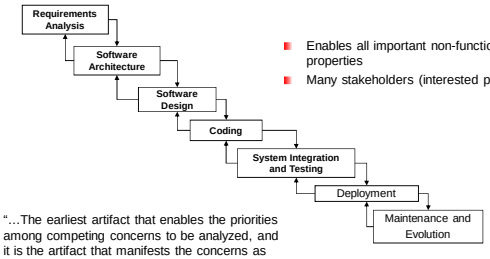
OMSE 532 - Week 2
A Case Study in Architectures the A-7E

Dr. Stuart Faulk
Computer and Information Science
University of Oregon

Overview

- Lecture 1 high points: relating architectural influences to architectural structures
- A-7E case study

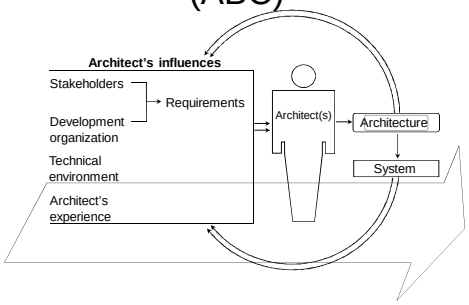
Early Design Decisions about Key Qualities



- Enables all important non-functional properties
- Many stakeholders (interested parties)

"…The earliest artifact that enables the priorities among competing concerns to be analyzed, and it is the artifact that manifests the concerns as system qualities."

Architecture Business Cycle (ABC)



Architectural Feedback

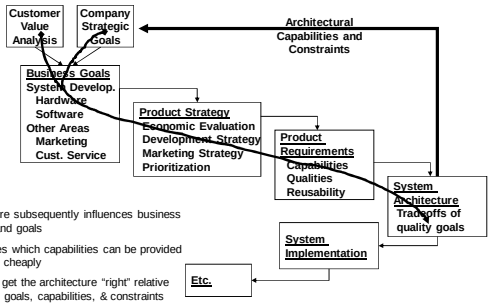
- Architecture influences the things that influence it
- Architecture influences organization
  - Influences organizational structure by work breakdown
  - As an organizational resource, may affect business goals
- May influence customer requirements since it affects the cost and speed of subsequent development
- Act of construction influence the architect's experience
- Occasionally influences the technical environment

Why Study Architecture in SE?

Definition: The *purpose of software engineering* is to gain and maintain intellectual & managerial control over the products and processes of software development.

- Need for engineering of software architecture
  - Profoundly affects system and business qualities
  - Requires making tradeoffs
  - Control means achieving system qualities by choice not chance
    - Understanding what the tradeoffs are
    - Understanding the consequences of each choice
    - Making appropriate choices at appropriate times
  - Applies to all of the things influenced by architecture!

The Business Cycle



- Architecture subsequently influences business decisions and goals
- Determines which capabilities can be provided quickly and cheaply
- Goal is to get the architecture "right" relative to business goals, capabilities, & constraints

Architectural Design in Context

Methodology independent view

1. Understand the business case for the system
2. Specify the requirements
   1. Specify behavioral and developmental requirements the architecture must support
   2. Identify stakeholder priorities
3. Design the architecture (understanding effects)
4. Perform detailed design and implementation
5. Verify and validate the system
   1. Is it consistent with the architectural specification
   2. Does it meet business goals

Design the Architecture*

View design as the process of building in the required system properties

1. Must identify and define the design goals and priorities
2. Choose appropriate components and relations for the purpose
3. Choose appropriate representations to communicate design decisions & tradeoffs to the stakeholders
4. Choose and apply an effective design approach
   - Methods/principles to systematically achieve desired properties
5. Establish and metrics of "goodness" relative to purpose

*We will see this repeatedly!

What gets Built?

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them."

