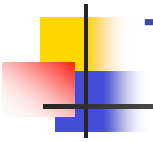
Rational Unified Process as Implemented at SNL

Karen M. Erickson
Data Systems Lead Engineer
Software Product Realization Organization
Department 5522 MS0974
Sandia National Laboratories
kmerick@sandia.gov



Topics

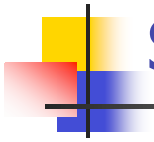
- Overview of SNL Satellite Ground System Project
- Software Development Process based on the RUP
- Lessons Learned

Satellite Ground System Description

- Processes satellite telemetry data
  - Acquire data from multiple satellites and different downlinks
  - Extract, process, store and display data
  - Combine data into meaningful information for the end users to enable their decisions
- Operational military system
  - High rigor, Reliable, Maintainable

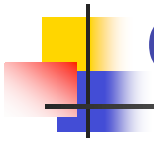
Satellite Ground Systems

- Two Satellite Ground Systems developed in parallel to promote software reuse.
- Systems are subject to extensive developmental control and testing by our customers.
- The systems were developed for multiple customers whose requirements can conflict
  - System must be optimized to meet all requirements



System Development

- Full life cycle development – cradle to grave
  - 8 years from inception to deployment
- Object Oriented Analysis and Design (UML)
  - 130 Use Cases
  - 5700 Classes
  - Currently ~1 million LOC in C++



Software Development Organization

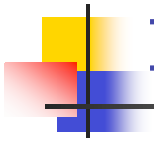
- 65 Software Professionals in 15 teams
  - System Engineering
  - Requirements Analysis
  - Architecture
  - Software Design & Development
  - Configuration and Build Management
  - Systems Integration
  - Integration Test
  - Computer Engineers
  - Deployment Engineers
- 1/3 to ½ are developers at any one time

Complete System Development

- To develop the system requires additional capabilities
  - Independent Test Organizations
    - System Test
    - Mission Analysis and Simulation
  - Research & Development
    - Algorithms
    - Simulators
    - Modeling
  - Support
    - System Administration
    - Development Environment Tool Development
- Additional 60 staff members, creating a multi-disciplinary team

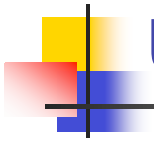
Software Development Process

- The Software Development Process is derived from the Rational Unified Process (RUP)
  - Iterative
  - Use Case Driven
  - Architecture Centric
  - Object Oriented Methodology
  - Supported by an integrated tool set



Iterative Development Process

- Supports full software development life cycle from requirements to test every iteration
  - Requirements Capture
  - Architecture Analysis
  - Design
  - Implementation
  - Test



Use Case Driven

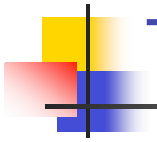
- Use Cases
  - Capture derived requirements
  - Describe the interaction of the user or external interface with the system to perform a single function
  - Use cases and scenarios drive the process flow from requirements through testing
  - Provides coherent and traceable threads through both the development and the delivered system

Architecture Centric

- Focuses on early development and baselining of a robust software architecture
  - Facilitates parallel development
  - Minimizes rework
  - Increases reusability
  - Increases reliability

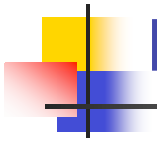
Object Oriented

- OO Methodology uses concepts of objects, classes, and the associations between classes
- Unified Modeling Language (UML) is used as the common notation in the RUP
  - Booch, Rumbaugh, Jacobson - The Unified Modeling Language User Guide:
    - "…a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML gives you a standard way to write a system's blueprints…"*



