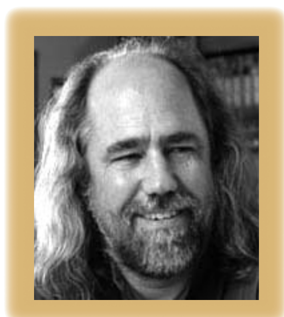
Editor: Grady Booch ■ IBM ■ architecture@booch.com

The Well-Tempered Architecture

Grady Booch

Virtually all well-structured music, music that pleases the ear and moves the spirit, is full of patterns.

In Douglas Hofstadter's brilliant book *Gödel, Escher, Bach: An Eternal Golden Braid* (20th anniversary edition, Penguin Press Science, 2000), Achilles, the Tortoise, the Crab, and the Anteater have a conversation. The Anteater remarks:



> *Fugues have that interesting property, that each of their voices is a piece of music in itself; and thus a fugue might be thought of as a collection of several distinct pieces of music, all based on one single theme, and all played simultaneously. And it is up to the listener (or his subconscious) to decide whether it should be perceived as a unit, or as a collection of independent parts, all of which harmonize.*

The fugue is perhaps the most sophisticated compositional style in Western musical tradition and, if done right, is a work of fierce beauty born of elegant complexity. Music is a primal medium of expression, and while some musicians in every age push the envelope of contemporary practice, there have emerged over the centuries common patterns of song structure, motifs, and even scales to which our ears have become accustomed. As it turns out, these musical patterns aren't so much constraining as they are liberating. Each level of structure imposes a discipline that limits a musical work from being something else and thus distinguishes one music piece from another. The writer who is faced with a completely blank page will often face writer's block—a blank page is sometimes intimidating because, although empty, it's filled with possibilities. But once the first words are written, the problem is futher constrained, making it possible for the writer to proceed. An artist who chooses some set of constraints—for example, deciding to draw portraits in chalk—has the freedom to innovate within that chosen medium.

Musical patterns

The structure of a song forms its general shape. Most Western music is repetitious, and a lot of popular music uses a 32-bar form consisting of two verses, a bridge, and then another verse (the AABA form). Much religious choral music takes an even simpler form, verse-chorus-verse-chorus (ABAB). Blues music has a distinctively different form, shaped around a 12-measure cycle.

Within that general shape, many musical works have an inner motif. For example, consider the easily recognizable musical phrase in "Hedwig's Theme" by John Williams from the *Harry Potter* films, the recurring phrases throughout the music in the TV series *Battlestar Galactica*, or the theme in "My Sweet Lord" by George Harrison. (The motif in Harrison's song was so similar to that in Ronald Mack's song "He's So Fine" that it resulted in a lengthy plagiarism suit.)

At the core of every song are its notes and silences. A musician can choose from many different rhythmic patterns: 2/4 time (often used in marches), 3/4 time (the form of the waltz), or the most common rhythm, 4/4 time. A musician may even choose different tunings and scales. In contemporary Western music, the most common scale consists of a run of seven notes that repeat every octave. Many, many other scales are possible, such as Pythagorean tempering (which is mathematically pure but dissonant the further you get from the starting point), the chromatic scale, the pentatonic scale (found in many Ori-

 0740-7459/07/$25.00 © 2007 IEEE

Authorized licensed use limited to: Cal Poly State University. Downloaded on October 15, 2008 at 13:08 from IEEE Xplore. Restrictions apply.

ental styles), or the maqam mode (found in traditional Arabic music).

Software patterns

Similarly, all well-structured software-intensive systems are full of patterns. Architectural patterns serve the same role as song structure; design patterns and musical motifs are at the same level of abstraction; programmatic idioms and musical rhythms and scales are isomorphic.

Architectural patterns shape the general texture of an automated system. However, we don't yet have names for all these patterns; this is indeed one of the goals of my *Handbook of Software Architecture*. We do know that

- the structure of a Web-centric system is quite different from that of a time-triggered embedded system, and
- a system that relies on the presence of a semantically dense persistent state (such as in a database) will be shaped differently from one that relies primarily on the state of the real world.

In the fullness of time, every development culture in every domain tends to converge on a small set of architectural patterns, because extensive use will have proven that these patterns are good enough to resolve the many forces that weigh in on that problem domain.

Design patterns are the motifs of software, the reoccurring themes that weave their way through a software-intensive system. In well-structured systems, these patterns provide a namable texture that cuts across levels. A design pattern is a collaboration, a society of classes or components that work together to carry out some interesting behavior. Poorly designed systems contain no such patterns, so their internal structures have no regularity. Like a musical score with no rhyme or reason, a software system without design patterns, either intentional or accidental, shouts of disorder and cacophony.

Idioms, a term Jim Coplien first used in this context, denote patterns at the level of the programming language itself. Programming style is typically language- and team-specific. For example, many unwritten rules of "good" programming style in C++ are quite different from those found in Lisp or Python. Similarly, although I know of no empirical studies to back this up, the code I've studied that was written in Silicon Valley smells subtly different than that written in Europe (where there is generally a greater emphasis on mathematical precision), New York City, or India.

Harvesting patterns

While conducting archeological digs on the systems under study for the *Handbook*, I've encountered only a relatively small number of development organizations that have an intentional culture of patterns. I find in these merry few that their developers typically communicate using the language of the Gang-of-Four design patterns or other pattern authors; the more sophisticated groups have even composed their own patterns. More common, however, are reasonably well-structured systems whose architects never really explicitly introduced patterns. Here, the developers' experience led them to best practices that had worked previously and hence were applied again, without really naming those patterns as such or even knowing that they had applied a pattern. In such cases, the process of untangling and then understanding these systems is largely the activity of discovering and then naming these common motifs. Naming entails either matching them to patterns that have already been identified in the literature or, in the case of unique patterns, authoring a new design pattern that codifies that practice.

To aid my digs, I've cataloged almost 2,000 design patterns found in the literature. As I harvest a system's architecture, I'll draw from this catalog, finding the best match to the motifs I encounter. The absence of any match suggests the need for a new design pattern to be added to the catalog.

IEEE Standard 1471, Recommended Practice for Architectural Description for Software-Intensive Systems, introduces the idea of representing a system's architecture from the perspective of multiple interacting points of view. Although the standard emphasizes these views' importance, it doesn't specify which views are the most relevant for a given problem domain. For a variety of reasons, I've selected Philippe Kruchten's 4+1 model view of architecture because its family of five views is largely sufficient and complete for my purposes. In a well-structured system, each of these views—the logical, process, implementation, deployment, and use case views—can be described by the patterns they embody.

A lot of what I do in the field these days involves helping organizations establish a sound practice of architectural governance. I've come to realize that architectural patterns and design patterns are

- *constructive*, in the sense that people often use patterns to direct a system's forward engineering; and
- *transformative*, in the sense that a developer can use patterns to refactor a system into a more advanced or a simpler system.

Either way, there's considerable value in filling a software-intensive system with architectural and design patterns: such systems are easier to understand and adapt because of their regularity and simplicity.

Grady Booch is an IBM Fellow and one of the UML's original authors. He also developed the Booch method of software development (*Object-Oriented Analysis and Design*, Addison-Wesley, 1993). He's working on a handbook of architectural patterns, available at www.booch.com/architecture. Contact him at architecture@booch.com.

QUESTIONS? COMMENTS?
IEEE Software wants to hear from you!
EMAIL software@computer.org

Authorized licensed use limited to: Cal Poly State University. Downloaded on October 15, 2008 at 13:08 from IEEE Xplore. Restrictions apply.