From *Software Architecture in Practice,* Bass, Clements, Kazman

Architectural Structures

- To describe one, we must answer:
  - What are the components? The relations? The interfaces?
  - To design one, must also know what it is good for - i.e., what quality attributes are affected by that structure.
- E.g., Calls Structure
  - Components: procedures
  - Relation: calls or invokes
  - Interface: procedure interface (parameters) and visible effects
  - Useful for: execution flow (scenarios)
- Commonalties
  - Each portrays some aspects of the system, abstracts from others
  - Chosen to support design of some system attribute (or related set)
  - Exactly what is visible and what is abstracted away depends on what attributes must be engineered

Architecture of the A-7E

A-7E Corsair II Aircraft

- U. S. carrier-based, light attack aircraft
- Used from the 1960s through the 1980s
- Small computer on board for navigation, weapons delivery

Behavioral Requirements

- Read data from sensors such as
  - Air probe, forward-looking radar, Doppler radar, inertial measurement set
  - Pilot controls (cockpit switches)
- Control weapons
- Manage cockpit displays
  - Heads-up display (HUD)
  - Moving map display, keypad and alpha-numeric display, warning lights, cockpit dials

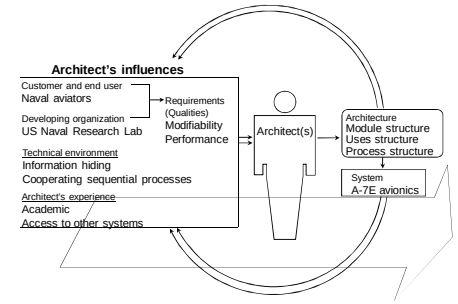
Behavioral Requirements (2)

- Software was responsible for
  - Computing real-world values (such as position or altitude)
  - Navigating by providing pilot with current location in any of 18 different navigation modes
  - Computing ballistic weapon solutions
    - 100 different weapon types
    - 21 different delivery modes

Quality Requirements

- The weapons and navigation calculations had to be performed 25 times per second on a very slow computer (25 MHz)
- The entire program still had to fit in 32K
- The program had to be extremely modifiable
  - Constant equipment, weapons changes
  - Frequently added requirements
  - Long life-span

ABC for the A-7E (NRL)



Based on CMU/SEI tutorial for Software Architecture in Practice

Why Study the A-7E?

- Recall basic concepts:
  - Architectural view describes system components and their connectors
  - Any real system has many possible architectural views
  - A given architecture is created to meet particular system goals or requirements
- But, for most systems (even academic ones)…
  - The architectures are byproducts of code design
  - Different kinds of architectural styles and concerns are mixed together such that they cannot be studied separately
  - Clear architectural goals are never defined
  - Principles of construction are *implicit* and *inconsistently applied*
  - No architecture is ever clearly and completely documented

 18

Why Study the A-7E? (2)

- A-7E is a good "model" system…
  - The architectural views are well defined and distinct in purpose
  - Purpose and objectives of each architectural view are *clearly stated in terms of system qualities*
  - Principles of construction are clearly stated and applied with daunting consistency
  - The architectures were relatively well documented
- But it's not the only system…
  - Good illustration of principles of architectural design
  - Specific architectures and design approach may or may not be appropriate for other systems/domains

Architectures of the A-7E OFP

- Module structure
  - Specifies the *design-time* decomposition of the program into work assignments
  - Components are modules
  - Relation is "implements secrets of"
  - Useful for: separating concerns, ease of change, work breakdown
- Uses structure
  - Specification of the inter-program dependencies
  - Components are programs
  - Relation is "requires the presence of"
  - Useful for: subsetting, incremental development
- Process structure (not today)
  - Decomposition of the run time structure
  - Components are processes (threads)
  - Relation is "synchronizes with"
  - Useful for: separation of concerns, concurrent execution

Architectural Design Elements (Revisit for Each Structure)

- Design goals
  - What are we trying to accomplish in the decomposition?
- Structure
  - What are the components, relations, interfaces?
- Method of communication
  - How are the choices of components, relations and interfaces communicated to the stakeholders?
- Decomposition principles
  - What decomposition principles support the objectives?
  - What do I do next?
  - Am I done yet?
- Evaluation criteria
  - How do I tell a good design from a bad one?

