Editor: **Eoin Woods**
Endava
eoin.woods@artechra.com

Architecture Haiku
A Case Study in Lean Documentation

Michael Keeling

SOFTWARE ARCHITECTS ARE taught to create comprehensive, complete documentation. We're taught to stamp out all assumptions and explicitly articulate our designs. Without restraint, however, architecture descriptions can become long-winded treatises on a system's design. In the course of being thorough, we often inadvertently obfuscate the architectural vision in documents that consequently aren't read.

An architecture haiku is a "quick-to-build, uber-terse design description" that lets you distill a software-intensive system's architecture to a single piece of paper.[1] Creating an architecture haiku is more than just an exercise in terse documentation. To write an effective haiku, you must focus on only essential design decisions and rationale. Much like its poetic cousin, an architecture haiku follows strict conventions and can express a mountain of information with a tiny footprint.

What Is an Architecture Haiku?

An architecture haiku aims to capture the architecture's most important details on a single piece of paper. Placing extreme constraints on the architecture description forces architects to focus on the design's most important aspects. With only one page to work with, there simply isn't space to waste on extraneous details.

For an architecture haiku to be comprehensive, Fairbanks recommended that it include[1,2]

- a brief summary of the overall solution,
- a list of important technical constraints,
- a high-level summary of key functional requirements,
- a prioritized list of quality attributes,
- a brief explanation of design decisions, including a rationale and tradeoffs,
- a list of architectural styles and patterns used, and
- only those diagrams that add meaning beyond the information already on the page.

An important key to success is to focus on the most essential ideas for each item on this list. If something isn't important, don't include it. If an idea requires more detail, don't skimp—but don't overdo it. The point isn't to omit or reduce critical information but to highlight it. Removing noise that might otherwise drown out the most important ideas helps shine a spotlight on them.

Architecture haiku creates a *cognitive trigger* that facilitates the recall of essential contextual information about a design decision. Rather than fully describing a decision using comprehensive prose and diagrams, assume the reader already has much of the required knowledge, and focus on the critical details.[2]

Two of the most common examples of a cognitive trigger are architectural styles and patterns. With a single word

0740-7459/15/$31.00 © 2015 IEEE

or phrase—for example, "layer" or "pipe and filter"—you can reference a wealth of knowledge that doesn't need to be repeated in your documentation. Less formally, architects often use homegrown metaphors[3] or even refer to systems the team has previously built, to the same effect. Any common reference point—such as a whiteboard discussion, a framework, an argument, or even other published material—can be a cognitive trigger that connects an essential decision to the context and background supporting it.[2]

For example, an architecture haiku might provide a "take-away lesson" from a collaborative whiteboard discussion, the details of which might be stored as raw notes and whiteboard photos. In many cases, a couple of sentences or a few bullet points will get the point across. From a team perspective, an architecture haiku should comprehensively describe the whole system for small or routine designs. For large or extremely complex systems, an architecture haiku might describe only a subcomponent for which a team is responsible. Or, it might provide an overview at the highest level to help create a common vision among distributed teams. You might even try creating a haiku of haikus, opting to use single-page descriptions for each viewpoint that comes together to create a comprehensive description. In any case, the objective is the same: record sufficient information so that team members can recall essential design decisions based on previous collaborations or on knowledge from other sources.

Effectively Using Architecture Haiku

While using architecture haiku, my project teams at IBM discovered how to consistently create useful one-page architecture descriptions.

> An architecture haiku aims to capture the architecture's most important details on a single piece of paper.

My teams were experienced but small—typically, three to five engineers with an average of at least five years' software industry experience. The teams always worked directly with customers, so the customer's architectural experience and documentation requirements often influenced how a team approached design and documentation.

Although architecture haiku was extremely effective, it wasn't the right tool for every project or situation. Using it effectively requires some thought and practice. Here, I describe our best advice for creating effective architecture haikus.

Start with a Shared Understanding

When we first learned about architecture haiku, we thought it was a great idea—but there were some problems. Although we were well versed in enterprise search concepts, not everyone spoke the same software architecture dialect, and some engineers weren't familiar with certain concepts. For example, some engineers were familiar with the 4+1 view model, whereas others preferred Software Engineering Institute viewpoints, and still others used IEEE terminology. Some engineers talked about nonfunctional requirements, and others discussed quality attributes. We quickly resolved these differences, but not before creating some confusion that was occasionally amplified by the sparse language of a haiku.

