Patterns at Work

Patterns are not substitutes for design thinking, they are channels for it

Kevlin Henney

Patterns and Best Practices

Alice had become interested in **patterns** fairly early on. Over the last decade she had found them useful for communicating design ideas between developers and architects. When used effectively, they promoted a broad, reflective, and respectful design dialog. However, she noted that all too often the seminal work in the field, *Design Patterns* by the "Gang of Four," tended to dominate the perception of what patterns were. This certainly appeared true of some of her colleagues in her new place of work.

Beyond the Gang of Four

Alice had been flicking through the documentation for their systems when Bob, one of the other architects in the department, wandered over.

"Hi, finding your way around the system OK?"

"Yes, thanks, Bob. Nice portal-style web pages for the projects. I found all the build details, API documentation, and source code, no problem. I thought the use of the Wiki was good for capturing some of the key design decisions and their relationships."

"Thanks! It took a bit of effort to get to this point, but the Agile initiative we kicked off last year gave us the perfect excuse to sort out the build and bring everything under one roof. We also moved away from some of the dead-tree-inspired documentation that was being actively ignored. Carol was responsible for extracting the essence of that forest and originally populating most of the Wiki."

"I wonder, have Carol—or you—considered using a pattern-based style for capturing some of the decisions and style of the systems?"

"Design patterns? Oh yeah, I'm a big fan of those. I use **Singleton** a lot."

"Uh huh," Alice was just able to stop the thought bubble turning into a speech bubble. She had met (too) many Singletons before, and knew all too well that they were often daubed over a system as a somewhat overdressed stand-in for global variables, and would often lead to subtle architectural gridlock over the lifetime of a system. "Well, there's more to patterns than the Gang of Four—and more to the Gang of Four than Singleton." She smiled.

"Maybe, but I'm not sure it would help us. I mean, I know we use Singleton, but is there really much else beyond a few Adapters and Observers? Given that we're trying to keep the documentation pretty lean, I don't think it adds much."

“Well, looking through the various projects, I can see a number of well-known patterns: Layers, Broker, Publisher–Subscriber, Lifecycle Callback, Session Component, Gateway, Data Transfer Object, to list a few. Being able to refer to these by name helps to establish a common design vocabulary that establish the style of various systems.”

“Well-known vocab? Hmm, I know the Gang-of-Four patterns, but I haven’t heard of any of those others, except perhaps Layers. But that’s not a pattern, surely? I mean, it’s a well-known architectural solution that addresses the problem of large-scale organization of a system with different technologies, abstraction concepts, and multiple developers. The idea of layering has been around for ages and you can find it in lots of systems. So, why is it suddenly a pattern? Isn’t that just...I dunno...bandwagoning and rebranding?”

“Good patterns identify recurring designs that capture proven design practice. From that point of view, the twenty-three patterns in the GoF book form the tip of the iceberg. The fact that you’ve seen examples of Layers in many systems—and that it’s considered a designed solution to a problem—is what makes it a pattern. It’s also been documented as such. What also makes the pattern perspective a little different is that instead of just naming a solution, the focus of a pattern is also on the problem and the problem’s context.”

“Hmm, I guess that’s a more constructive way of looking at it, but I have to admit that I’m only really familiar with the GoF patterns and a couple of others I’ve read about in articles.”

“I’ll send you some references, if you’re interested. If you get the chance to look through them, you will probably find that you’re familiar with many of the patterns, but by sight rather than by name. For instance, I noticed in one system that the **Interceptor** pattern was used to allow instrumentation and filtering behavior to be introduced for objects created by a framework.”

“Which one?” Bob leaned over as Alice pulled up a window with the source code. “Oh yes, that one. Interceptor is what it’s called, eh? Dave convinced me to use this approach in the C# code rather than the approach I used in the C++ framework, which with hindsight I would say was based on the **Template Method** pattern. For adding instrumentation and other extra-functional behavior, this Interceptor approach was less intrusive on the main class hierarchy.”

“Even if you didn’t know the name of the pattern, that pattern-based thinking—interplay between problem forces and solution consequences—is very useful. I’ve also just learned something about the design! It’s the kind of thing I reckon would be worth noting down.”

“Hmm, you could be right. Feel free to add a note on the Wiki!”

Testing Times

“Bob,” called Alice, “I’m having a hard time writing a unit test for this class. It’s just a simple piece of functionality I’m trying to add, but the plumbing in the class seems to be drawing in a lot of external dependencies. It looks like I need to set up some

config files and fire up a version of the database as a minimum. All of which makes it somewhat more of an integration test than a unit test."

"Good to see you getting so hands-on," replied Bob, wandering over.

"You bet—an **Architect also implements**!" Alice saw the blank look on Bob's face. "It's a pattern." Bob raised a quizzical eyebrow. "Not a design pattern: an organizational pattern."

"A pattern for every occasion, eh?" smiled Bob. "What have organizations got to do with object orientation, though?"

"Well, patterns aren't just about objects. That's where they first grew to popularity in software, but they have been used to document recurring solutions to problems in other design styles and domains—enterprise architecture, concurrency, organizational structure, user-interface design, you name it. Predating the Gang of Four, the first paper on using patterns in software focused on a set of UI patterns. And then, of course, don't forget that patterns were first about building architecture before they were about software architecture."