Tool Support

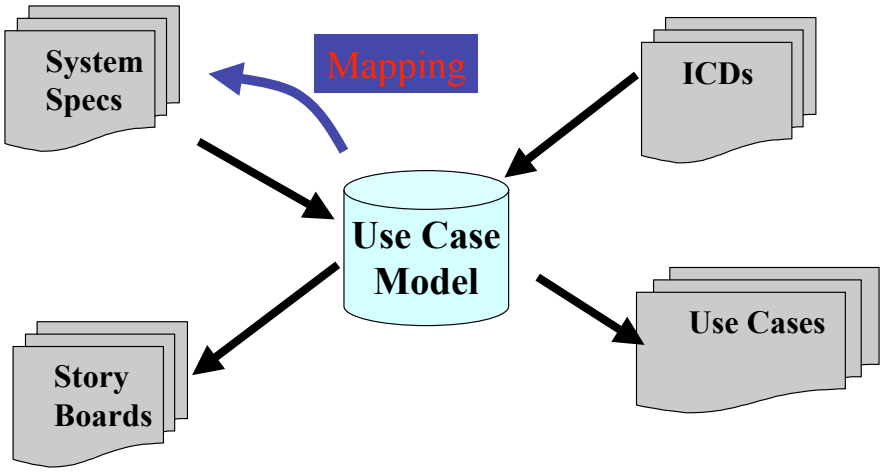
- The RUP is supported by tools that automate large parts of the process
- Tools are used to create and maintain the various artifacts from each process step
- Tools support maintaining models to describe the system design and replaces paper documentation



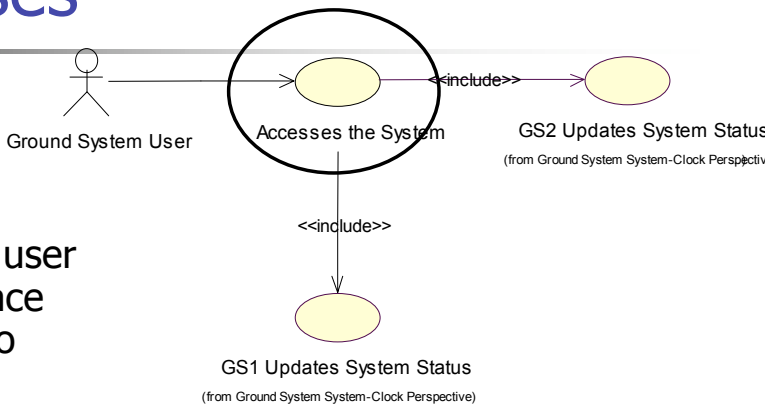
Project Implementation of RUP

- **Requirements Capture**
  - System Specification
  - Use Case Descriptions
- **Architectural Analysis**
  - Use Case Realizations
  - Subsystem Analysis Reports
- **Design**
  - Use Case Design
  - Class Design
- **Implementation**
  - Coding
- **Testing**
  - Integration Testing to Use Cases
  - System Testing to the System Spec

Requirements Capture



Use Cases



- Describes the interaction of the user or external interface with the system to perform a single function
- No specific architecture or implementation expressed

Use Case Descriptions

Typical Flow of Events

Actor Action	System Response
1. This use case begins when the Ground System User selects to gain access or change current access to GS1/GS2 as an individual user.	1. The ADP Software requests the user identification, password and user type.
2. The Ground System User enters a user identification, password and user type.	3. The ADP Software displays the appropriate user interface (based on the user type).
4. The Ground System User optionally selects to change the current user type (with a valid user identification and password). If not, go to step 9.	5. The ADP Software displays the appropriate user interface (based on the user type).
6. The Ground System User optionally selects to change the current user identification and password. If not, go to step 12.	7. The ADP Software maintains the current user interface (with a new user).
8. The Ground System User optionally selects to change the current user password.	9. The ADP Software requests the user's current password, the new password, and a confirmation of the new password.
10. The Ground System User enters current password, new password, and confirms the new password. 11. The Ground System User can repeat steps 5 and 7 and 9 as often as needed.	
12. The Ground System User selects to terminate their system access.	13. This use case ends when the user interface is terminated