A-7E Architectures

- Module Structure
- Uses Structure
- Process Structure

Module Structure Design Goals

- Divide software into set of work assignments with the following properties:
  - *Easy to Understand:* Each module's structure should be simple enough that it can be understood fully
  - *Easy to Change (mutability):* It should be possible to change the implementation of one module without knowledge of or affecting the implementation of other modules
  - *Proportion:* Effort of making a change should be in (reasonably) direct proportion to the likelihood of that change being necessary
  - *Independence:* It should be possible to make a major change as a set of independent changes to individual modules

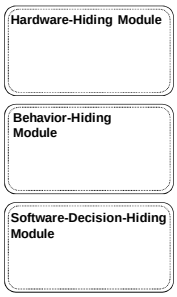
Modular Structure

- Components
  - Called modules
  - Leaf modules are work assignments
  - Non-leaf modules are the union of their submodules
- Relations
  - submodule-of => implements-secrets-of
  - The union of all submodules of a non-terminal module must implement all of the parent module's secrets
  - Constrained to be acyclic tree (hierarchy)
- Interfaces (externally visible component behavior)
  - Defined in terms of access procedures (services or method)
  - Only way to access internal state

Decomposition Criteria

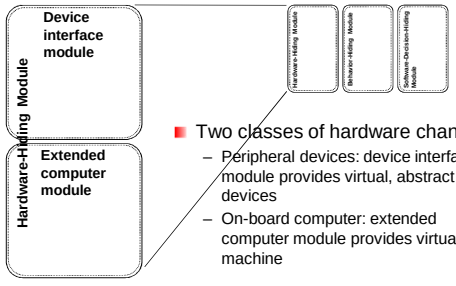
- Principle: information hiding
  - System details that are likely to change independently should be encapsulated in different modules.
  - The interface of a module reveals only those aspects considered unlikely to change.
- What do I do next?
  - For each module, determine if its secret contains information that is likely to change independently
- Stopping criteria
  - Each module is simple enough to be understood fully
  - Each module is small enough that it makes sense to throw it away rather than re-do.

A-7E Module Decomposition



- Hardware-Hiding (HH)
  - Secret = properties of physical hardware
  - Encapsulates any hardware changes
- Behavior-Hiding (BH)
  - Secret = algorithms/data addressing requirements
  - Encapsulates requirements changes
- Software Decision (SD)
  - Secret = decisions by designer
  - Encapsulates internal design decisions

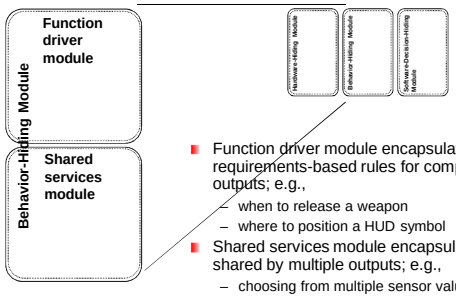
Hardware-Hiding Modules



- Two classes of hardware change
  - Peripheral devices: device interface module provides virtual, abstract devices
  - On-board computer: extended computer module provides virtual machine

Based on CMU/SEI tutorial for Software Architecture in Practice

Behavior-Hiding Module

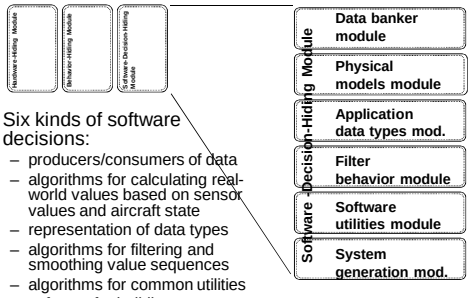


- Function driver module encapsulates requirements-based rules for computing outputs; e.g.,
  - when to release a weapon
  - where to position a HUD symbol
- Shared services module encapsulates rules shared by multiple outputs; e.g.,
  - choosing from multiple sensor values
  - current system modes