"True enough, now I remember. Christopher Alexander was the guy's name, right?"

"That's right. I can send you some links and things about some of these other pattern types, if you want."

"Uh, sure. Anyway, back to your testing problem."

"Yes, well, I'm not sure if it's so much a problem of testing as a problem of design. It's become noticeable because it's making it hard to write simple tests. Do you want to take a look?"

"Sure." Bob sat down and scrolled through the code, with the occasional, "Yuh," nod, "Hmm," shake of the head, and "Uh, huh." Eventually, "OK, I think I see the issue, and I'll admit this bit's not as well documented or self-explanatory as it could be. You've hit a small constellation of Singletons that wrap up external resources, such as config files, databases, networking, and so on. For testing, we have some neat tricks and infrastructure to work around them—you just have to know which tricks."

"Such as...?"

Bob proceeded to explain the various techniques used in each system, depending on programming language and deployment environment, but paused part way through. "Alice, you look a little surprised."

"I'm just a little taken aback by the ingenuity involved and effort invested in working around the Singletons. I guess if objects were people, I would be thinking of handing them their notice or reskilling them instead of apologizing for them and covering up their problems."

"Ah, um, that's one way of looking at it, I guess."

“Any problems with multithreading?”

“Oh, do you mean the **Double-Checked Locking** pattern? Yeah, that’s a non-GoF pattern I came across and read up on. We introduced some fixes and Singleton preloads to compensate for the various platform differences.”

“Hmm, still sounds like a lot of buck for not much bang. Do you mind if I take a look at some of those Singletons I’ll need and see how they are used? It would help me familiarize myself with the system, and also see what forces were being resolved, and which forces might have gone unresolved. I might be able to come up with something. Maybe just an interface here and there.”

“Sure, no harm in that. A fresh pair of eyes is always useful. Feel free to make changes, just”—regaining his composure, with a twinkle in his eyes—“don’t break the build or the programmatic interface.”

In Context

“Alice, I noticed you made some changes to some of the Singleton classes. They seem quite, well, restrained. Given what you said about workarounds, I guess I was half expecting you to rout every Singleton in your path!”

Alice smiled coyly, “Oh, it’s not like that. All I did was loosen things up a little. Having each Singleton implement a usage interface means that it doesn’t matter whether or not there is a Singleton involved: you just pass the instance around. The consequent changes were quite minimal: an extra parameter or field here and there, based on the new interfaces, makes the architecture more pluggable. Singleton was quite hardwired for some of the design situations—a bit more Highlander than necessary.”

“Highlander?”

“Yeah, remember the film? ‘There can be only one!’”

“Cute.”

“In many of the cases the uniqueness was not actually a proper constraint, just a coincidental property of the application at the time it was written. Looking through them, many of the Singletons were playing the role of **Context Objects**, which is now easier as they can be passed around behind an interface rather than hardwired to the concrete type. Also easier to substitute mock objects for testing. Here, take a look.”

Bob leaned over and scrolled through Alice’s tests. “Uh huh...yuh...I see...hmm. Neat.” Bob looked up from the code, “Context Object? Is that a pattern?”

“Yes. Not a GoF one, although if you look closely at the Gang-of-Four book you can see it plays a role within the Interpreter pattern. Context Object has been documented a number of times and in a number of ways—I can send you some links.”

“Yeah, that would be cool, thanks. And are there any other places I can find out about patterns?”

“Yes, as well as the usual links and books, there are some other online resources that you can look at. *Software Engineering Radio*, for instance, has a number of podcast interviews with various pattern authors. Offline, patterns normally appear in some form at a number of development conferences. Conferences, such as OOPSLA, OOP, and JAOO, often feature key members of the pattern community. For instance, a few years ago I saw Frank Buschmann give a very practical talk on designing with patterns—I think it was called *Patterns at Work*.”

“Nice title. I think we have some conference budget left for the year, so I’ll keep my eyes open. In the meantime, podcasts are always good for the drive home, so I’ll look into that! Anyway, speaking of Singletons, are we saying that Singleton is an anti-pattern?”

“Well, that’s not a particularly constructive term, but Singleton certainly contradicts the Gang-of-Four’s own design principle of ‘program to an interface, not an implementation’—you end up programming to an instance, not just an implementation!”

“Good point. So, I guess it needs to be used more sparingly and perhaps with a better understanding of its consequences...Say, Alice, given the architectural knowledge wrapped in these patterns, do you reckon that patterns might be useful for capturing some of the decisions that shape our systems?”

“You know, Bob, that sounds like a great idea.”

