with the adapter.

In arguing for his point, Jeff shows that his argument wears no clothes (to paraphrase an earlier point). Jeff and another individual shared these thoughts:
Who are these architects that I always hear about that blindly apply the Gospel of the Design Patterns instead of thinking? I’ve not seen them, and the GoF book doesn’t seem to advocate it as far as I can tell.
Indeed. Ever read “The Bible”, or “The Quran”? Strangely, those books don’t advocate a lot of the things you’ll see people doing in the name of them, either.

The Bible and The Quran are often used as justification for strange arguments - that doesn’t make those arguments right. Those who apply the gospel of Design Patterns instead of thinking are not right either. In condemning the book because it is a source of unnecessary complexity, Jeff seems to be saying that because some zealots misinterpret its intent, the book is bad. But his counterargument above seems to imply that, despite being misinterpreted by some zealots, The Bible and The Quran are not bad. Was the cake tasty Jeff? Do you still have it around? You can’t have it both ways.

Now perhaps I am misinterpreting Jeff’s intent since he doesn’t seem to disagree with the idea of patterns, just the overreliance of some in the software industry on this one book instead of the ideas that were embraced by Mr. Alexander years ago.

My concern with Jeff’s post here is more that it will be used as ammunition for those who don’t want to design, who don’t want to worry about future changes that are known to be coming do…n the req…irements nine…thous…ho…ill arg…e that a “simpler sol…tion” is better. I…ooked

down the requirements pipe, those who will argue that a "simpler solution" is better. I worked with people like that. One very senior developer (senior in years, not skill in my humble opinion) argued that design patterns would do us no good - that you could not come up with a solution as elegant as the one he came up with without playing with the code for a while. It turns out that the elegant solution he came up with, after six months of uncontrolled coding because managers were afraid to interfere with “the great one” (eyes roll, sarcasm spews from mouth), was simply the interpreter pattern and I showed it to him in a book in 5 minutes once he finally was willing to discuss his grandiose project. All this work of his was justified by the argument of “simpler solutions”. Other work of his and his team that was justified by “simpler solutions” included a 40000 line class, some 2000 line methods in which the set of variables that tracked the state of the operation changed as you proceeded through the code, methods with cyclomatic complexity in excess of 60, etc. Because their code was “simple” it could not be discussed outside of the code - and therefore there were no sanity checks to stop these debacles.

Design patterns is an excellent book and should be on every developers shelf. It enhances team communication, it provides a source of inspiration when one is confronted by a tough problem, it allows developers to avoid six month ratholes because they are convinced nobody else can help them or they are too proud or arrogant to ask around. Because a few zealots overuse the book and religiously adhere to what it says does not mean the book does not provide a valuable service.

baya Jul '07

We all know actual implementation tends to deviate somewhat from the initial design generally because of exactly what Jeff is pointing out here - in a dynamic environment there’s no way one designer can anticipate all the requirements of a system.

Design patterns are useful when you need to communicate the design of something as opposed to its implementation. If I tell someone “xxx is supposed to act as a Facade here” or “yyy is a Factory”, we all have a more precise idea of the subject. Whether or not xxx actually is a Facade or yyy actually is a factory is a subjective matter.

Personally I’m looking forward to what I think is the next big thing here: a more expressive “Design Goal” methodology capable of evaluating the cost and benefit of applying different design patterns to a given problem.

YoniR Jul '07

If you are designing software in an OO way you are using at least some of the GoF Design Patterns. If you are using the GoF Design Patterns you are doing at least a little OO design.

The GoF Design Patterns were not an invention but rather a compilation of simple OO design practices which had been in use for years and a vocabulary to be used for them. Some patterns may be better than others but the book states pro's and con's so I don't see that as a problem.

Fregas1 Jul '07

Jeff,

You know, when I first started learning design patterns, i was a little turned off for exactly the reasons you mentioned: over-complicated, cookie cutter solutions that like any piece of technology, they can and are frequently abused.

Collection and iterator patterns are almost useless now, because they are built into java and .net. The same is true of many "Enterprise Architecture" patterns, because they are built into one framework or another, if not the language itself. Truly, many times design patterns grow into a framework or even a language feature. This isn't a bad thing at all, and i think it shows that the concepts behind patterns are inherently useful.

That being said, sometimes i really DO need something like a singleton pattern. It would be nice if the language had a built-in way to make one, although many times its a variation and not exactly what the GOF or "Head First" books show in their source code. For example, in asp.net, i need a singleton where the object is global to the current http request, but not global to the whole app domain. I can reuse the concept of the singleton with a slight variation. I can also tell another developer: "This is a variation the singleton pattern" and they'll understand. Which is the whole point of design patterns. So I still think they are useful, but your points are definintely well taken.