Architecture haiku relies heavily on domain-specific vocabulary and cultural norms within a team to build cognitive triggers. So, readers must be educated in basic software architecture concepts to understand implied assumptions in the haiku. This includes a basic understanding of architectural drivers; quality attribute scenarios; the use of structures, architectural styles, and patterns; and how to identify and evaluate design tradeoffs. The team must also agree on a vocabulary.

Explore First, Then Record Decisions

Creating an architecture haiku doesn't replace design exploration, nor does it magically transfer knowledge from one teammate to another. A good haiku will act as a medium for recalling essential information about design decisions such as previous discussions and collaborative whiteboard sessions. Without this context, an architecture haiku is just a terse document.

You can create context by collaboratively exploring the design space with stakeholders. Collaborative exploration lets multiple stakeholders walk through the design process and arrive at a shared understanding of the design. Then, the haiku be-

comes a lightweight monument to those creative insights, a reminder of what you decided to build and why.

In our experience, an architecture haiku is excellent for communication and recall but terrible for exploration. The biggest problem we experienced was deadlock. Artificially constraining design activities to a single page too early constricted creativity and didn't leave breathing room to explore alternatives. We found that the best strategy is to explore first—collaboratively when possible—then record decisions in the haiku, updating it as you learn more about the architecture.

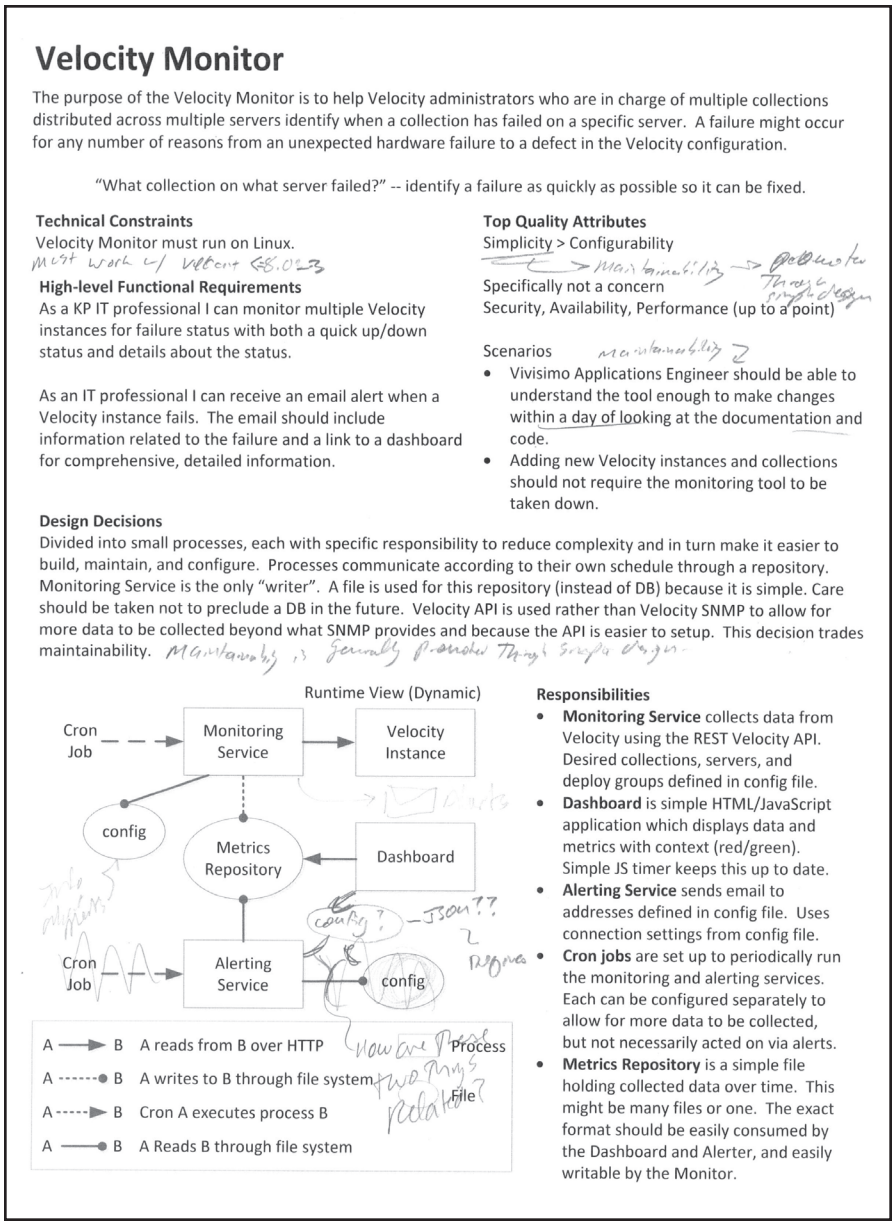

FIGURE 1. An architecture haiku that we printed out and then annotated as we learned more about the system and refined the architecture.[2]

Treat the Haiku as a Living Document

One benefit of constraining an architecture haiku to a single page is that it can provide guidance in a way that can easily evolve as the team learns more about the system being built. Figure 1 shows the haiku we used to describe the Velocity Monitor, a software tool that monitors the crawling and indexing status of search collections. This tool's scope was relatively small but included several interesting integration points with other systems.

We recorded design decisions in this haiku as brief prose, including rejected choices. It included a diagram of the most informative view of the architecture because the design didn't exploit any architectural patterns.

Note the annotations on the page. As construction began, we learned that some decisions could be improved, so we altered the design accordingly. Updating the haiku was simply a matter of scribbling notes and scratching out elements on the page. Such changes are expected and should be embraced.

Use the Haiku as an Outline for Future Documentation

Using an architecture haiku as a recall mechanism for established rationale, context, and decisions requires that the readers remember information. Although this is usually easy