That Friday Feeling

A couple of weeks later, Alice overheard Bob chatting to one of the developers on the other side of the open-plan partition: “...is definitely one way of doing it—we’ve certainly got a lot of code that looks like that. Recently, however, I’ve started adopting a slightly more loosely coupled style...Uh huh, yeah, with hindsight a lot of our Singletons were kind of OK at the time—and better than a load of exposed `statics`—but we’re finding a lot of friction between them and our agile initiative. Of course, I’m sure a couple of them are fine, but you’ve got to look at the context of each design situation more carefully, and perhaps be a little more sensitive to the forces at play...Anti-pattern? Well, that’s not a very constructive term, but Singleton is certainly misunderstood and undoubtedly overused. I think if you think about this from a more pluggable perspective, you may consider passing some of the environmental dependencies through in a more encapsulated form. In this case, I would consider using a Context Object...Context Object? Oh, that’s a pattern...No, it’s not in the Gang-of-Four book, but if you look closely...”

Alice smiled to herself and logged off. It was Friday. The weekend beckoned. And Monday was looking a whole lot brighter.

Critical Thinking Questions

- What documented patterns are you familiar with?

- Which patterns have you found to be most influential and beneficial in your architecture or development process?

- Which patterns have been most problematic, either because the patterns themselves were dysfunctional or because they were misapplied?

Sources

Gamma E, Helm R, Johnson R, Vlissides J. 1995. Design Patterns. Addison-Wesley.

Beyond the Gang of Four

Buschmann F, Henney K, Schmidt D C. 2007. Pattern-Oriented Software Architecture, volume 4: A Pattern Language for Distributed Computing. Wiley.

Buschmann F, Henney K, Schmidt D C. 2007. Pattern-Oriented Software Architecture, volume 5: On Patterns and Pattern Languages. Wiley.

Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M. 1996. Pattern-Oriented Software Architecture, volume 1: A System of Patterns. Wiley.

Coplien J. 1996. Software Patterns. SIGS. <http://users.rcn.com/jcoplien/Patterns/WhitePaper/>

Fowler M. 2003. Patterns of Enterprise Application Architecture. Addison-Wesley.

Kircher M, Jain P. 2004. Pattern-Oriented Software Architecture, volume 3: Patterns for Resource Management. Wiley.

Schmidt D C, Stal M, Rohnert H, Buschmann F. 2000. Pattern-Oriented Software Architecture, volume 2: Patterns for Concurrent and Networked Objects. Wiley.

Vlissides J. 1998. Pattern Hatching. Addison-Wesley.

Völter M, Schmid A, Wolff E. 2002. Server Component Patterns. Wiley.

Testing Times

Beck K, Cunningham W. 1987. Using Pattern Languages for Object-Oriented Programs. OOPSLA. <http://c2.com/doc/oopsla87.html>

Coplien J, Harrison N. 2005. Organizational Patterns for Agile Software Development. Prentice Hall. <http://www.easycomp.org/cgi-bin/OrgPatterns>

Graham I. 2003. A Pattern Language for Web Usability. Addison-Wesley.

Meyers S, Alexandrescu A. July and August 2004. C++ and the Perils of Double-Checked Locking. Dr. Dobb's Journal. <http://www.aristeia.com/Papers/DDJ_Jul_Aug_2004_revised.pdf>

O'Reilly T. September 2005. What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>

In Context

Alur D, Crupi J, Malks D. 2003. Core J2EE Patterns. Second edition. Addison-Wesley.

Henney K. 2005. Context Encapsulation. 10th European Conference on Pattern Languages of Programs. <http://www.two-sdg.demon.co.uk/curbralan/papers/europlop/ContextEncapsulation.pdf>

Kelly A. 2003. Encapsulated Context. 8th European Conference on Pattern Languages of Programming. <http://www.allankelly.net/patterns/encapsulatecontext.pdf>

Krishna A, Schmidt D C, Stal M. 2005. Context Object. 12th Pattern Languages of Programming Conference. <http://www.cs.wustl.edu/~schmidt/PDF/Context-Object-Pattern.pdf>

Software Engineering Radio. <http://www.se-radio.net/>

Zdun U. 2005. Patterns of Argument Passing. 4th Nordic Conference of Pattern Languages of Programs. <http://www.infosys.tuwien.ac.at/Staff/zdun/publications/arguments.pdf>

About the Author

Kevlin Henney is an independent consultant based in the UK, practicing consultancy, training, and development across a number of domains. He specializes in programming languages, development process, patterns, and software architecture. He has been a columnist for various magazines and sites, including *The Register*, *Application Development Advisor*, *Java Report*, *JavaSpektrum*, *C++ Report*, and *C/C++ Users Journal*. Kevlin is coauthor of two volumes in the *Pattern-Oriented Software Architecture* series.

Glossary

Architect Also Implements. A pattern that ensures architects do not become disconnected from the reality of practice.

Context Object. A pattern that captures execution context in object form so it can be passed to services or plug-in component.

Double-Checked Locking. A somewhat subtle pattern that reduces cost of locking for thread-safe lazy resource acquisition.

Interceptor. A pattern that supports discreet plug-in behavior for framework events, such as filtering, instrumentation, or logging.

Pattern. A named and recurring solution to a problem within a given context, most often—but not necessarily—associated with design.

Singleton. A somewhat problematic pattern that aims to constrain a class's instantiation to a single instance, but is often used as a global.

Template Method. A pattern that defines a skeleton of an algorithm within an operation, deferring some steps to subclasses.