Review of ENSE 621/ENPM 641: Part 03

*Methods of Systems Engineering Development*

Mark Austin

E-mail: austin@isr.umd.edu

Institute for Systems Research, University of Maryland, College Park
Strategies of Systems Engineering Development:

Topics:

1. Top-down and bottom-up development.
2. Guidelines for design decomposition.
3. Established strategies of development.
4. Vertical and Horizontal Integration.
5. Key Steps in ENSE 621.
6. Extensions for ENSE 622.
Top-Down and Bottom-Up Development

Top-down design (decomposition)

- NEW PROBLEM
- DECOMPOSITION
- SUB-PROBLEMS

Bottom-up development (synthesis)

- INDEPENDENT MODULES
- COMPOSITION
- COUPLED MODULES
Advantages/Disadvantages of Top-Down Decomposition

- Can customize a design to provide what is needed and no more.
- Decomposition simplifies development – lower-level (sub-system) development may only require input from a single discipline.
- Start from scratch implies slow time-to-market.

Advantages/Disadvantages of Bottom-up Development

- Reuse of components enables fast time-to-market.
- Reuse of components improves quality because components will have already been tested.
- Design may contain (many) features that are not needed.
Example 1. Layered development and organization of requirements at NASA Goddard.

Level 0 — Mission Objective.

Level 1 — Science Requirements

Level 2 — System-level engineering requirements

Level 3 — Sub-system requirements

Requirements are organized into clusters for team development.
Example 2. Top-down decomposition and bottom-up synthesis coupled to reuse of objects/sub-systems.
Guidelines for Design Decomposition

Guidelines for the design of modules are:

1. One module should have no more than seven subordinate modules.
2. There should be separation between the controller modules and the worker modules.
3. Every module must perform a task appropriate to its place in the hierarchy.
4. Modules should only receive as much information as they need.

The motivation for following these guidelines is modules that will be functional, easy to understand, testable, and reusable.
Guidelines for Design Decomposition

Coupling

Coupling is a measure of the interface complexity (or degree of interdependence) between modules.

In design, we should:

1. Keep the interfaces as minimal as possible;
2. Keep the interfaces as simple as possible, when in fact, one must exist.
Guidelines for Design Decomposition

Cohesion

Cohesion is a measure of how well the components of a module are related to one another (put another way, cohesion is a measure of the functional association of the element within an element).

In design, we should:

1. Keep related functions together;
2. Keep unrelated functions apart.

Coupling and cohesion work together

Generally speaking, ...

...modules with components that are well related will have the capability of plugging into loosely coupled systems.
Module Complexity

Modules should be kept as simple as possible, and hide the details of implementation from the outside environment.

Figure 1: Elements of module complexity.

Three factors that contribute to module complexity are: (a) its size, (b) the number of internal functions and connections within the modules, and (c) the number of interfaces to the modules.
Strategy 1. Simplify Design through Separation of Concerns

Complex systems are often characterized by ...

... many components, intertwined network structures, concurrent behaviors, and complicated communications and interactions among subsystems and components.

To facilitate understanding of these design issues/concerns, we aim to ...

... pull a design apart and examine it from perspectives (or ”facets” or viewpoints) that are almost orthogonal, thereby factoring out so-called cross-cutting concerns.

Achieving (almost) orthogonality of concerns is important because ...

... it means we can explore options in one viewpoint (or dimension of design) without affecting other concerns.
Example 1. Separation of concerns (e.g., structure, behavior, communication) in simple network.
Example 2. Synthesis of models for engineering system and surrounding environment.

- What events happen in the surrounding environment?
- What does the system do?
- How is the system built?
- What are the objects in the system?
- How are the objects organized?

---

**Structure**
- How is the system built?
- What are the objects in the system?
- How are the objects organized?

---

**Behavior**
- What does the system do?
- How will the system respond to unusual events?

---

**Environmental Model**

- What events happen in the surrounding environment?
- How will the environment and system interact?
- What about uncertainties in the environment?

---

**Structure**
- How is the surrounding environment built?
- What objects are in the environment?
- How are the objects organized?
Example 3. Separation of SE activities/products – requirements, design and validation results.
**Strategy 2. Function before Physical**

We promote the description of systems in two orthogonal ways:

- The function that the systems is intended to provide,
- Candidate architectures for realizing the functionality.

**Function-Architecture Co-Design**

Map models of system behavior onto system structure alternatives.

Identify measures of effectiveness. Then evaluate and rank design alternatives.
Benefits of Function-Architecture Co-Design

Lessons learned in industry indicate that a clean separation of system functionality and system architecture enables:

1. Much easier modifications of the design at the system level,
2. More effective exploration of alternative solutions.
3. Reuse of major components.
Strategy 3. Layered Approach to Development

The tenet of “breadth before depth” leads to a layered approach to development.
Goals of Systems Integration

System integration involves ...

... joining existing disparate services or systems together into a single view or process for the user.

