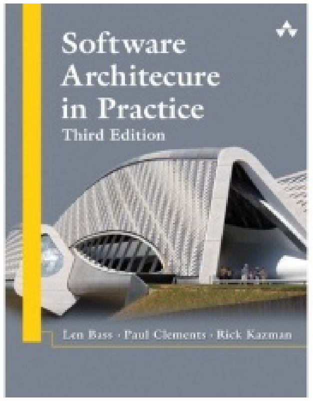
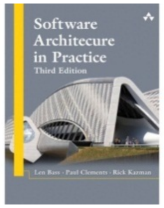
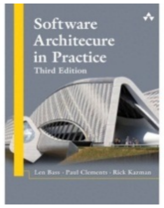

Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

Chapter 17: Designing an Architecture



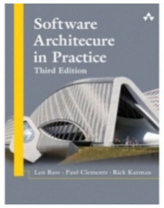
Chapter Outline

- Design Strategy
- The Attribute-Driven Design Method
- The Steps of ADD
- Summary


Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

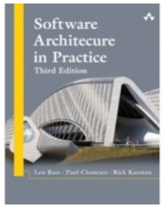
Design Strategy

- Decomposition
- Designing to Architecturally Significant Requirements
- Generate and Test



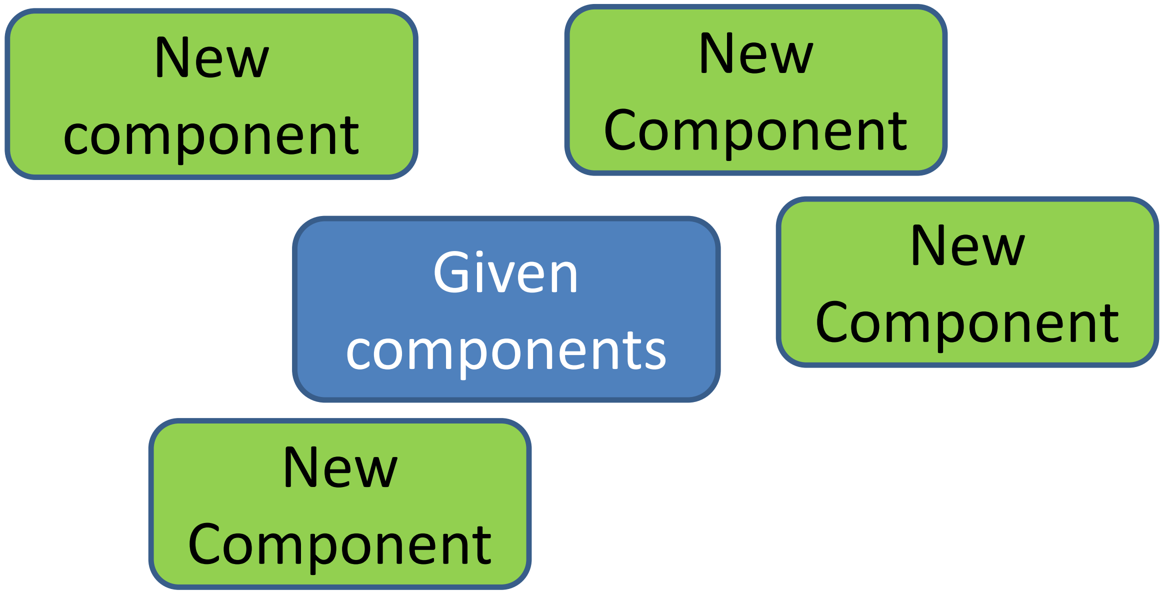
Decomposition

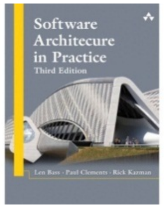
- Architecture determines quality attributes
- Important quality attributes are characteristics of the *whole* system.
- Design therefore begins with the whole system
  - The whole system is decomposed into parts
  - Each part may inherit all of part of the quality attribute requirements from the whole



Design Doesn’t Mean Green Field

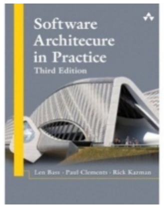
- If you are given components to be used in the final design, then the decomposition must accommodate those components.