for project teams, there are risks to consider, such as long-term recall of information, turnover, and project transition.

Refreshing teams owing to turnover or project transition is fairly common. In the former case, engineers might simply no longer be available to consult on a project because they've moved on to new roles or organizations. In the latter case, a project might be transitioned to a new team for maintenance or future work. Over months or years, memories decay, people forget specifics, and the effectiveness of the cognitive triggers captured in an architecture haiku decreases.

The simplest way to mitigate these risks is to write a more traditional architecture description before the team disbands. This documentation aims to capture ideas referenced in the haiku that might exist only as an oral history within the team.

Once you've created an architecture haiku, writing a tight, well-organized architecture description becomes much easier. By creating the haiku first, the team has already identified the essential information, much like an outline. Furthermore, the haiku can serve as an executive summary or be included in the appendix of a longer document. Using the haiku in this way can help achieve balance among agility, delivery speed, and the need for documentation.

Create a Template

This tip might seem obvious, but it's important. To promote communication within the team, our haikus follow the same basic template. This template promotes basic graphic design practices, including the use of white space and alignment. We also don't allow fonts smaller than 10 points. Small fonts are too difficult to read and defeat the purpose by allowing for too much text.

Following a template makes it easier to quickly discern the documentation's salient points and helps provide guidance to the haiku writers. We've distributed our template as two PowerPoint slides: one with the MegaVista example (see the sidebar "Architecture Haiku for the MegaVista Project") and the other with the tips shared in this article. You can access the template at www.neverletdown.net/2015/03/architecture-haiku.html.

It's easy to shrink the font and cram words onto a page. Producing a highly valuable, information-dense resource that people actually want to read is more difficult and much more rewarding.

When I was in high school, my calculus teacher let us bring a single page of notes to the final exam—an officially sanctioned "cheat sheet." I spent weeks preparing it, carefully selecting concepts I thought would be on the exam, including complex equations, proofs, and examples. On exam day, much to my surprise (but not to my teacher's), I hardly used my cheat sheet at all. Creating it was the best study guide I could have used. In many ways, architecture haiku is a cheat sheet for your architecture.

Philippe Kruchten said, "If, instead of a fully robust process, I were permitted to develop only one document, model, or other artifact in support of a software project, a short, well-crafted Vision document would be my choice."[4] The intent of architecture haiku is very much in this spirit.