Since many of the participating subsystems will have well-defined interfaces,

... integration involves joining the subsystems together by gluing their interfaces together.

The interfaces may be operational, logical, or both.

When the interfaces don’t directly interlock, then ...

... the glue between them can provide the required mappings.
Vertical Systems Integration

Vertical integration takes places when ...

... one business integrates with another business at a different stage in the chain of production.

Usually the goal is to pull together the scattered parts of a product or process – both upstream and downstream – and ...

... stack them up under the same roof.

Strategically linking knowledge and production in one place can ...

... eliminate waste in time and resources, create better communication, and develop more efficient processes.
Vertical and Horizontal Integration

Integration of Processes into Vertical Silos

No Integration

End Customers
Distribution
Assembly
Intermediate Manufacturing
Raw Materials

Vertical Integration

End Customers
Distribution
Assembly
Intermediate Manufacturing
Raw Materials

Integration at Apple

End Customers
Distribution
Assembly
Intermediate Manufacturing
Raw Materials
Horizontal Systems Integration

Horizontal integration takes places when ...

... one business integrates with another business at the same stage in the chain of production.

Horizontal integration can mean ...

... movement towards a monopoly within an industry.

Simple Example: Integration of Restaurant Chains
**Problem Definition.** Development of an Operations Concept.
Pathway from goals and scenarios to simplified models of behavior and requirements.

- Use Case Diagram
- Activity Diagrams
- Sequence Diagrams
- Use Case 1
  - scenario 1
  - scenario 2
- Use Case 2
  - scenario 3
  - scenario 4
- Individual Use Cases and Scenarios
- High-Level Requirements:
  - Req 1.
  - Req 2.

Sequence of messages between objects.
Models of System Behavior and System Structure.
Key Points:

- The functional description dictates what the system must do.
  Here, we employ a combination of use cases (and use case diagrams), textual scenarios, and activity and sequence diagrams to elicit and represent the required system functionality.

- A complete system description will also include statements on minimum levels of acceptable performance and maximum cost.
  Since a system does not actually exist at this point, these aspects of the problem description will be written as design requirements/constraints.

- Further design requirements/constraints will be obtained from the structure and communication of objects in the models for system functionality (e.g., required system interfaces).
Problem Solution. Pathway from Requirements to Models of System Behavior/Structure and System Design

1. Goals and Scenarios
   - Operations Concept
   - System Behavior
     - Performance Attributes
   - Traceability via use cases.
   - Traceability
   - Mapping
   - Iteration strategy to satisfy constraints.
   - Detailed description of the system’s capabilities.

2. Project Requirements
   - System Structure
     - Objects and Attributes
   - System Design
   - System Evaluation
     - System Specification
   - Traceability
   - Selection of System Architecture

Problem Domain

Solution Domain
Key Points:

- Requirements are organized according to the role they will play in the system-level design.
- Models of behavior specify what the system will actually do.
- Models of structure specify how the system will accomplish its purpose.
- The nature of each object/subsystem will be captured by its attributes. Attributes includes:
  - The attributes of the physical structure of the design,
  - The attributes of the environmental elements that will interact the the system.
  - Attributes of the system inputs and system outputs
- We create the system-level design by mapping fragments of system functionality/behavior onto specific subsystems/objects in the system structure.
Increased Emphasis on using SysML

Taxonomy of diagrams in SysML (Source: Adapted from SysML tutorial)

Use Multi-Scale Approaches to System Modeling

Key Points:

- Semi-Formal Models: High-level view of the complete system (efficiency).
- Formal Models: Detailed view of the actual system (accuracy).
Behavior Modeling for Real-Time Systems

Generic framework for development of active objects.
Behavior Modeling for Real-Time Systems

Schematic for modeling of active object behavior.

Thread of execution for active object behavior ....
Support for Top-Down and Bottom-Up Architecture-Level Development

- Activity Diagrams
- Sequence Diagrams
- System Behavior Task Graphs.
- Allocate behavior to components
- Develop Class Hierarchies
- Selection of alternatives from database
- System Architecture
- Identify bottlenecks...
- Schedule Table for System Evaluation
- Database of Components

Req 1.
Req 2.

High-Level Requirements.

Iterative strategy to satisfy constraints.

Allocate behavior to components

Allocate behavior to components

Develop Class Hierarchies

Add class hierarchies to database

Selection of alternatives from database

- p. 30/32
SysML/Simulation Support for System Evaluation and Improvement

Architecture selection, system design and evaluation (Source: Petru Elis, Linkoping, Sweden).

System Behavior (task graph) → Communications buss. → System Structure

Iterative strategy to satisfy constraints.

Schedule Table for System Evaluation

Identify bottlenecks....
Support for Platform-Based Design

Platform-based design methodologies ...

... improve the efficiency, correctness and economics of design by restricting the space of design options to pre-defined components, connectors, and rules for assembly (all contained in a library).

Flowchart of Platform-Based Design Activities