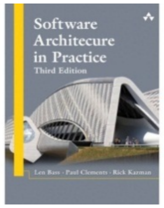
Designing to Architecturally Significant Requirements

- Remember architecturally significant requirements (ASRs)?
- These are the requirements that you must satisfy with the design
  - There are a small number of these
  - They are the most important (by definition)



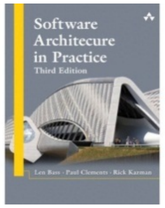
How Many ASRs Simultaneously?

- If you are inexperienced in design then design for the ASRs one at a time beginning with the most important.
- As you gain experience, you will be able to design for multiple ASRs simultaneously.


Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

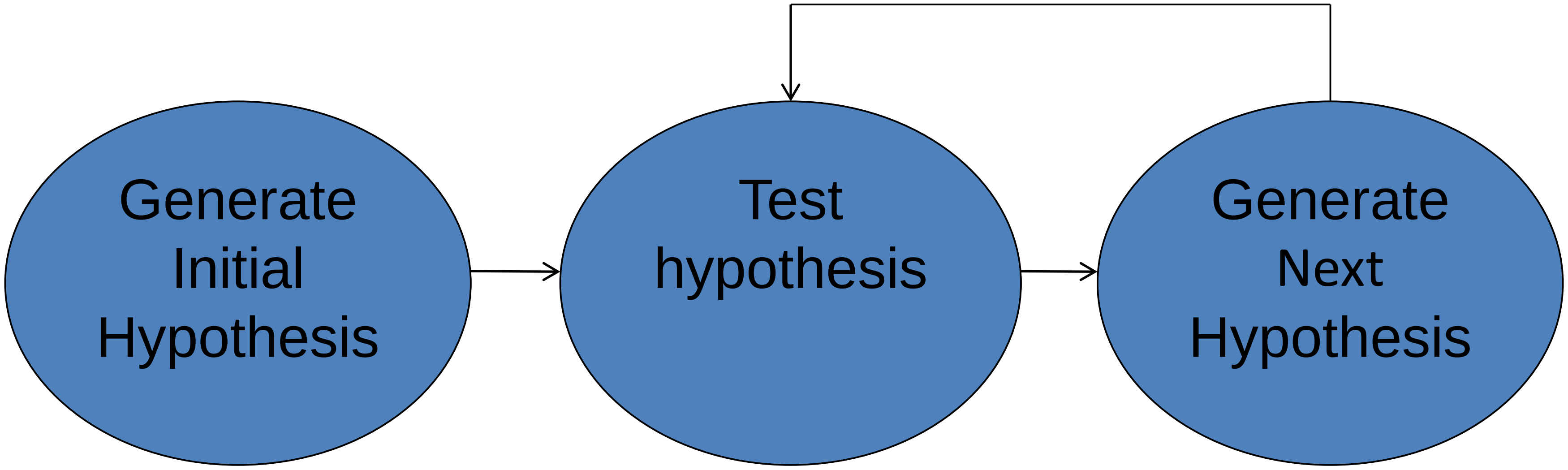
What About Other Quality Requirements?

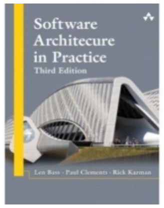
- If your design does not satisfy a particular non ASR quality requirement then either
  - Adjust your design so that the ASRs are still satisfied and so is this additional requirement or
  - Weaken the additional requirement so that it can be satisfied either by the current design or by a modification of the current design or
  - Change the priorities so that the one not satisfied becomes one of the ASRs or
  - Declare the additional requirement non-satisfiable in conjunction with the ASRs.



Generate and Test

- View the current design as a hypothesis.
- Ask whether the current design satisfies the requirements (test)
- If not, then generate a new hypothesis

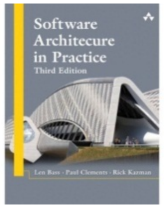




Raises the Following Questions

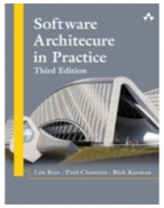
- Where does initial hypothesis come from?
- How do I test a hypothesis?
- When am I done?
- How do I generate the next hypothesis?

- You already know most of the answers; it is just a matter of organizing your knowledge.


Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

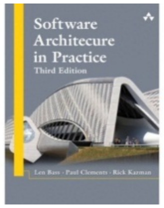
Where Does the Initial Hypothesis Come From?