Alternate Flow

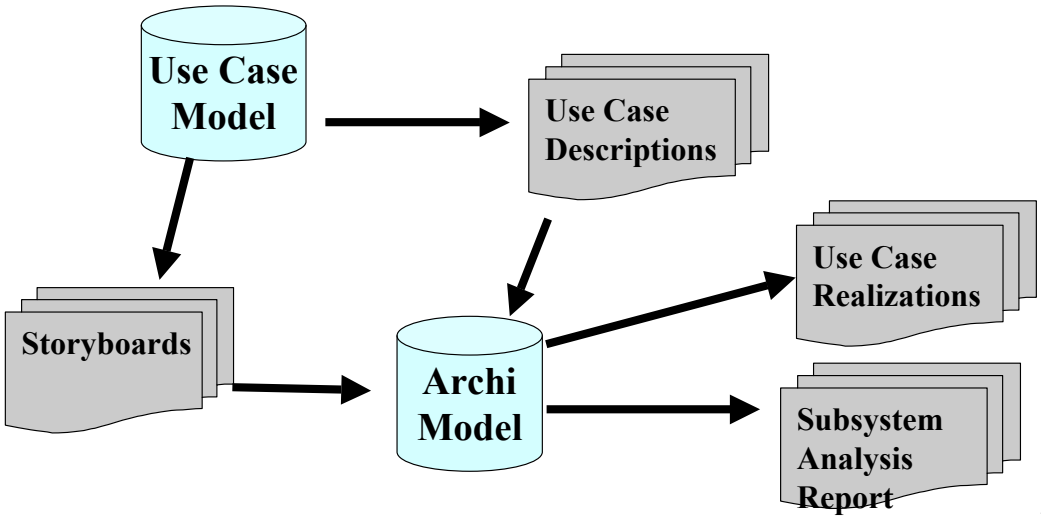
Use Case Storyboards

Step 1

The user enters his user identification (Tom), password and user type (AMC) and selects "OK".

Figure 1: Log on Window

Architectural Analysis



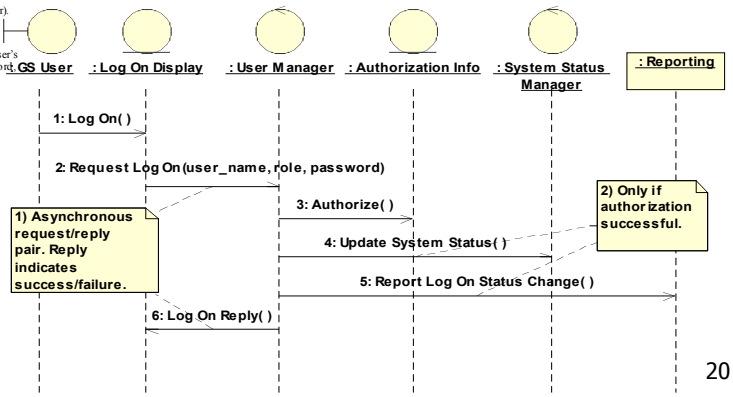
From Use Case to Realization

Typical Flow of Events

Actor Action	System Response
1. This use case begins when a Ground System User selects to gain access or change current access to GS1/GS2 as an individual user.	1. The ADP Software requests the user identification, password and user type.
2. The Ground System User enters a user identification, password and user type.	3. The ADP Software displays the appropriate user interface (based on the user type).
4. The Ground System User optionally selects to change the current user type (with a valid user identification and password). If not, go to step 6.	5. The ADP Software displays the appropriate user interface (based on the user type).
6. The Ground System User optionally selects to change the current user identification and password. If not, go to step 12.	7. The ADP Software maintains the current user interface (with a new user).
8. The Ground System User optionally selects to change the current user password.	9. The ADP Software requests the user's current password, the new password, and a confirmation of the new password.
10. The Ground System User enters current password, new password, and confirms password.	
11. The Ground System User repeat steps 5 and 7 and 9 as often as needed.	
12. The Ground System User selects to terminate their system access.	13. This use case ends when the user interface is terminated

