Streamlined Flight Software Design

Steven A. Stolper
Senior Flight Software Engineer / Flight Software Group Supervisor
December 10, 2010
Agenda

- The Development Process
- Software Development Methodology
- Capturing the Software Requirements and Design
Defining Streamlined Development

- Streamlined Development is *not*
  - Defined by cost
  - Defined by schedule
  - Defined by project size or resources
Defining Streamlined Development

- Streamlined development *IS*
  - Applying state-of-the-art techniques to accelerate
    - Requirements analysis
    - Design Capture
    - Implementation
    - Integration and test
  - A set of strategies designed to produce high performance, highly-reliable flight systems
    - Less costly than “traditional” methods
    - Fewer engineers than “traditional” methods
    - More aggressive schedule than “traditional” projects
# The Development Process

## NASA Project life cycle

<table>
<thead>
<tr>
<th>Formulation</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pre-Phase A</strong></td>
<td></td>
</tr>
<tr>
<td>Concept Studies</td>
<td></td>
</tr>
<tr>
<td><strong>Phase A</strong></td>
<td></td>
</tr>
<tr>
<td>Preliminary Analysis</td>
<td></td>
</tr>
<tr>
<td><strong>Phase B</strong></td>
<td></td>
</tr>
<tr>
<td>Definition System Definition / Prelim Design</td>
<td></td>
</tr>
<tr>
<td><strong>Phase C</strong></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
</tr>
<tr>
<td><strong>Phase D</strong></td>
<td></td>
</tr>
<tr>
<td>Development</td>
<td></td>
</tr>
<tr>
<td><strong>Phase E</strong></td>
<td></td>
</tr>
<tr>
<td>Operations</td>
<td></td>
</tr>
<tr>
<td><strong>Phase F</strong></td>
<td></td>
</tr>
<tr>
<td>Disposal</td>
<td></td>
</tr>
</tbody>
</table>
The Development Process (cont)

- What is a Phase?
  - Products
  - Activities
  - Milestones
  - Quality Gates

- Examples?
The Development Process (cont)

**Pre-Phase A**
- **Program Development**
  - Major Products:
    - Mission Concept
    - Systems Concept
    - Strawman schedule, and budget
    - Project Proposals

**Phase A**
- **Requirements Analysis**
  - Major Products:
    - Mission requirements
    - Requirements on flight system, ground data system, support equipment, and operations

**Phase B**
- **Design**
  - Major Products:
    - Architecture Design
    - Source Selections
    - Concept verification/proof
    - Risk mitigation tests
    - High level design
    - Low Level design

**Phase C**
- **Implementation / Fabrication**
  - Major Products:
    - Flight, ground, and SE software and hardware
    - Unit tests
    - Small-scale integration
    - Early integration tests

**Phase D**
- **Integration**
  - Major Products:
    - Integrated components and checked-out system

**Phase E**
- **Test**
  - Major Products:
    - Completed tests:
      - Systems test
      - Operational tests
      - Environmental testing
      - Tested ops procedures

**Phase F**
- **Launch Operations**
  - Major Products:
    - Completed:
      - Arrival checkout
      - Final FSW load and params
      - Mate to LV
      - Launch

**Phase G**
- **Primary Mission**
  - Major Products:
    - Completed:
      - On-orbit checkout
      - Instrument checkout
      - TCM – if required
      - Scheduled operations
Do What Pays The Rent

“Never mistake activity for achievement.”

– John Wooden, UCLA Basketball Coach

Requirements:
- What does the system do?
- Apportionment of responsibilities? (To functional subsystems)
- Scenarios (Use cases, derived requirements)
- Architectural Approach

Design: Answer the five questions:
- What are the pieces?
- How do they fit?
- Where is the logic?
- Dynamic Behavior of the system?
- Tasking Model?
Flight Software Design Methodology

Mission Requirements -> Scenarios

Hardware Definition -> Context Diagram

Detailed Subsystem Requirements -> FSW Decomposition

- Architectural Definition
- Tasking Model

Problem Specifications

- Object Design
- Algorithms (Behavior model)
- Tasks
- Usage Model
- Interfaces

Design

PDR

Initial Implementation
Capturing the Design

- **Context Diagram**
  - Shows all of the hardware and software interfaces that FSW interacts with either directly or through other interfaces.

- **Decomposition Diagram**
  - Identifies subsystems of the flight software. Specifications to be written for each.

- **State Diagrams**
  - Show behavioral view of flight software for both mission-related activities and software-related activities.
The "Medusa" Diagram
This is the initial Functional Decomposition for the Pathfinder Flight Software. The arrows indicate direction of major data flow.

* Not to scale
Specification:
Name of Functional Subsystem

1. Requirements Sources

1.1 Interviewees:
Sources of information. Hallway conversations are valid sources of information.

1.2 Documents:
Existing documents used as sources of information.

2. Scope

2.1 Purpose:
What is this subsystem supposed to do? Why should it exist? How does it fit? How is it used by other parts of the system?

