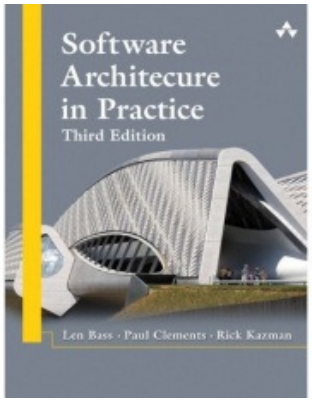
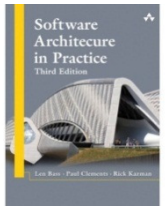
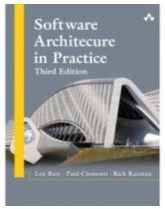


Chapter 7: Modifiability



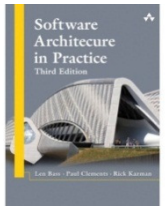
Chapter Outline

- What is Modifiability?
- Modifiability General Scenario
- Tactics for Modifiability
- A Design Checklist for Modifiability
- Summary



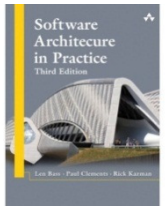
What is Modifiability?

- Modifiability is about change and our interest in it is in the cost and risk of making changes.
- To plan for modifiability, an architect has to consider three questions:
  - What can change?
  - What is the likelihood of the change?
  - When is the change made and who makes it?



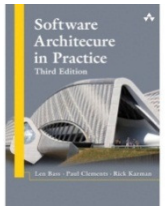
Modifiability General Scenario

Portion of Scenario	**Possible Values**
Source	End user, developer, system administrator
Stimulus	A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology
Artifacts	Code, data, interfaces, components, resources, configurations, …
Environment	Runtime, compile time, build time, initiation time, design time
Response	One or more of the following: • make modification • test modification • deploy modification
Response Measure	Cost in terms of: • number, size, complexity of affected artifacts • effort • calendar time • money (direct outlay or opportunity cost) • extent to which this modification affects other functions or quality attributes • new defects introduced



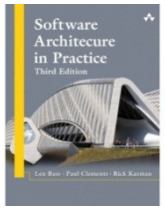
Sample Concrete Modifiability Scenario

- The developer wishes to change the user interface by modifying the code at design time. The modifications are made with no side effects within three hours.

