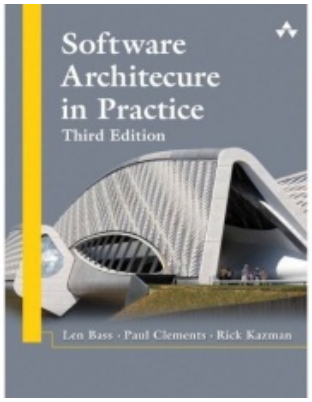
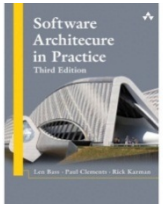
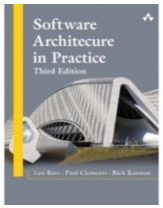


Chapter 19: Architecture, Implementation, and Testing



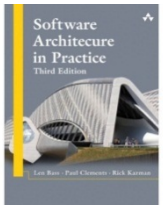
Chapter Outline

- Architecture and Implementation
- Architecture and Testing
- Summary



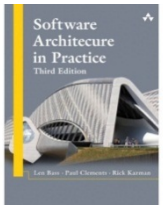
Architecture and Implementation

- Four techniques to help keep the code and the architecture consistent:
  - Embedding the design in the code
  - Frameworks
  - Code templates
  - Keeping code and architecture consistent (i.e. avoiding “architecture erosion”)



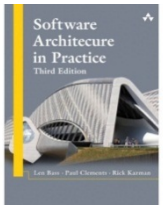
Embedding the Design in the Code

- Architecture acts as a blueprint for implementation. This means
  - Implementers know what architectural structure they are implementing. E.g. layer, pub/sub, MVC, broker, …
  - They can document the architectural structure in the code as comments. Then anyone picking up the code will know some of the constraints.
  - Projects should have conventions about how to do this. Then tools can automatically relate the code and the architecture.



Frameworks

- A framework is a reusable set of classes organized around a particular theme.
- A programmer uses the services provided by a framework.
- Examples are
  - Ruby on Rails is based on MVC is designed for web applications. It offers the ability to gather information from the web server, talk to or query the database, and render templates.
  - AUTOSAR is designed for the computers inside of automobiles.



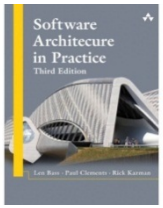
Code Templates

- A code template is collection of code within which the programmer provides application specific portions.
- The example below implements a failover protocol:

In the event that a failure is detected in a critical-application component, a switchover occurs as follows:

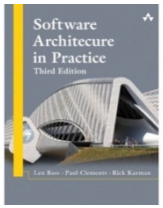
1. A secondary copy, executing in parallel in background on a different processor, is promoted to the new primary.
2. The new primary reconstitutes with the application's clients
3. A new secondary is started to serve as a backup for the new primary.
4. The newly started secondary announces itself to the new primary.

If failure is detected within a secondary, a new one is started on some other processor.



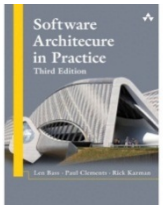
Code Templates

- Process:
  - Use the code template for every critical component that must have a hot standby.
  - Place application specific code in fixed places within the template.



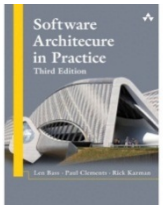
Advantages of Code Templates

- Components with similar properties behave in a similar fashion.
- Template only needs to be debugged once.
- Complicated portions can be completed by skilled personnel and handed off to less skilled personnel.



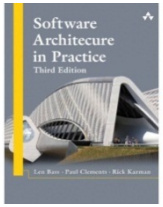
Keeping Code and Architecture Consistent

- The implementation will drift away from the documented architecture.
- Implementers may make decisions that are not consistent either with each other or with the architecture.
- The architecture may not have foreseen all eventualities that come up.



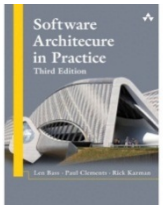
Preventing Architecture Erosion

- Use tools to enforce architectural constraints.
  - can have architecture rules added that are enforced during a build or check in.
- Mark documentation as out of date when erosion occurs. Will give more credence to remaining portion.
- Schedule documentation/code synchronization times.



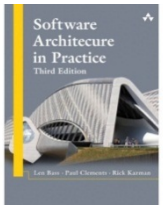
Architecture and Testing

- Unit test
- Integration test
- Network effects
- Test activities



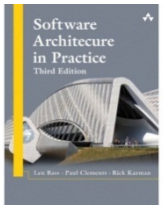
Unit Test

- Architecture defines the units that are to be tested. They are components or modules.
- Architecture defines the responsibilities and interactions of the units.
- Test harness will drive the element to be tested. The test harness can test:
  - Responsibilities for functional correctness
  - Performance through synthetic loads
  - Availability through fault injection. i.e. what happens if a component on which the component being tested does not respond.
- Modifiability requirements can also be tested by assigning changes to test teams.



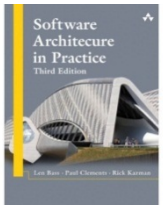
Integration Test

- As with unit test, integration test can test functionality, performance, availability, and security.
- Security can be tested by having the test harness execute various attack scenarios.
- Systems may degrade after being run for a long time if resources are not freed or a configuration is incorrectly specified.



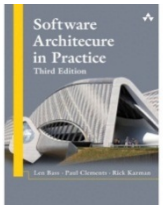
Network Effects

- Suppose an error causes a 2% performance degradation.
  - This is within normal variation if using one server
  - May cause severe degradation if system is deployed to thousands of servers.
- Configuration errors are common during installation and can lead to network effects. i.e. routing a message through an intermediary rather than directly will introduce latency.
- In Chapter 10 we saw the Latency Monkey and the Chaos Monkey as testing techniques for running systems.
- Network effects are best found through self-aware systems, i.e. system monitors itself and makes values available externally.



Test Activities

- The architect should be actively involved in
  - Test planning, since the architect knows the sensitive areas of the system.
  - Test development. Test driven development is a technique where the next increment of the system is developed to satisfy a predetermined test.
  - Test interpretation. The architect knows what various monitored values should be and is best equipped to interpret test results.
  - Test harness creation. The test harness has to intimately interact with the system and this requires architecture knowledge.



Summary

- Implementation
  - Implementation activities can embed architecture knowledge in the code
  - Templates can be used for critical sections that reoccur
  - Architecture erosion can be prevented through use of tools and management processes
- Testing
  - Unit and integration tests depend on architectural knowledge and a test harness.
  - Network effects are difficult to discover when deploying a system to 1000s of servers.
  - The architect should be involved in a wide variety of test activities.