Module 11 – the Attribute Driven Design Method

Recall Generate and Test - 1

- Design is the combination of process of generate and test and making decisions.
- The initial design is generated from existing or similar systems, frameworks and components, or patterns.
- Early design decisions inform the initial hypothesis
- The design is tested against the architecturally significant requirements and a collection of quality attribute test cases to derive additional responsibilities and constraints on responsibilities.

Recall Generate and Test - 2

- The design is analyzed against quality attribute models to discover shortcomings.
- Tactics are used to propose alternatives for improving the design
- The next hypothesis is generated based on additional responsibilities and constraints discovered during test and analysis.

Attribute-Driven Design (ADD) Method

- The ADD method is a packaging of Generate and Test into a method with defined steps.
- It follows a recursive decomposition process where, at each stage in the decomposition, tactics and architectural patterns are chosen to satisfy a set of quality attribute scenarios.

Architecture Created by ADD

- Is a representation of the high-level design choices
- Describes a system as
  - containers for functionality
  - interactions among containers
- Is the first articulation of the architecture during the design process and is necessarily coarse-grained
- Is critical for achieving desired quality and business goals, and providing the framework for achieving functionality

Input into ADD

- Functional requirements – expressed as use cases
- Quality attribute requirements – expressed as quality attribute scenarios
- Constraints - design decisions made as a portion of the requirements, e.g. use web based solution, use Android OS.

Steps of the ADD Method

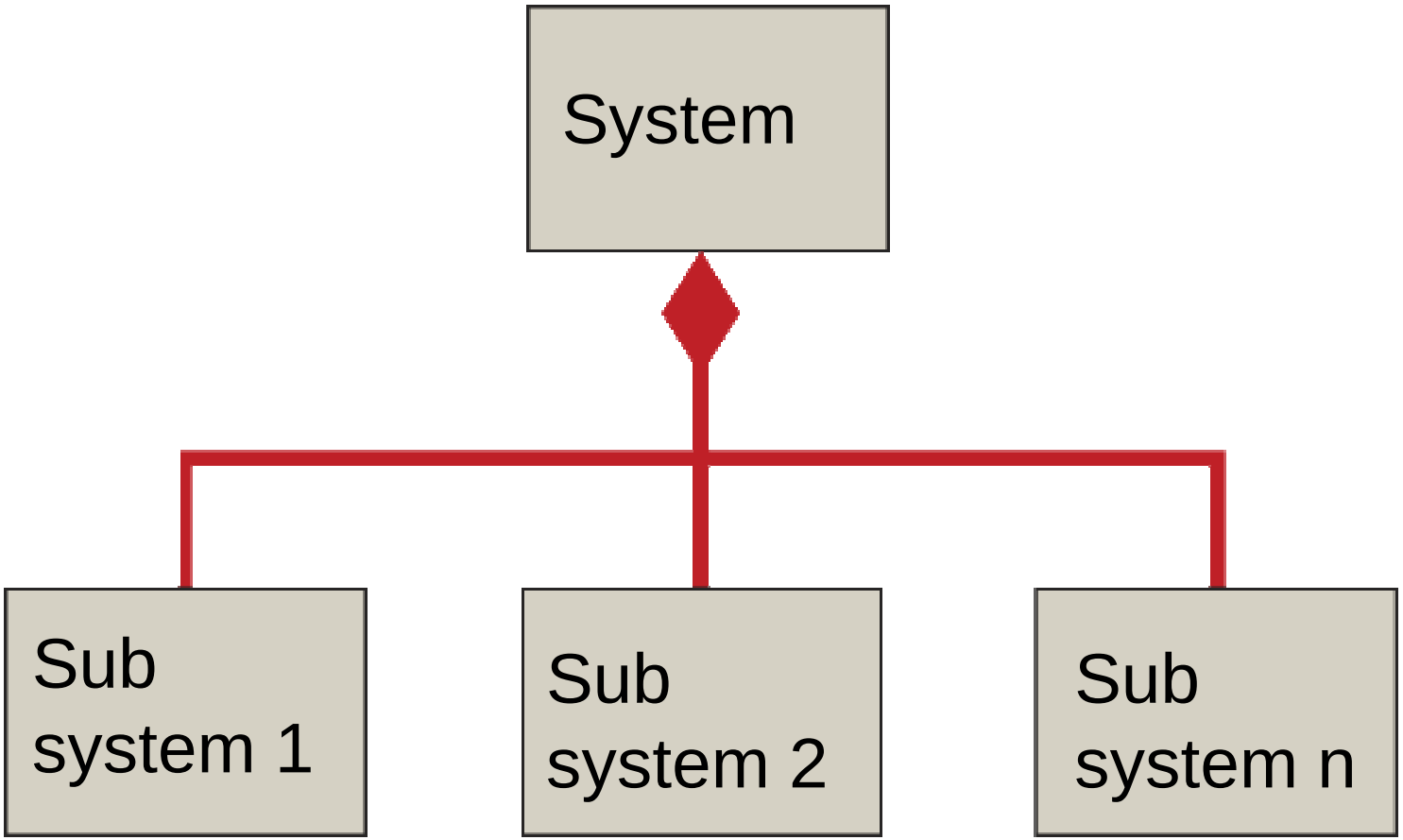
1. **Choose an element of the system to decompose**.
2. Identify architecturally significant requirements.
3. Determine architectural solution for architectural drivers and important use cases (Generate and Test)
4. Verify and refine requirements and make them constraints for instantiated elements.
5. Repeat these steps for the next element.