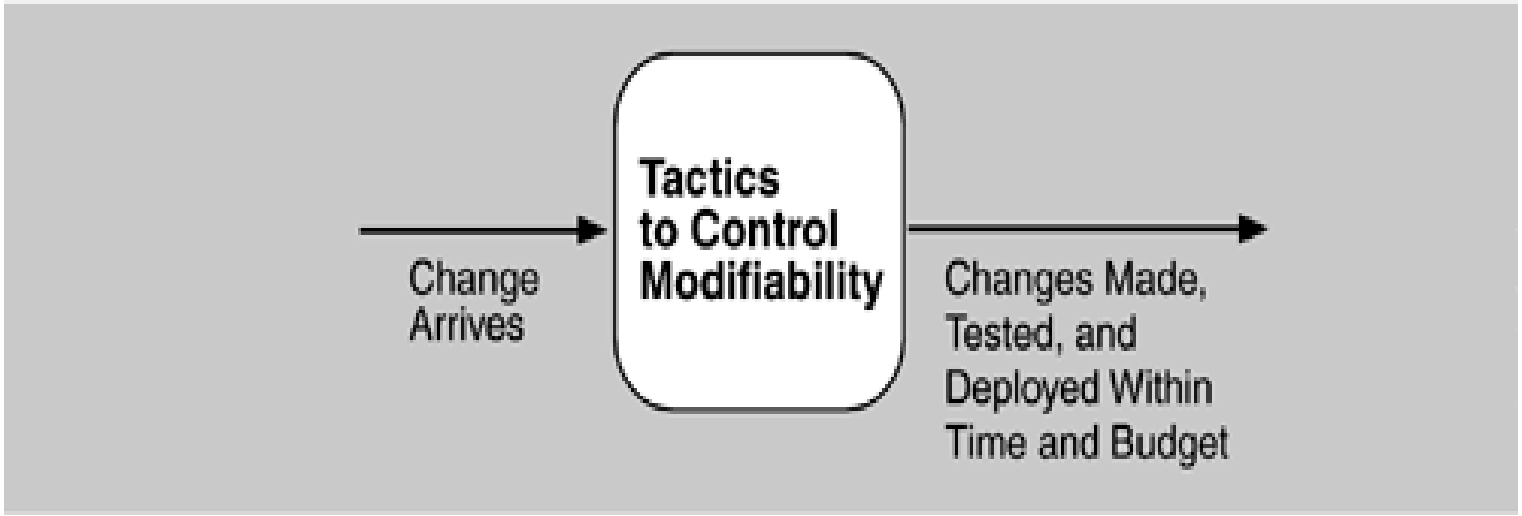


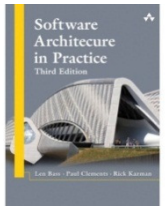
Goal of Modifiability Tactics

- Tactics to control modifiability have as their goal controlling the complexity of making changes, as well as the time and cost to make changes.