Based on CMU/SEI tutorial for Software Architecture in Practice

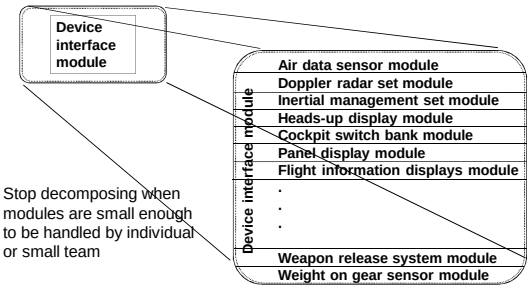
Based on CMU/SEI tutorial for Software Architecture in Practice

Software Decision-Hiding Module



- Six kinds of software decisions:
  - producers/consumers of data
  - algorithms for calculating real-world values based on sensor values and aircraft state
  - representation of data types
  - algorithms for filtering and smoothing value sequences
  - algorithms for common utilities
  - software for building system

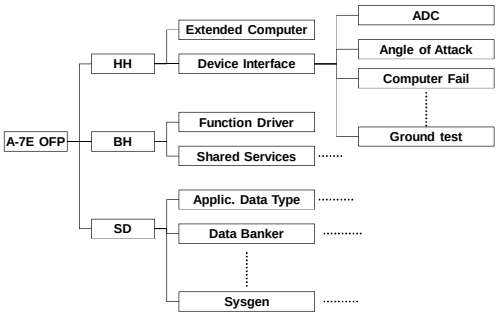
Example of Third-Level Modules



- Stop decomposing when modules are small enough to be handled by individual or small team

Based on CMU/SEI tutorial for Software Architecture in Practice

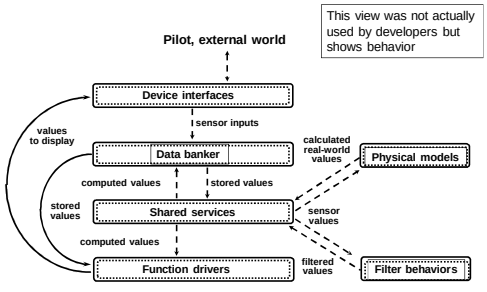
A-7E Module Structure



How Modules Work Together

- Function driver module produces output values on schedule
  - Asking data banker module for current data
  - Asking physical models module to calculate real-world values
  - Computing output values
  - Telling device interface module to send values to output devices
- Data banker is updated by
  - Device interface module, with sensor values
  - Shared services module, with current mode, best sensor choice, other data

Data Flow View



Based on CMU/SEI tutorial for Software Architecture in Practice

Method of Communication

Module Guide

- Documents the module *structure*:
  - The set of modules
  - The responsibility of each module in terms of the module's secret
  - The "submodule-of relationship"
  - The rationale for design decisions
- Document purpose(s)
  - Orientation for new team members
  - Guide for finding the module responsible for some aspect of the system behavior
    - Where to find or put information
    - Determine where changes must occur
  - Baseline design document
  - Provides a record of design decisions (rationale)

Method of Communication (2)

Module Interface Specifications

- Documents all assumptions user's can make about a (leaf) module's externally visible behavior
  - Access programs, events, types, undesired events
  - Design issues, assumptions
- Document purpose(s)
  - Provide all the information needed to write a module's programs or use the programs on a module's interface
  - Specify required behavior by fully specifying behavior of the module's access programs

Evaluation Criteria

- Completeness
  - Is every aspect of the system the responsibility of one module?
  - Do the submodules of each module partition its secrets?
- Ease change
  - Is each likely change hidden by some module?
  - Are only aspects of the system that are very unlikely to change embedded in the module structure?
  - For each leaf module, are the module's secrets revealed by it's access programs?
- Usability
  - For any given change, can the appropriate module be found using the module guide?
  - …etc.