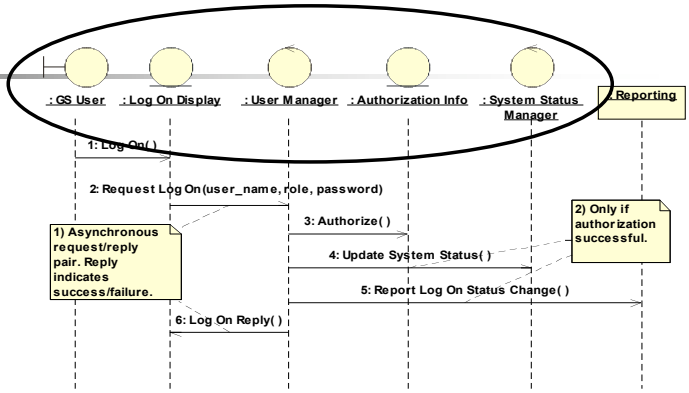
Alternate Flow

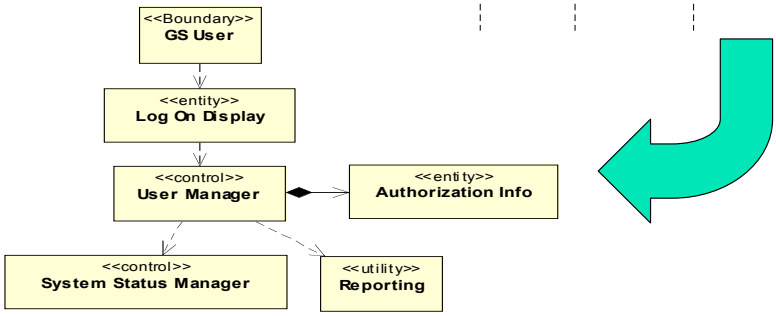
- **Realizations shows how the system should behave from an *internal* point of view**
- **One Realization for each Use Case**
- **Identifies and describes high-level system components and associated responsibilities**
- **The collection of Realizations as a whole represents one view of the Architecture**



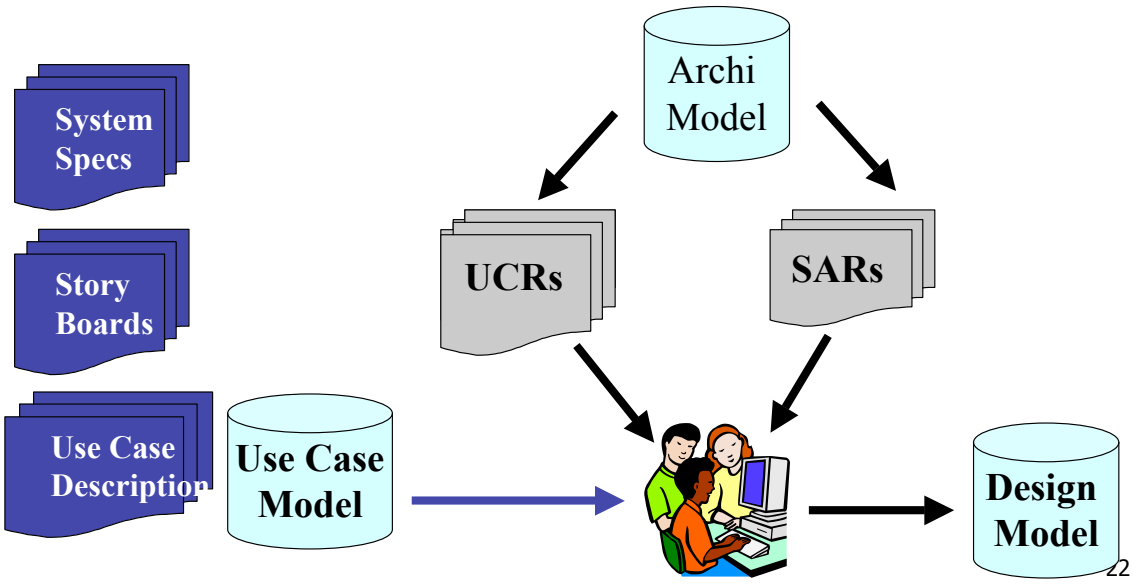
From Realization to Analysis Classes

- Realizations identify analysis classes
- Analysis classes are captured in Subsystem Analysis Reports (SARs)