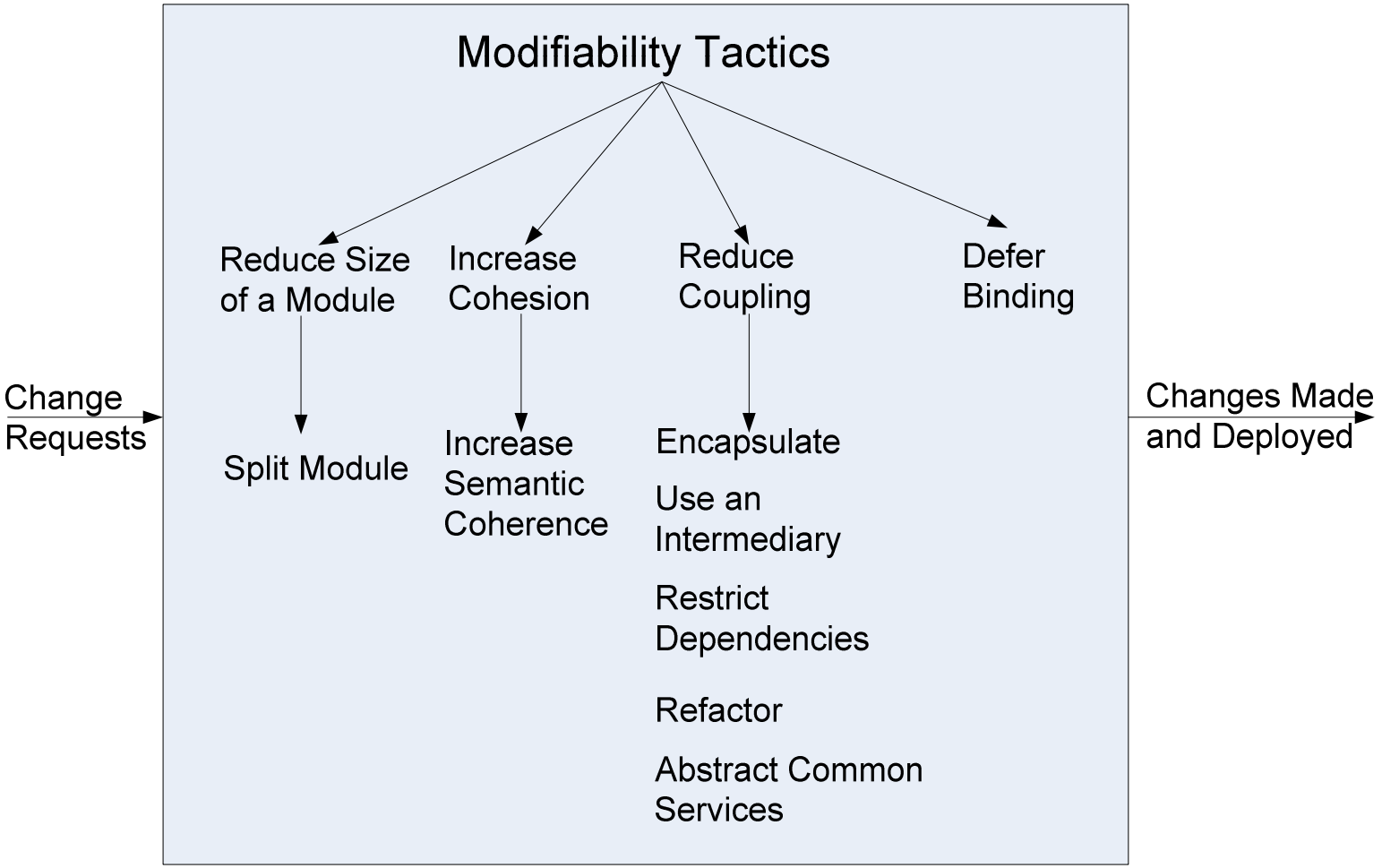


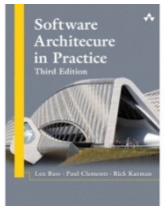
Goal of Modifiability Tactics





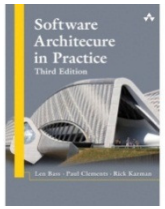
Modifiability Tactics





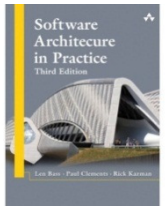
Reduce Size of a Module

- Split Module: If the module being modified includes a great deal of capability, the modification costs will likely be high. Refining the module into several smaller modules should reduce the average cost of future changes.



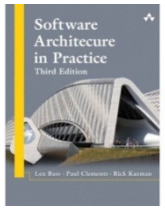
Increase Cohesion

- Increase Semantic Coherence: If the responsibilities A and B in a module do not serve the same purpose, they should be placed in different modules. This may involve creating a new module or it may involve moving a responsibility to an existing module.



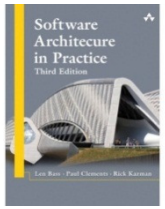
Reduce Coupling

- Encapsulate: Encapsulation introduces an explicit interface to a module. This interface includes an API and its associated responsibilities, such as “perform a syntactic transformation on an input parameter to an internal representation.”
- Use an Intermediary: Given a dependency between responsibility A and responsibility B (for example, carrying out A first requires carrying out B), the dependency can be broken by using an intermediary.



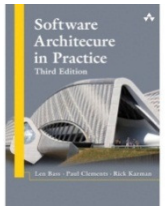
Reduce Coupling

- Restrict Dependencies: restricts the modules which a given module interacts with or depends on.
- Refactor: undertaken when two modules are affected by the same change because they are (at least partial) duplicates of each other.
- Abstract Common Services: where two modules provide not-quite-the-same but similar services, it may be cost-effective to implement the services just once in a more general (abstract) form.



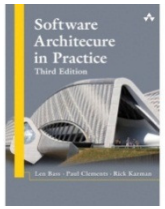
Defer Binding

- In general, the later in the life cycle we can bind values, the better.
- If we design artifacts with built-in flexibility, then exercising that flexibility is usually cheaper than hand-coding a specific change.
- However, putting the mechanisms in place to facilitate that late binding tends to be more expensive.



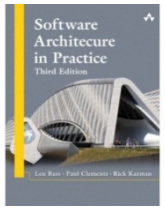
Design Checklist for Modifiability

Allocation of Responsibilities	**Determine which changes or categories of changes are likely to occur through consideration of changes in technical, legal, social, business, and customer forces. For each potential change or category of changes** . **Determine the responsibilities that would need to be added, modified, or deleted to make the change.** . **Determine what other responsibilities are impacted by the change.** . **Determine an allocation of responsibilities to modules that places, as much as possible, responsibilities that will be changed (or impacted by the change) together in the same module, and places responsibilities that will be changed at different times in separate modules.**



