Case Study in Architectural Structures:

A-7E Avionics System - A
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
Background of Old System

• Fit into 32K of memory

• Written in Assembler

• Had to be optimized for efficiency

• Brittle, hard to modify
ABC for the A-7E II

Architect’s influences
- Customer and end user
- Naval aviators
- Developing organization
- US Naval Research Lab
- Technical environment
- Information hiding
- Cooperating sequential processes
- Architect’s experience
- Academic
- Access to other systems

Requirements (Qualities)
- Modifiability
- Performance

Architect(s)

System
- A-7E avionics
Behavioral Requirements -1

- Data in from
  - sensors
  - pilot controls

- Data out to
  - cockpit display
  - weapon release hardware
Behavioral Requirements -2

• Read data from sensors such as
  – air probe
  – aim able forward-looking radar
  – Doppler radar
  – inertial measurement set
  – type of weapon loaded on each wing station
  – dozens of switches in the cockpit
Behavioral Requirements -3

• Release wing-mounted weapons.

• Manage cockpit displays such as
  – heads-up display (HUD)
  – moving map display
  – keypad and alpha-numeric display
  – warning lights
  – cockpit dials
• The pilot could communicate the location of a ground target to the software by
  – moving a cursor on the HUD over it, and pressing a “designate” button
  – moving a cursor on the map over it, and pressing a “designate” button
  – manually aiming the forward-looking radar to the point and pressing the “designate” button
  – entering its latitude/longitude via the keypad
Behavioral Requirements -5

• Software was responsible for
  – computing real-world values (such as position or altitude) by choosing the best currently-available sensor(s) and performing appropriate integration/smoothing
  – 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
  – releasing the chosen weapon at the right time
Quality Requirements

• The behavioral requirements were easy to satisfy compared to these quality requirements.
  – The weapons and navigation calculations had to be performed 25 times per second on a very slow computer.
  – The entire program still had to fit in 32K.
  – The program had to be extremely modifiable because this was a demonstration of information hiding as a design discipline.
Architectural Approach

• Concentrated on three architectural structures
  – *module structure*
    • to achieve modifiability
    • to achieve flexibility of producing subsets
    • to allocate expertise
  – *uses structure*: to ease production of subsets
  – *process structure*: to achieve portability, performance tuning
A-7E Architectural Structures

• Module structure
  • Uses structure
  • Process structure
Principles for Creating Modules

• Units of this structure are modules (work assignments).

• Information hiding was the design principle.
  – identify areas of likely changes and assign a module to each
  – encapsulate the changeable aspects in the module’s implementation
  – build the constant aspects into the module’s interface
  – decree that all uses of the module occur via the facilities on its interface
  – hide data structures, algorithms, and other changeable aspects
Classifying Changes

• Three classes of change
  – hardware
    • new devices
    • new computer
  – required behavior
    • new functions
    • new rules of computing cockpit displays
    • new modes
  – software decisions
    • new ways to schedule processes
    • new ways to represent data types
    • new ways to keep data current
Three First-Level Modules (Work Assignments)

- Hardware-hiding module
- Behavior-hiding module
- Software-decision-hiding module
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
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.,
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
A-7E Module Structure (2 Levels)

Hardware-Hiding Module
- Device interface module
- Extended computer module

Behavior-Hiding Module
- Function driver module
- Shared services module

Software-Decision-Hiding Module
- Data banker module
- Physical models module
- Application data types mod.
- Filter behavior module
- Software utilities module
- System generation mod.
Example of Third-Level Modules

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

- Device interface module
  - Air data sensor module
    - Doppler radar set module
    - Inertial management set module
    - Heads-up display module
    - Cockpit switch bank module
    - Panel display module
    - Flight information displays module
    - .
    - .
    - Weapon release system module
  - Weight on gear sensor module
Module Structure As Team Structure

• Basis for team assignment: One team was formed for each second-level module.

• Basis for document organization: The bulk of the document corresponded to modules, although other structures were documented separately.
How Modules Work Together

• Function driver module produces output values at appropriate times by
  – 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

Pilot, external world

- Device interfaces
  - Device interfaces
  - Sensor inputs
  - Calculated real-world values
  - Sensor values

- Data banker
  - Data banker
  - Computed values
  - Stored values
  - Real-world values

- Shared services
  - Shared services
  - Computed values
  - Stored values

- Function drivers
  - Function drivers
  - Computed values
  - Filtered values

- Physical models
  - Physical models
  - Calculated real-world values
  - Sensor values

- Filter behaviors
  - Filter behaviors
  - Filtered values

Values to display
A-7E Architectural Structures

- Module structure
- *Uses structure*
- Process structure
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).
How Does Uses Structure Help Define Subsets?

• If program $A$ is to be included in subset $S$, then so must the transitive closure of $A$’s uses relation.
A-7E Uses Rules (Simplified)

- Extended computer module 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

- Function drivers
  - Shared services
    - Data banker
    - Physical models
    - Filter behaviors
  - Device interfaces
  - Application data types
  - Extended computer

- Software utilities
Layering

- Layering is a well-known style that can provide portability across computing platforms and quick reimplementations of applications.

- Layering is not clean. There are often “short-cuts” for performance or other reasons.

- The uses structure suggests a layering, but they are not interchangeable. The layered structure allows only a very restrictive definition of subsets.
A-7E Subset: Display HUD Altitude

Function drivers

Shared services

Data banker

Physical models

Filter behaviors

Device interfaces

Application data types

Extended computer

Software utilities

Data banker

Physical models

Filter behaviors

Device interfaces

Application data types

Extended computer

Software utilities
A-7E Architectural Structures

- Module structure
- Uses structure
  - Process structure
Process Structure -1

• Units of this structure are processes.

• In A-7E, processes resided in
  – function driver modules
    • periodic processes to continuously compute values
      (usually displays)
    • sporadic processes that take an action in response to
      events (e.g., release weapon)
  – value-computing modules, when timing required
    a value to be pre-computed and waiting
Process Structure -2

- The computer had only one processor.
- Off-line scheduling was used to build a schedule without the expense of a runtime executive.
- The primary relation was “synchronizes with” or “excludes” (in the case of using shared resources). The scheduler used these relations to generate a schedule.
- Processes could be merged by the scheduler for performance gains.
A-7E Process Structure

Function driver process

Function driver process

synchronizes with

Device interface process
Case Study Summary -1

• Three distinct structures (uses, module, and process) were used to design this system. Those structures (and others) are its architecture.

• Each structure was engineered to achieve particular quality attributes. Structures were not allowed simply to happen on their own.

• Information hiding was new and untested at the time of the A-7. It was shown to be a viable design strategy for building hard-real-time embedded computer software.
Case Study Summary -2

• Information hiding has come to be accepted as a standard structuring technique for software architectures.

• The requirement of fitting the program into 32K was not met.