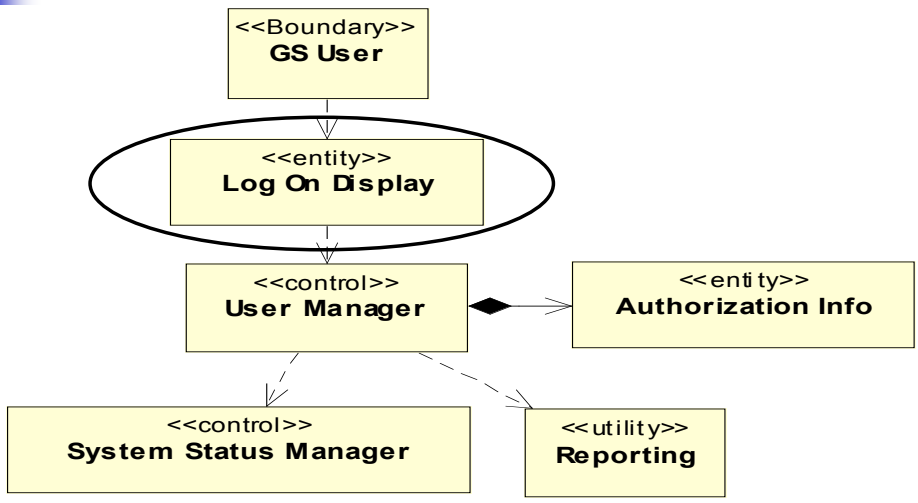




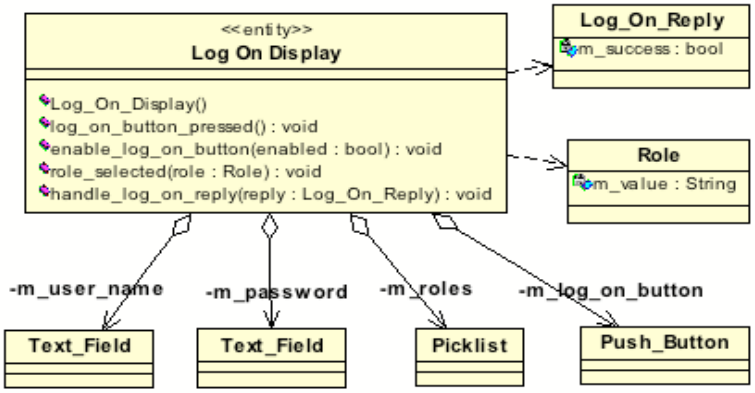
Design



Analysis Classes to Design Classes



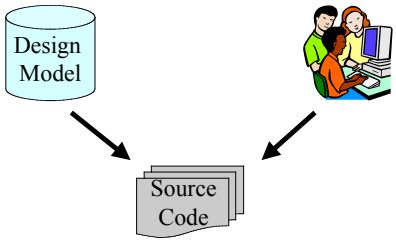
Design Classes



- A further elaboration of the Architecture
- Low-level: all details specified
- Suitable for generating source code framework

Implementation

- Implementation consists of
  - Code generation from the model
  - Filling in .cc files with detailed implementation
- Code generation from the model
  - Creates header files (.h)
    - Data definitions
    - Class interfaces



Code Inspections

- Code Inspections are a two step process
  - Review the class design in the model
    - Provides conceptual understanding and context
    - Examine relationships
    - Reviews details of data
  - Inspect Code
    - Focus on the implementation
    - Reviews not necessary for headers because reviewed with the model