I'm curious though, do you never use any design patterns yourself?

Craig

Fred_Ross Jul '07

I also regard design patterns as a bad idea, but for a different reason. I regard myself as a

mathematician, though I've made my living programming before, spent some years as a physicist, and now I'm doing biology. To me, design patterns are a first step, but they're incredibly incomplete.

When I am faced with a pattern in whatever system I'm studying/crafting/whatever, I pull it out and encode it as a nice, clean mathematical structure. I play with it until a useful decomposition becomes obvious, and in many cases that decomposition is in terms of mathematical structures I consider old friends.

But it goes in the other direction, too. A design pattern is an interpretive dance: you recognize that you are going through this motion again, and the recognition lets you draw on past experience to go through it a little more easily. Instead, I take the mathematical structure I have distilled, and consider it as an element of some class, then seek the natural structure on that class. Many people protest this as too abstract and a waste of time. What they don't realize is that very often by abstracting as hard as you can, you find a space where the problem is trivial. Then you take your solution just plug in whatever the given case is.

You can do this in any language. Object oriented programming isn't particularly helpful for it. Functional programming is helpful for exactly two reasons: it maps very easily to how mathematicians think, and functional languages usually make it incredibly easy to define data structures. My two choices would be Haskell, followed by ALGOL68.

RussellJ Jul '07

Jeff,

Couldn't have said this better myself! 🙂 Like many others, I use GoF and other materials as a starting reference point and adapt them to fit the problem at hand.

Oh, and for the guy who thinks you don't build a bridge from scratch each time – newsflash – you DO! At least if you want it to work. Every bridge is different. The physics of a suspension bridge are GENERALLY the same (which is akin to saying all C# programs are written in C#), this [only] gives you a starting point from which to base your design, but the parameters of the bridge's location, what it spans, weather, etc. all come into play during the design and implemenation of the bridge.

Jim_Cooper Jul '07

"Dominus believes that the Gang of Four version obstructs Alexander's message, replacing

Dominus believes that the Gang-of-Four version obstructs Alexander's message, replacing actual thought and insight with a plodding, mindless, cut-and-paste code generation template mentality"

Anyone who believes that is missing the message in the GoF. They specifically say patterns are **not** code templates. Admittedly this is a difficult thing to get across to some people.

codinghorror Jul '07

Admittedly this is a difficult thing to get across to some people

Yes, this book full of code that looks *exactly* like cut and paste code templates? It's not a book of cut and paste code templates!

Jim_Cooper Jul '07

"Yes, this book full of code that looks *exactly* like cut and paste code templates? It's not a book of cut and paste code templates!"

Well, they don't look like cut and paste templates to me, but then I read the words in between them 🙂

I think that is largely the problem - too many people don't take the time to read what the book actually says (I've even met people prepared to argue about what it says, having never read it).

It may not be the easiest book in the world to read, but Dominus is assigning blame in the wrong place. The GoF definitely did *not* intend their book to be a set of code templates. That point is made over and over again (in the discussion of every pattern, in fact).

JustinC Jul '07

It seems like design patterns are great for understanding how to properly solve a particular problem in an application but perhaps if you have the same problem occuring over and over again rather than re-implement the same design pattern it might be a good time to think about implementing a domain specific language.

This book:
Domain-Specific Development with Visual Studio DSL Tools

(http://www.amazon.com/Domain-Specific-Development-Visual-Studio-Microsoft/dp/0321398203/)

Has some very interesting comments in the first few chapters. Mostly it's a manual for using the tools but at least the first 100 pages or so talks about when it is appropriate to use a DSL and the virtues of DSL's. I found it interesting.

EsadH Jul '07

Hi,

A number of design patterns (for example factory/singleton) have very trivial implementations in a dynamic language such as ruby, and actually lot of patterns are there to fight the static nature of the language.

Why would you need the decorator pattern in ruby, when you can simply add new methods to any object at runtime? (there's actually no difference between runtime and compile time as the language is interpreted).

Andy Jul '07

Are you joking?

This article is the hype that I don't believe. Far from being irrelevant: although the book is now over a decade old and is still relevant. Overwhelmingly the book has changed many people's coding style.

1 Unless your system is small then not using patterns is more complex. I cannot understand your first criticism, even fundamentally. If you have people who fail to grasp GoF on the project then they are still getting to grips with OO. Imaging designing a GUI without some form of Observer. You often end up with an unthinkable tangle of code: not much of which you can reuse. There are many other examples.