Step 2: Choose an Element - 1

- ADD is a decomposition method:
  - Typically, it starts with a system.
  - The system is then decomposed further into elements and sub-elements.
- In ADD, we call all of these parts elements.

We call the chosen element the parent element. The parent element defines the context in which the remaining steps are applied.

Decomposition tree



Step 2: Choose an Element - 2

- The order of the decomposition tree will vary based on the
  - *business context* – the specific skills to be fully utilized.
    It argues for depth-first decomposition.
  - *domain knowledge* – having no major unknowns.
    It argues for breadth-first decomposition.
  - *new technology* – The need for prototypes argues for depth-first decomposition.

Steps of the ADD Method

1. Choose an element of the system to decompose.
2. **Identify architectural significant requirements.**
3. Determine architectural solution for architectural drivers and important use cases (Generate and Test)
4. Verify and refine requirements and make them constraints for instantiated elements.
5. Repeat these steps for the next element.

Architecturally Significant Requirements

- Some requirements are more influential than others on the architecture and the decomposition of each element.
- Influential requirements can be
  - functional (e.g., training crews in flight simulator)
  - quality attribute related (e.g., high security)
  - business oriented (e.g., product line)
- Architecturally Significant Requirements (ASRs) are the combination of functional, quality attribute, and business requirements that “shape” the architecture or the particular element under consideration.

