# Discussion Slides



## Deeper

- What does it mean for an architecture to be stable and robust?
  - *The basic definitions of the functionality and interface remain stable as we elaborate the design of each component, and as component and system requirements evolve.*
- Why is this important?
  - *Component design changes have limited impact. If the architecture changes, we may have to redo work in many components.*



## Deeper

- How could a bad architecture result in unbuildable components?
  - *By apportioning responsibilities in a way that makes a task difficult to perform.*
  - *By requiring them to perform functions without access to required inputs.*
  - *By requiring them to control events without giving them sufficient ability to influence system behavior.*
  - *By requiring them to implement functions that we do not know how to implement.*



## Deeper

- Give examples of troublesome inter-component dependencies.
  - *Interfaces that expose implementation details, and are likely to change over time.*
  - *Interfaces that share internal state between components.*
  - *Interfaces that make the client responsible for ensuring resource integrity.*
  - *Interfaces that require interactions rather than providing simple services.*



## Deeper

- In a hierarchical system, how do I know what is a system component and what is a design detail?
  - *The answer will be different at different levels of the design.*
  - *If something is a component (at the current level) seeing inside it would not lead you to desire or expect different interfaces or behavior.*
  - *This is also a function of how well abstracted the interfaces are.*



## Deeper

- How could architecture become more specific, without becoming design?
  - *General statements of functionality, become high level interface summaries, which eventually become complete interface definitions.*
- What would be an example of a not-yet-specific design?
  - *General designs will be replaced by, routine specifications, each of which will eventually have detailed algorithm descriptions.*

## Deeper

- How can architecture solve hard problems?
  - *By apportioning responsibilities in such a way that the problems cease to be difficult.*
    - *e.g. avoid distributed consensus issues by centralizing local proxies for remote objects in a single context.*
    - *e.g. improve resource integrity by making its container smarter.*
    - *e.g. avoid complex error recovery by implementing goal-seeking with fall-back to a safe state.*



## Deeper

- How does architecture drive performance and robustness
  - *Performance is driven by the number of operations that must be performed, and the number of components that are involved in each operation.*
  - *Robustness is determined by the ability of a component to detect errors and respond appropriately.*
  - *These are characteristics of the interfaces between the components.*



## Deeper

- How does architecture drive the division of tasks among groups?
  - *Component development can be effectively divided among multiple groups if the components are highly independent from one another.*
  - *Component coupling is driven by the definitions of component responsibilities, and the interfaces between them.*



## Deeper

- How does architecture drive the order of component development?
  - *Component A may depend on component B in ways that make it difficult to develop component A until component B is already available.*
  - *The interdependencies of components are determined by the architecture.*



## Deeper

- How does architecture determine the manner in which components can be tested?
  - *Most component testing is done through their defined interfaces. If those interfaces are designed with an eye towards automated testing, it will be much easier to test those components.*
  - *The interface specifications may also determine how easy it is to simulate other components for testing purposes.*



## Deeper

- How does architecture determine the manner in which components will be integrated?
  - *If the architecture defines components that are easily stubbed and interfaces that are easily simulated, it may be possible to build an entire system before any of the components are completed. This will enable easy incremental integration of subsequent changes in small steps.*

## Deeper

- How does architecture affect the manner in which problems are diagnosed?
  - *Well designed interfaces make it possible to inspect intermediate results at component boundaries.*
  - *Well designed interfaces enable good error checking at component boundaries.*
  - *Such capabilities greatly enhance the ease of tracking down a problem.*



## Deeper

- Why does modularity lead to naturally hierarchical design?
  - *A key attribute of hierarchical design is that it is possible to navigate the design by zooming in on individual components.*
  - *If there is minimal coupling between components, then the design of each component can be elaborated and understood independently of other components.*



## Deeper

- How does modularity enable parallel development?
  - *Good modularity implies simple, well abstracted interfaces between components, with good information hiding and minimal coupling of their designs.*
  - *If there is minimal coupling between components, it should be possible to design, build, and perhaps even test each component independently from the others.*



## Deeper

- How does modularity lead to easier design, coding, and testing?
  - *Cohesion (doing only a few things in each module) leads to simpler specifications.*
  - *Simpler specifications, combined with low coupling (minimal interactions with other modules) means that the designer has to consider fewer things.*
  - *Simpler code is easier to write, and more likely to be correct.*
  - *Simpler specifications and simpler code mean fewer test cases.*



## Deeper

- How does modularity lead to most changes involving only a single module?
  - *Cohesion means that all of the code related to a particular class or operation is within a single module.*
- Why should modularity lead to fewer unintended consequences of changes?
  - *Low inter-module coupling means that non-interface changes to one module are unlikely to affect other modules.*



## Deeper

- How can we know what functionality a client needs or whether a proposed interface would be convenient?
  - *We have to put ourselves in the client's position, and consider what we know and what our goals are. We might need input from someone with domain experience.*
  - *This might be difficult in pure top-down design, which is why McConnell says that design is iterative and combines both top-down and bottom-up.*

## Deeper

- Why would a well abstracted interface be simpler for the client?
  a) *this is part of the definition of well abstracted.*
  b) *usually, when we expose implementation details to a client, we force (or invite) them to write their code in a way that conforms to our implementation. A better abstracted view of the services would probably impose less burden on our clients.*



## Deeper

- How does a better abstracted interface provide more flexibility for the developer?
  *An interface that has been decoupled from its implementation is less likely to impose troublesome constraints on future implementations.*
  *This means it should be relatively easy to make any necessary future implementation changes while preserving the original interfaces.*



## Deeper

- What would happen if we unilaterally decided to change an interface contract?
  *We could break clients who depended on the old interface.*
  *We might no longer be able to integrate or interoperate with programs that honored the old interface.*
  *We might upset a lot of customers, and cause a lot of support problems.*



## Deeper

- What does mechanism/policy separation imply about the interfaces to a service?
  *There will probably be external interfaces for dynamic setting of policy preferences, or plugging in policy implementations.*
- How do you know what things might be construed as policy?
  *Anyplace where a decision involves choice (there is no obviously correct answer)*