2 Patterns aren't supposed to be boiler-plate for one: which is why the community backed off building pattern support directly into lanagages. Varients of patterns are often used or adapted to suit the problem at hand.

As for turning programmers into macro-processors, that's like saying authors are dictionaries. It's more of a The whole point about patterns is to allow others to pick up your code. The fact

a lot of code looks the same is a good thing as actually may developers aren't actually as talented as they think they are: they learn by copying and adapting. Which is why GoF is here to stay for the foreseeable future. Comparing patterns to idioms in other languages does not weaken the case for patterns as it's not the same thing being compared. How an iterator can be compared to a perl foreach loop, or a bit of lisp is nonsense. They are different tools: this is a valed attack on some of the languages where the patterns are most use. Both perl and lisp might allow the programmer to be expressive but hands up who would like to pick up the maintenance? Even so Patterns are not irrelevant in both.

As for architectual patterns, my understanding was that Alexander is mostly read by computer professionals and not architects at all. I'm sure you could equally argue that GoF has far more influence over the software community than Alexander over architecture.

Tim_Howland Jul '07

I'm with Keebler on July 5, 2007 01:36 PM; Andy on July 10, 2007 02:49 AM. Forgive me for just citing them. 😉

Carl_Dreher Jul '07

I also have objections to the Design Pattern religion, but for a reason not mentioned here: a complex concept cannot be reduced to a single name that means something to everyone.

For example, what is a "decorator" pattern? Can anyone honestly say that the word "decorator" describes a function you regularly do in programming? In my mind, a decorator is someone that comes in and throws colorful pillows on my couch.

I have bjections to most of the GOF naming convention. The names don't describe the problem they are attempting to solve, so trying to find a pattern that applies to a particular problem takes way to much time. It is simply impossible to abstract a complex issue to a single word. (Although politicians try mightily to do that.)

I've actually had a job interviewer ask me what various patterns do BY NAME. That is how insidious this religion has become. He never asked how I would handle a given programming problem. It was all about having memorized the book.

This objection extends to much of the programming landscape. It seems that everytime some fresh hot programmer (re: wet behind the ears) thinks he or she has written something new

(…doubtful) they try to give it a catchy name, or worse, an acronym that is already in use. (I've actually seen RAM and DMA used for someone's pet design!) It is like criptic comments in code. Give it up! Use a multi-word description.

RJBotting Jul '07

I read Christopher Alexander's "A Pattern Language" book within weeks of it's publication because I though it might describe the kind of "patterns" I was using to design software at that time… I enjoyed it. I liked the way the patterns fitted togerher and the discussion of the forces that the patterns try to resolve.

But I thought that some of the architectural patterns were unworkable… and that it had no application in computing.

Duh.

Herr_Ziffer Jul '07

Wait. We're supposed to actually use design patterns? I thought they were just terms you use at design meetings to sound impressive in front of your boss, e.g. he says, 'functionality XYZ is essential to this project. Do we know how to handle it?' And without looking at the requirements you reply, 'Sure, a combination of the strategy pattern and the singleton should take care of it.'

Design patterns are also a good way to haze people you don't want to hire during job interviews, so you can go back to your boss and say, 'boss, your nephew seems like a great guy but he isn't quite up to snuff when it comes to design patterns.

So now you're telling me that all along we were also supposed to be using these design patterns in our coding? What's next, actually pairing up to do extreme programming?

StephanF Jul '07

I've seen more garbage written in the name of some design pattern than anything else so when someone I work with says the words design pattern my hair just stands on end.

It's been said GoF is mostly a band-aid for C++ to make up for features standard in functional languages. E.g. Peter Norvig argued in 1998 70% of the GoF patterns are superfluous in Lisp. I suggest the same applies now to JavaScript and Ruby and that the other 30% were

I suggest the same applies now to JavaScript and Ruby and that the other 30% were nonsense right from the start anyway (see e.g. Interpreter).

JeffR Jul '07

"If you know beforehand to what extent your application or library will grow, then you should consider more of a rapid development approach, if you don't, put in the extra effort.

CptBongue on July 6, 2007 06:15 AM"

I think you've identified a key issue. Many of the patterns carry with them the implicit assumption that the design will change. The choice of pattern is often based on how or where that change will occur.

The more subtle assumption is that it will be more effective (cheaper, faster, better) to design the architecture to anticipate the change (or at least its type).

In practice, change is not inevitable, the type of change is not always predictable (even at the 10,000 foot level), and sometimes it is more effective to wait for the change to come before dealing with it.

The best choice will depend on the entire context of the project including operating environment, programming language, staff, requirements, market, budget, maturity of the company etc.

