Position Paper: Architecture Island

John Klein (john.klein@computer.org)

Revised 29 April 2019

1 Introduction

The challenge statement for this workshop is to identify "the most important information" about software architecture that I will need to take to the new colony. This position paper takes a very personal perspective on the needed information, and divides the information into three categories. The first category is information that I would use during architecture design. The second category is information that I would want on-hand as I mentor the next generation of architects. The third category is general reference material.

I ascribe use the strategy of using "small framework of well-known tools" [Elliott-McCrea, 2019], choosing to try to master a few methods and tools that can be applied in many situations. For any particular situation, my portfolio of tools and methods may be suboptimal, yet still allows me to produce acceptable results.

I work mostly in the domain of data-centric enterprise applications–my packing list is sufficient to allow me to dabble in other domains (e.g., mobile or cyberphysical), but if I was going to have to cover that ground, I would pack more domain-specific material.

2 John's Packing List

Some of the artifacts that I list would be used for more than one purpose. I list each in the category where I find it to be essential.

2.1 Architecture Design

- [Cervantes and Kazman, 2016]: This book outlines the design process.
- [Fowler, 2002]: I've found this book of application patterns useful for many domains.
- [Hohpe and Woolf, 2004]: Fowler's patterns tell how to build applications. Hohpe's patterns help you connect application together.

- [Bass et al., 2013]: Software Architecture in Practice contains tactics, which are not discussed anywhere else. It also contains quality attribute-specific checklists. If I'm pressed for space, I would take only Part 2 of this book.
- [Rozanski and Woods, 2012]: Rozanski and Woods tell you how to document your design.
- [Beauvoir, 2019]: I've found the Archimate language, and the Archi tool, extremely useful for defining context and top-level structure–the precision enforced by the language sharpens and clarifies my thinking.
- [PlantUML, 2019]: PlantUML enables lightweight UML sketching for several diagram types.
- [Software Engineering Institute, 2018]: The student handouts for the ATAM Evaluator Training provide enough information about architecture evaluation in general, and the ATAM in particular.
- [Dean and Barroso, 2013]: Jeff Dean's paper on long tail latency in large-scale systems provides a checklist for evaluation and observability.

2.2 Mentoring

- [Keeling, 2017]: Keeling's book is full of ways to bring the team along with you as you design the architecture and develop the system.
- [Elliott-McCrea, 2019]: This explains my strategy of minimizing the diversity of our tool, method, and technology stack portfolio.
- [Abadi, 2012]: I would bring Abadi's PACELC framing of the CAP theorem so that I wouldn't have to keep explaining it.
- [Conklin, 2005]: This book is a practical introduction to collaborative problem solving, and provides a method to structure the discussion.
- I'd like to bring Grady Booch's Handbook of Software Architecture, but it seems to have disappeared from the interweb.

2.3 General Reference

- [Kruchten et al., 2019]: I haven't read this yet, but the topic seems important.
- [Thomas, 2013]: Architects must write code. I chose Ruby as my general-purpose scripting and data-wrangling language, but I don't write enough to stay sharp. The *Pickaxe* book provides a quick and comprehensive reference.
- [Pilgrim, 2004]: Like the *Pickaxe* book, but for Python.

3 Summary

This may seem like a small list, but I am also bringing my years of experience. These artifacts represent material that I cannot recreate ad hoc, or cannot remember (eg the pattern books). If I'm also responsible for my office supplies, I'd be sure to pack as large a whiteboard as the loadmaster will allow (along with a lifetime supply of markers), and a digital camera.

References

Daniel J. Abadi. Consistency tradeoffs in modern distributed database system design: Cap is only part of the story. *Computer*, 45(2):37–42, 2012. doi: 10.1109/MC.2012.33.

Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice.* Addison-Wesley, 3rd edition, 2013.

Phillip Beauvoir. Archi – open source archimate modelling, 2019. URL `https://www.archimatetool.com`.

Humberto Cervantes and Rick Kazman. *Designing Software Architectures: A Practical Approach.* The SEI Series in Software Engineering. Addison-Wesley, Boston, MA, USA, 2016.

Jeff Conklin. *Dialogue Mapping: Building Shared Understanding of Wicked Problems.* Wiley, 2005.

Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, February 2013. doi: 10.1145/2408776.2408794.

Kellan Elliott-McCrea. Questions for a new technology., 2019. URL `https://kellanem.com/notes/new-tech`.

Martin Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional, 2002.

Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns.* Martin Fowler Signature Books. Addison-Wesley Professional, 2004.

Michael Keeling. *Design It!: From Programmer to Software Architect.* The Pragmatic Programmers. Pragmatic Bookshelf, 2017.

Philippe Kruchten, Robert Nord, and Ipek Ozkaya. *Managing Technical Debt: Reducing Friction in Software Development.* Addison-Wesley Professional, 2019.

Mark Pilgrim. *Dive into Python.* Apress, 2004.

PlantUML. PlantUML in a nutshell, 2019. URL `http://plantuml.com`.

Nick Rozanski and Eoin Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2nd edition, 2012.

Software Engineering Institute. ATAM evaluator training-student materials. 2018.

Dave Thomas. *Programming Ruby 1.9 & 2.0: The Pragmatic Programmers' Guide*. The Facets of Ruby. Pragmatic Bookshelf, 2013.