- Desirable sources
  - Existing systems
  - Frameworks
- Less desirable sources
  - Patterns and tactics
  - Domain decomposition
  - Design checklists
- Why “less desirable”?
  - The less desirable ones do not cover all of the requirements. They typically omit many of the quality attribute requirements.



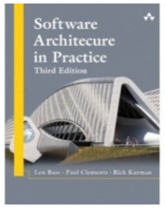
How Do I Test a Hypothesis?

- Use the analysis techniques already covered
- Design checklists from quality attribute discussion.
- Architecturally significant requirements
- What is the output of the tests?
  - List of requirements – either responsibilities or quality – not met by current design.



How Do I Generate the Next Hypothesis?

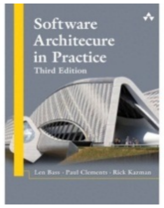
- Add missing responsibilities.
- Use tactics to adjust quality attribute behavior of hypothesis.
  - The choice of tactics will depend on which quality attribute requirements are not met.
  - Be mindful of the side effects of a tactic.



When Am I Done?

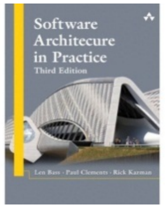
When…

- All ASRs are satisfied and/or…
- You run out of budget for design activity
  - In this case, use the best hypothesis so far.
  - Begin implementation
  - Continue with the design effort although it will now be constrained by implementation choices.

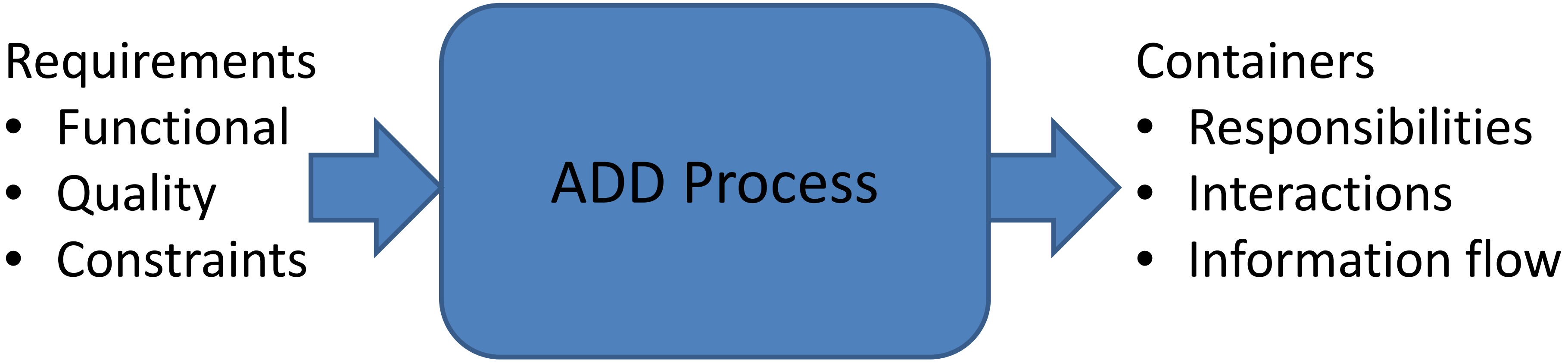


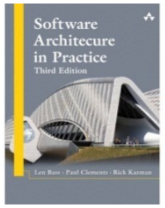
The Attribute-Driven Design Method

- Packaging of many of the techniques already discussed.
- An iterative method. At each iteration you
  - Choose a part of the system to design.
  - Marshal all the architecturally significant requirements for that part.
  - Generate and test a design for that part.
- ADD does not result in a complete design
  - Set of containers with responsibilities
  - Interactions and information flow among containers
- Does not produce an API or signature for containers.