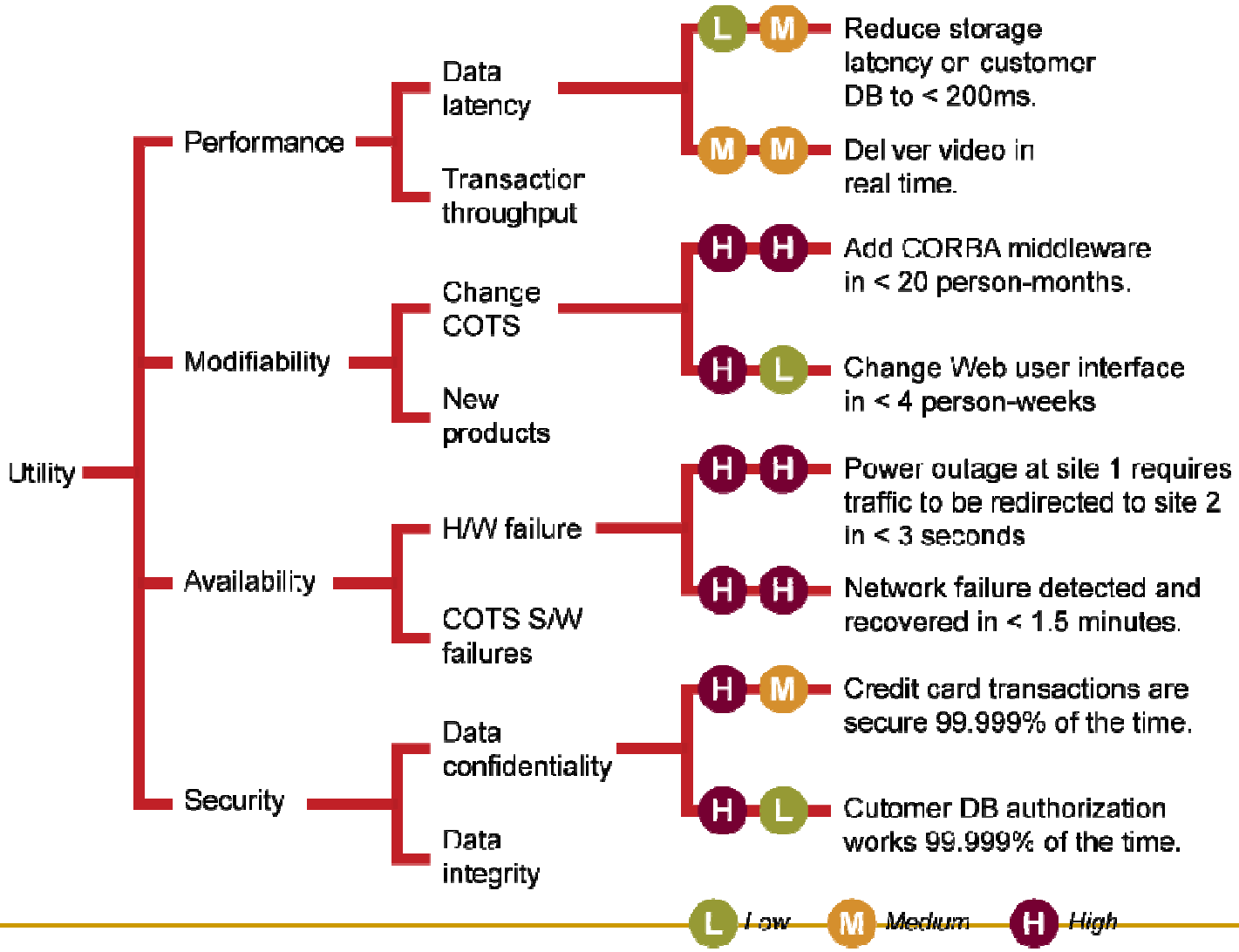
Quality Attribute Requirements: Utility Tree

- Quality attribute utility trees provide a mechanism for translating the business drivers of a system into concrete quality attribute scenarios.
- A utility tree shows the prioritization of quality attribute requirements, realized as scenarios. The utility tree serves to make concrete the quality attribute requirements, forcing architect and customer representatives to define relevant quality attributes precisely.

Quality Attribute Requirements – Utility Tree – Ratings

- Each scenario is rated high, medium, or low in two dimenstions
  - Business importance
  - Architectural importance
- Those scenarios rated high in importance and high in difficulty provide the most critical context against which the architecture can be analyzed. These scenarios are candidates for the ASRs.

Utility Tree - example



Architecturally Significant Requirements

- Those entries in the utility tree that are labeled H,H
- Should be relatively few of these – less than 10.
- If more than 10, then maybe assignment of labels may be incorrect.

Steps of the ADD Method

1. Choose an element of the system to decompose.
2. Identify architectural significant requirements.
3. **Determine architectural solution for architectural drivers and important use cases (Generate and Test)**
4. Verify and refine requirements and make them constraints for instantiated elements.
5. Repeat these steps for the next element.