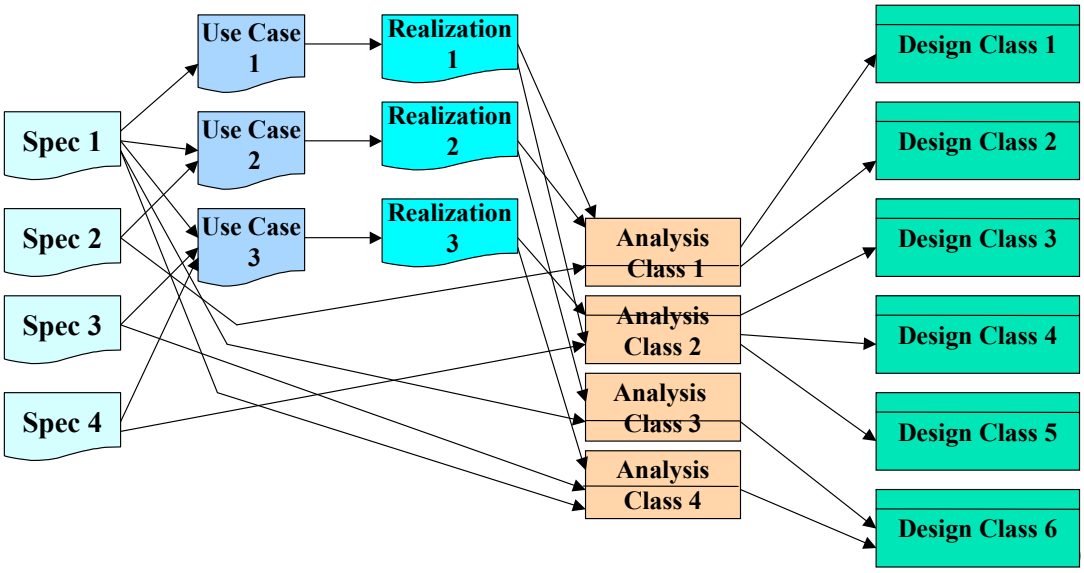
Unit/Integration Test and Delivery

- Components are controlled and built
- Unit testing is based on
  - Subsystem Analysis Reports (SAR's)
  - Use Case Realizations (UCR's)
- System is built and delivered to integration testbed
- Integration testing is based on Use Case Descriptions

System Test

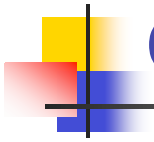
- System is built and delivered to system testbed
- System Testing
  - Based on System specs
  - Use Cases provide guidance and context for how to operate the system
  - Mapping of specs to Use Cases through to design provides traceability to help determine which test cases to execute
    - ~1000 specs
    - >100 Use Cases
    - Many to many mappings

Requirements Traceability



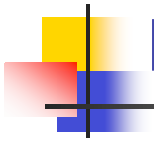
Site Delivery/Deployment

- System Tested release is delivered to site
  - System Verification Testing is performed
  - Acceptance Testing is performed by the customer



Operations & Maintenance

- O&M follows same development process as original development
- Modifications are put in the field at pre-defined intervals as requested by the customer



Lessons Learned?

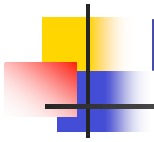
- Expectations vs. Reality
- Reflections
- SNL Success with the RUP

Expectations vs. Reality

- RUP expects
  - Small projects built in a short amount of time
  - Short iterations
  - Same people doing most of the steps of the process

Expectations vs. Reality

- Sandia Reality
  - Largest Software Development Project at SNL
  - Cost Estimate predicted 8 years of development
  - Iterations of 6 months
    - Not long enough to complete a full life cycle
    - Takes us approximately 18 months
  - Division of responsibilities between teams
    - No continuity of personnel in steps of process
    - Handoffs between teams more formal than RUP envisioned

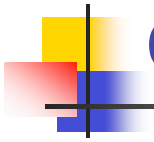


Reflections

- Sandia was one of the original customers of the RUP.
  - Our use of the RUP evolved as the RUP itself was evolving.
  - We had a good working relationship with Rational.
    - Opportunity to provide feedback to Rational and have it incorporated into their product.
    - Biggest project that had ever been built using RUP.
    - Relationship changed when Rational was purchased by IBM.
  - Process/Tool Development
    - The process and tools did not meet our needs out of the box
    - It took us several years to fully understand and implement the process before we were very productive.
    - We had to integrate a lot of the tools ourselves and make them fit our version of the process.

SNL Success with the RUP

- Once the process was defined and the tools well integrated we evolved into a highly productive organization
  - We were able to integrate 14 new staff members one summer and still meet our deliverables that iteration
- These two satellite ground systems are being delivered ***on time, within budget*** and ***meeting all requirements***.



Conclusion

- The RUP provided a framework for Sandia to develop a process that works for our project.
- Sandia will be using our modified version of the RUP on future projects.