Architecture haiku helps your team focus on the most essential information relevant to the architecture, provides clear guidance for construction, and encourages collaboration. Combining architecture haiku with traditional documentation is a clear winner. Although architecture haiku in its most literal form is a useful tool, it also offers a chance for general reflection on architecture documentation practices. The number of pages used is less important than designing documentation that's right for your team and situation. Whether you use architecture haiku as the only documentation or simply as a litmus test for how well the architecture is understood, the goal is always to create high-value documentation that maximizes awareness and understanding across the team.

In conclusion,

Design for your team
Less is more when describing
Your architecture.

References

1. G. Fairbanks, "Architecture Haiku," blog, 2011; http://georgefairbanks.com/architecture-haiku.
2. M. Keeling, "Creating an Architecture Oral History: Minimalist Techniques for Describing Systems," presentation at the 8th Software Eng. Inst. Architecture Technology User Network Conf. (SATURN 12), 2012; http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=20330.
3. M. Keeling and M. Velichansky, "Making Metaphors That Matter," *Proc. 2011 Agile Conf.* (AGILE 11), 2011, pp. 256–262.
4. D. Leffingwell and D. Widrig, *Managing Software Requirements: A Unified Approach,* Addison-Wesley Professional, 1999, p. 169.

MICHAEL KEELING is a software engineer at IBM. Contact him at mkeeling@neverletdown.net.

ARCHITECTURE HAIKU FOR THE MEGAVISTA PROJECT

Clearly listing the tradeoffs in the design helped us with decision making during implementation. Because the team was experienced in building search applications, it was sufficient to list the key architectural patterns used in the design. Diagramming was essential when we collaborated around whiteboards before creating the haiku. However, it added little value to the haiku itself, so we didn't include it.[1]

MEGAVISTA GUIDES SEARCH

A publicly consumed search solution based on the Velocity Search platform that will help users find companies of interest.

Business Drivers

- Better search encourages listed companies to purchase advertising, metadata supplements, and paid listings.
- Faster results for users and advanced search options (such as refinement and spelling suggest) bring more users to the site.

Key Tradeoffs

- Crawlability over Maintainability—Need to split the data source across multiple search collections, distributed across several servers. Incidentally promotes Scalability.
- Crawlability and Queryability over Configurability—Highly configurable system reduces speed of crawl and query.
- Flexibility/Development Speed over Cost—Stakeholders are not 100 percent on all features and have a strong desire to go live as soon as possible.
- Modifiability over Maintainability—IT will maintain the system and know databases, not Engine, so when they make a change it should be detected and reflected.

Architecture Styles and Patterns Used

3-tier (data, crawl, query)
Source-Selector—promotes reliability
Query Redundancy—promotes availability
Virtual Documents—all data crawled
Document Enqueue—for website data
Collection Sharding—promotes crawlability
Crawler Clone—promotes availability
Geolocation Lookup—promotes maintainability/modifiability

Top Quality Attributes

Crawlability > Queryability > Scalability

- Crawlability—A batch of up to 900 metadata updates are published to the database and reflected in search within 3 minutes.
- Queryability—An average-sized result set can be calculated within 2 seconds of Engine receiving the query.
- Scalability—The size of the data source increases beyond current capacity, and the system can be easily expanded to deal with this.

Design Decisions with Rationale

- Database must be crawled as multiple views (vice-JOINed) to avoid stressing it too much. Currently 1.6 million companies, each with dozens of metadata fields. Metadata joined via Virtual Document (company ID = vse-key).
- Company website must be crawled and website content made searchable under the company (i.e. return a single results with all content searchable as the same document). Website data joined with metadata via Virtual Document (company ID = vse-key).
- Full recrawl / week, DB refresh / 30 sec. if no crawl running. Full recrawl will pick up changes made to company websites (external data). 30-sec. refresh enables catchup for metadata crawls on DB updated every 60 seconds at most.
- Collections divided by company ID. Dividing allows maximum crawl throughput, using company ID (e.g. collection A has IDs 1–10) allows us to control partitioning.
- Paid listings and keyword weighting factors used for relevancy calculation will come from the database (not Engine configuration). This will allow the business unit to make changes quickly using their existing tools.

Reference

1. M. Keeling, "Creating an Architecture Oral History: Minimalist Techniques for Describing Systems," presentation at the 8th Software Eng. Inst. Architecture Technology User Network Conf. (SATURN 12), 2012; http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=20330.