Step 4: Determine architectural solution

- The goal of this step is to establish an architectural solution that satisfies the architectural significant requirements and the important use cases.
- Apply generate and test where the requirements are the architecturally significant requirements and the important use cases

Generate and test

- Initial hypothesis
- Test
- Next hypothesis

Document the Design

- Module views are useful for reasoning about and documenting the non-runtime properties of a system.
- Component and connector views are useful for reasoning about and documenting the runtime properties of a system.
- Allocation views are useful for reasoning about and documenting the relationships between software and non-software.

Result of this step

- Architecture decomposition for the chosen element that satisfies architecturally significant requirements and important use cases.
- Not done – what about other requirements?

Steps of the ADD Method

1. Choose an element of the system to decompose.
2. Identify architectural significant requirements.
3. Determine architectural solution for architectural drivers and important use cases (Generate and Test)
4. **Verify and refine requirements and make them constraints for instantiated elements.**
5. Repeat these steps for the next element.

Step 4: Verify and refine requirements

- Consider use cases and quality attribute requirements not involved in defining current hypothesis.
- For each use case, ensure that responsibilities to satisfy use case have been allocated to one of the children of the module being decomposed
- The responsibilities for a given element are grouped together. The rationale for grouping could be
  - functional coherence for modifiability
  - similar timing behavior for performance
  - secure/insecure areas for security
  - and so forth…

Exercising use cases

- In the process of considering use cases, you may discover
  - new responsibilities
  - new element types
  - new instances of element types

Create Additional Instances

- Create additional instances of element types in the following circumstances
  - There is a difference in the quality attribute properties of responsibilities assigned to an element.
  - Additional elements must be instantiated to achieve other quality attribute requirements.

Verify Allocation of Requirements

- If the requirements of the parent element are satisfied, nothing more needs to be done.
- Otherwise, requirements can be satisfied in one of the following ways:
  - The requirement is delegated to a child. Satisfying the requirement depends on the decomposition of the child.
    - more needs to be done when the child is decomposed
  - The requirement is delegated to multiple children.
    - the solution involves the cooperation of the children
    - requirement satisfaction becomes a global problem from the perspective of each involved child

Constraints on Child Design Elements

- Some constraints come from the parent element in the form of requirements.
- Some constraints are placed on child design elements
as the result of making design decisions.
- Additional constraints are placed on child design elements as quality attribute scenarios are refined.

Refine Quality Attribute Requirements - 1

- Quality attribute scenarios have to be refined and assigned to the child elements.
- A given quality attribute scenario might be satisfied by current decomposition without additional impact.
- A given quality attribute scenario might be satisfied by current decomposition with constraints on child elements and might involve:
  - individual child elements or
  - the coordination of individual child elements

Refine Quality Attribute Requirements - 2

- Decomposition might be neutral with respect to a given quality attribute scenario.
  - Assign the scenario to a child element. If all children are equal with respect to the scenario, assign it to an arbitrary child element.
- A given quality attribute scenario might not be satisfied by the current decomposition.
  - If it is a high-priority scenario, reconsider the current decomposition.
  - Otherwise you must justify not meeting the requirement

The Next Iteration of the Steps

- At the end of this step, a collection of child elements will exist. Each element will have
  - a collection of responsibilities
  - an interface
  - functionality
  - quality attribute scenarios
  - constraints
- The above items are the input for the next iteration of the ADD method steps.

ADD Summary

- Determine architecturally significant requirements
- Apply generate and test to these requirements
- Adjust the resulting design to account for other constraints and requirements.
- Represent solution in the module, concurrency, and deployment views.
- ADD is repeated recursively.

Exercise

- What are architecturally significant requirements for a smart phone?
- What is initial hypothesis for a design to satisfy those requirements?
- What requirements are unmet by initial hypothesis?