For example, my experience in start-ups suggests that any work that you can put off for a year or two will increase significantly your chances of survival. Sometimes it's literally true that there isn't time to do it "right" but there may be time to do it over.

BillS Jul '07

Saying that Design Patterns are only part of a larger concept of patterns misses the essential point of how GoF design patterns were developed specifically and the large number of OOP concepts they clarified. The value of the GoF work is that it clarifies OOP, and while OOP is certainly not the be-all and end-all of programming (ask any FORTH programmer), it has been a great aid in moving the whole discipline of programming ahead. By the same token, design patterns move it even further.

To say that one needs to go to the "source" of design patterns to really understand why GoF is talking about does not resonate. It's like saying that one needs to learn Latin and French to

talking about does not ressonate. It's like saying that one needs to learn Latin and French to really understand English. Nothing wrong with that, but it's really not necessary.

BillS Jul '07

Saying that Design Patterns are only part of a larger concept of patterns misses the essential point of how GoF design patterns were developed specifically and the large number of OOP concepts they clarified. The value of the GoF work is that it clarifies OOP, and while OOP is certainly not the be-all and end-all of programming (ask any FORTH programmer), it has been a great aid in moving the whole discipline of programming ahead. By the same token, design patterns move it even further.

To say that one needs to go to the "source" of design patterns to really understand why GoF is talking about does not ressonate. It's like saying that one needs to learn Latin and French to really understand English. Nothing wrong with that, but it's really not necessary.

Will Nov '07

I do not think that the GoF had the same agenda as Alexander; Alexander seems to have wanted to push the idea of distributed responsibility; the GoF seem to have had a set of generic solutions to software problems and a good way of documenting them for comparison and clarity. The GoF left the project management side out because they'd have been on a hiding to nowhere with all of the existing software project management mumbo-jumbo. Whereas the construction/architecture sector has stuck with the same management method forever, which Alexander was trying to change. So Alexander's book covers two subjects: how to manage projects, and design patterns for 'consistency and harmony of overall design'; the GoF book only covers the latter because they assume that either you are an experienced enough developer/project manager to understand project management, or that you'll seek your project methodology from other sources.

The GoF never presented their book or design patterns in general as cookie-cutter templates; they presented them as potential solutions, and provided hints and examples so that developers could learn to identify a problem space that one of the design patterns could fit rather than having to reinvent the wheel. The patterns aren't presented as dogma or a replacement for good design, they are presented as aids to conceptualisation, analysis, and design.

If developers have been lazy enough to read the book without thinking about what the patterns are or even reading the introduction, or are too dense to understand, they should be

patterns are or even reading the introduction, or are too dense to understand, they should be relegated to junior positions under the mentorship of developers not afraid to use their brains.

ClintonT Feb '10

I also disagree with you on this one.

Regarding your first issue:
Design patterns aren't a recipe, but a language. If your system is too complex, it's the designer's fault, not the fault of the language the designer used.

The problem space defines the complexity required for a solution. Language does not add to this complexity, but lack of language may - particularly if used by an inexperienced or clumsy designer.

Regarding your second issue:
The premise "bad engineering = rewriting boilerplate" is a statement in and of itself and completely accurate. The follow-up comment about broken language is disjoint and nonsensical.

It's well established that some problems are more easily described with some constructs of some languages. The concept of telescoping languages transcends "your language sucks" arguments. If your language doesn't support your desired functionality, build it or find an existing solution. This is why a thriving developer community is one of the most important aspects in selecting a language - whether you're choosing a programming language, or a system design language.

Jon_V Feb '10

The problem with design patterns is that the people that think they are most interesting are people to whom they are original concepts (crap coders) and it becomes a tool for them to cover their junk code tracks. This is the true legacy of code design patterns; despite the advantages of a 'common language' most of the time you end up using the common language to explain just how bad a mess someone has made.

And b) there is nothing great about Alexanders book. Just because it has a real architecture connection doesn't give it any special legitimacy.

Aaron_G Feb '10

I guess I'm lucky in that I never got exposed to the dogma in question. I know about the design patterns and I do use them on occasion, but even back in university (which was engineering, not CS, by the way) I'd learned that these patterns were um, patterns. Not rules.

Sometimes a polymorphic design with a factory class can save me a lot of headaches (emphasis on the "some" - I can recall exactly four instances over more than 15 projects). And iterators are nice, although they're kind of baked into the C# language by now.

So I agree with the post, I suppose, but the problem it's referring to is just alien to me. I simply haven't encountered this strict adherence to design patterns, in my work or earlier formal education. Is it really so widespread?