A-7E Architectures

- Module Structure
- Uses Structure
- Process Structure

Uses Structure

- Definition of uses
  - Procedure A uses procedure B if a correctly functioning procedure B must be present for A to meet its requirements.
  - Intuitively: Any system with A in it must also have B if A is to work correctly.
- Uses structure
  - Components are procedures (e.g., access programs)
  - Design time relation is "allowed to use" (constraints on "uses")

Design Goals

- Easily produce a subset/extension of system
  - Meet schedule demands
  - Provide variations on a family
- Integration and testing: build test subsets to fully test top program
- Support virtual machines: build layers to support portability and domain specific abstractions

Definition of Uses Relation

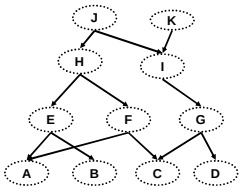
- Units of this structure are programs.
- Program A uses program B if a correctly functioning B must be present for A to meet its requirements.
- Similar to calls, but not quite the same
  - A might call B, but not use it (e.g., if B is an exception handler, A's correctness does not depend on anything that B computes).
  - A might use B even if it doesn't call it (e.g., assumption that B has left some computed value in an accessible place).

Decomposition Rules

- Decomposition principles (Parnas): allow A to use B if
  - A is essentially simpler because it uses B
  - B is not substantially more complex because it is not allowed to use A
  - There is a useful subset containing B and not A
  - There is no useful subset containing A and not B
- Were made more design-specific for A-7E

Uses Hierarchy

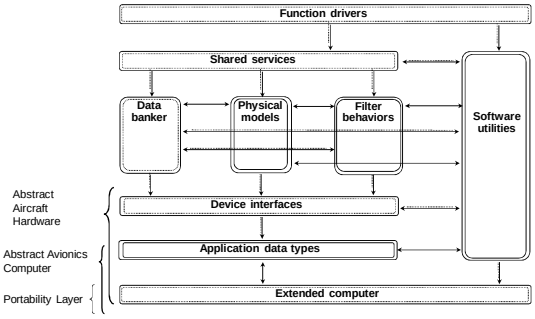
- "Ideal" design gives "loop-free" hierarchy with uses relation (acyclic tree)
- Level 0 uses nothing else
- Supports layers and subsets



A-7E Uses Rules (Simplified)

- Extended computer modules use no other programs
- Application data types programs use only extended computer programs
- Device interface programs can use extended computer programs, data types, filter behavior, and physical models programs
- Function driver and shared services programs can use data banker, physical models, filter behavior, device interface, extended computer, and application data types programs.

Layers Emerge from Uses Rules



Method of Communication

- Notionally an NxN matrix

USING PROGRAMS	USED PROGRAMS
Extended Computer Module	None
Device Interface Module	EC.DATA/PGM/IO*, AT.NUM/STE.2.5.1/STE.2.5.2, SU
Air Data Computer	PM.ECM, +DB.G_SS.Vertical_velocity_system+
Angle of Attack	
Audible Signal	
+S_BEEP_RATE+	EC.TIMER*
All others	EC.PAR*/SMPH
Computer Fail Signal	
Doppler Radar Set	
Flight Information Displays	DI.IOREP, DI.FLR
Forward Looking Radar	
+S_FLR_BLINK_RATE+	EC.TIMER*
All others	DI.IOREP, DB.DI.MFSW, EC.PAR*/SMPH
Head-Up Display	
+S_HUD_BLINK_RATE+	EC.TIMER*
All others	EC.PAR*/SMPH
Inertial Measurement Set	PM.ACM
Input/Output Rep'n	
Panel	EC.PAR*/SMPH
Radar Altimeter	
SINS	EC.PAR*/SMPH

Evaluation Criteria

- Loop free
- Consistent with required subsets
- Preserves with abstract machine layers

A-7E Lessons

- SAP cites three reasons architecture is important
  - Communication among stakeholders
  - Early design decisions (bind the right thing at the right time)
  - Transferable abstraction
- A-7E structures illustrate all three
  - The primary vehicle for communication with designers, programmers
  - Instantiated design goals
  - Reused many times (canonical decomposition)

Questions?