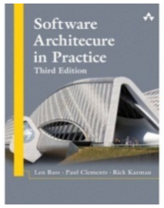
ADD Inputs and Outputs




Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

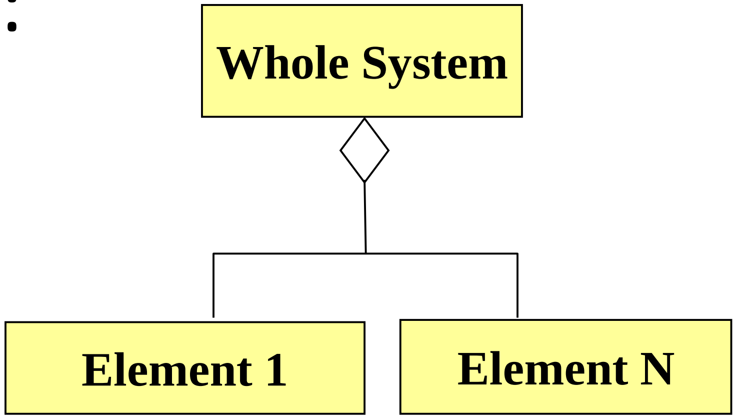
The Steps of ADD

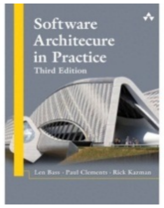
1. Choose an element of the system to design.
2. Identify the ASRs for the chosen element.
3. Generate a design solution for the chosen element.
4. Inventory remaining requirements and select the input for the next iteration.
5. Repeat steps 1–4 until all the ASRs have been satisfied.



Step 1: Choose an Element of the System to Design

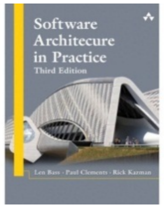
- For green field designs, the element chosen is usually the whole system.
- For legacy designs, the element is the portion to be added.
- After the first iteration:





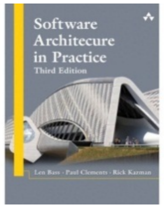
Which Element Comes Next?

- Two basic refinement strategies:
  - Breadth first
  - Depth first
- Which one to choose?
  - It depends ☺
- If using new technology => depth first: explore the implications of using that technology.
- If a team needs work => depth first: generate requirements for that team.
- Otherwise => breadth first.



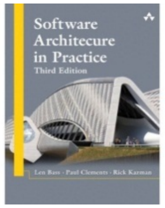
Step 2: Identify the ASRs for the Chosen Element

- If the chosen element is the whole system, then use a utility tree (as described earlier).
- If the chosen element is further down the decomposition tree, then generate a utility tree from the requirements for that element.



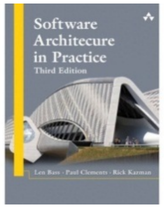
Step 3: Generate a Design Solution for the Chosen Element

- Apply generate and test to the chosen element with its ASRs



Step 4: Select the Input for the Next Iteration

- For each functional requirement
  - Ensure that requirement has been satisfied.
  - If not, then add responsibilities to satisfy the requirement.
    - Add them to container with similar requirements
    - If no such container may need to create new one or add to container with dissimilar responsibilities (coherence)
    - If container has too many requirements for a team, split it into two portions. Try to achieve loose coupling when splitting.



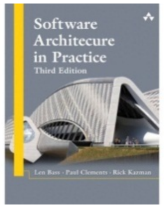
Quality Attribute Requirements

If the quality attribute requirement has been satisfied, it does not need to be further considered.

If the quality attribute requirement has not been satisfied then either

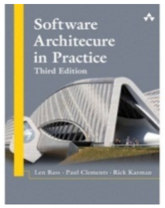
- Delegate it to one of the child elements
- Split it among the child elements

If the quality attribute cannot be satisfied, see if it can be weakened. If it cannot be satisfied or weakened then it cannot be met.



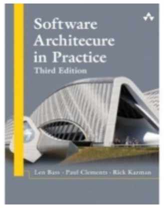
Constraints

- Constraints are treated as quality attribute requirements have been treated.
  - Satisfied
  - Delegated
  - Split
  - Unsatisfiable



Repeat Steps 1–4 Until All ASRs are Satisfied

- At end of step 3, each child element will have associated with it a set of:
  - functional requirements,
  - quality attribute requirements, and
  - constraints.
- This sets up the child element for the next iteration of the method.


Software Architecure in Practice
Third Edition
Len Bass · Paul Clements · Rick Kazman

Summary

- Designing the architecture is a matter of
  - Determining the ASRs
  - Performing generate and test one an element to decompose it to satisfy the ASRs
  - Iterating until requirements are satisfied.