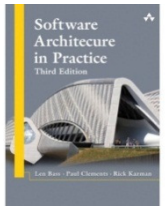
Design Checklist for Modifiability

Coordination Model	**Determine which functionality or quality attribute can change at runtime and how this affects coordination; for example, will the information being communicated change at run-time, or will the communication protocol change at run-time? If so, ensure that such changes affect a small number set of modules.** **Determine which devices, protocols, and communication paths used for coordination are likely to change. For those devices, protocols, and communication paths, ensure that the impact of changes will be limited to a small set of modules.** **For those elements for which modifiability is a concern, use a coordination model that reduces coupling such as publish/subscribe, defers bindings such as enterprise service bus, or restricts dependencies such as broadcast.**



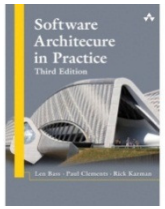
Design Checklist for Modifiability

Data Model	**Determine which changes (or categories of changes) to the data abstractions, their operations, or their properties are likely to occur. Also determine which changes or categories of changes to these data abstractions will involve their creation, initialization, persistence, manipulation, translation, or destruction.** **For each change or category of change, determine if the changes will be made by an end user, system administrator, or developer. For those changes made by an end user or administrator, ensure that the necessary attributes are visible to that user and that the user has the correct privileges to modify the data, its operations, or its properties.** **For each potential change or category of change** • **determine which data abstractions need to be added, modified, or deleted** • **determine whether there would be any changes to the creation, initialization, persistence, manipulation, translation, or destruction of these data abstractions** • **determine which other data abstractions are impacted by the change. For these additional abstractions, determine whether the impact would be on their operations, properties, creation, initialization, persistence, manipulation, translation, or destruction.** • **ensure an allocation of data abstractions that minimizes the number and severity of modifications to the abstractions by the potential changes** **Design your data model so that items allocated to each element of the data model are likely to change together.**



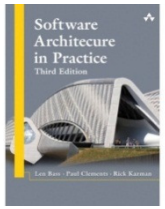
Design Checklist for Modifiability

Mapping Among Architectural Elements	**Determine if it is desirable to change the way in which functionality is mapped to computational elements (e.g. processes, threads, processors) at runtime, compile time, design time, or build time.** **Determine the extent of modifications necessary to accommodate the addition, deletion, or modification of a function or a quality attribute. This might involve a determination of, for example:** • **execution dependencies** • **assignment of data to databases** • **assignment of runtime elements to processes, threads, or processors** **Ensure that such changes are performed with mechanisms that utilize deferred binding of mapping decisions.**



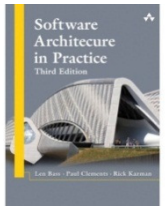
Design Checklist for Modifiability

Resource Management	Determine how the addition, deletion, or modification of a responsibility or quality attribute will affect resource usage. This involves, for example, . determining what changes might introduce new resources or remove old ones or affect existing resource usage. . determining what resource limits will change and how Ensure that the resources after the modification are sufficient to meet the system requirements. Encapsulate all resource managers and ensure that the policies implemented by those resource managers utilize are themselves encapsulated and bindings are deferred to the extent possible.



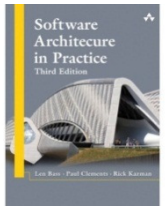
Design Checklist for Modifiability

Binding Time	**For each change or category of change** • **Determine the latest time at which the change will need to be made.** • **Choose a defer-binding mechanism (see Section 7.2.4) that delivers the appropriate capability at the time chosen.** • **Determine the cost of introducing the mechanism and the cost of making changes using the chosen mechanism** • **Do not introduce so many binding choices that change is impeded because the dependencies among the choices are complex and unknown.**



Design Checklist for Modifiability

Choice of Technology	**Determine what modifications are made easier or harder by your technology choices.** • **Will your technology choices help to make, test, and deploy modifications?** • **How easy is it to modify your choice of technologies itself (in case some of these technologies change or become obsolete)?** **Choose your technologies to support the most likely modifications. For example, an Enterprise Service Bus makes it easier to change how elements are connected but may introduce vendor lock in.**



Summary

- Modifiability deals with change and the cost in time or money of making a change, including the extent to which this modification affects other functions or quality attributes.
- Tactics to reduce the cost of making a change include making modules smaller, increasing cohesion, and reducing coupling. Deferring binding will also reduce the cost of making a change.