2.2 Inputs:
What information flows into the subsystem? Where will the information come from?

2.3 Outputs:
What information does the subsystem produce? Where will it go?

2.4 Responsibilities:
What is the subsystem role in the design? What services does it provide? What information does it maintain for the system? What requirements does it fulfill?

3. Functional Description

3.1 Narrative:
Give a high-level overview of the subsystem concept. Discuss the different aspects of how the subsystem is used by other parts of the software. Give a description of how the subsystem fulfills its system responsibilities.

3.2 Scenarios:
Enumerate all the usage scenarios for this subsystem. Provide a brief description of each including inputs, outputs, and execution characteristics (i.e. does not block caller, safe from interrupt level, failure scenarios...).

4. Requirements

4.1 Mandated Implementation:
List explicit, hard requirements. Elaborate if necessary.

4.2 Derived Requirements:
List requirements derived from explicit requirements or discovered in the course of gathering information for this document.

4.3 Performance:
List performance requirements. Non-blocking? Safe for use at interrupt level? Mutual exclusion requirements?

4.4 Interface:
List any requirements that affect the interface to other subsystems.

4.5 Other:
Catch other requirements here along with any information that the developer feels is important.
5. Assumptions

5.1 Other Software Components/Software Architecture:

What system-level assumptions were relied upon when the concept for this subsystem was formulated? What services and capabilities? What behavior is required from other parts of the software?

5.2 Operation:

Have assumptions been made about how the system will be operated?

5.3 Other:

Anything the developer is worried about.

6. Definitions

Any terminology unique to this subsystem.

7. Notes

Any additional information the developer believes is useful.
Capturing Functional Area Design

- **Object diagrams**
  - Identify “information” stored and methods (operations) that can be performed on each object

- **Collaboration (or ladder) diagrams**
  - Detailed identification of the interactions between related objects

- **Task Diagram**
  - Identifies tasks in subsystem

- **State Diagrams**
  - Show behavior view of the subsystem

- **Etc…**
Subsystem Context Diagram

Commands
- EHA_Set_RT_Pkt_Rate
- EHA_Set_REC_Pkt_Rate
- EHA_Set_Update_Pkt_Rate
- EHA_Set_Info
- EHA_Dump_Info
- EHA_Use_Measurement_Set
- EHA_Save_Measurement_Set

Timer Services
- Set/Cancel Repeating Timer

EH&A
- Time to Build Packet
- EHA_Store_Measurement

Measurement Producers
- Engineering Measurements

Downlink Subsystem
- EH&A Info Packet
- Fault Protection

Command Subsystem
- EH&A Info Packet

Engineering Packets (RT & REC)

Fault Protection
- Commands

Command Subsystem
- Commands

EH&A Info Packet
- Time to Build Packet

Set/Cancel Repeating Timer
Name of Object

Purpose:

What does this object do? Why does it exist? How is it used by other objects in the system?

Methods:

List each method and its parameters. Describe how each method behaves. Describe the parameters for each method and any internal data that is operated on.

Collaborations:

What other objects does this one interact with? What methods are invoked and why?

Internal Data:

Describe the private data maintained by this object. What knowledge does this object maintain?

Scenarios:

List the subsystem scenarios that this object participates in and describe the object’s behavior. Use state diagrams to assist in the description of the object behavior.
Timer → Packet Map

Packet-timer expiration message → Get_value() → Measurement value

Packet_send() → Downlink
BEGIN TASK
  Perform task initialization functions
DO FOREVER
  Wait for arrival of message
  SWITCH ON MESSAGE TYPE
    WHEN request for service A:
      Handle request
    WHEN request for service B:
      Handle request
    WHEN interrupt notification:
      Handle interrupt
    WHEN timer X expiration:
      Handle timer expiration
    WHEN timer Y expiration:
      Handle timer expiration
  END SWITCH
END DO
END TASK

Tasking Model

Task Object

BEGIN TASK
  Perform task initialization functions
DO FOREVER
  Wait for arrival of message
  SWITCH ON MESSAGE TYPE
    WHEN request for service A:
      Handle request
    WHEN request for service B:
      Handle request
    WHEN interrupt notification:
      Handle interrupt
    WHEN timer X expiration:
      Handle timer expiration
    WHEN timer Y expiration:
      Handle timer expiration
  END SWITCH
END DO
END TASK
Summary

“Never mistake activity for achievement.”
– John Wooden, UCLA Basketball Coach

- Requirements:
  - What does the system do?
  - Apportionment of responsibilities? (To functional subsystems)
  - Scenarios (Use cases, derived requirements)
  - Architectural Approach

- Design: Answer the five questions:
  - What are the pieces?
  - How do they fit?
  - Where is the logic?
  - Dynamic Behavior of the system?
  - Tasking Model?