Software Production Process

Tom Swain
Software Quality Research Laboratory
University of Tennessee
Department of Computer Science

Copyright © 2005 Software Quality Research Laboratory
Process Objectives

- Ensure **Software Reliability**
- Establish **Quantitative Project Management Criteria**
- Enable **Efficient Software Engineering Practices**
- Simplify **Compliance and Enforcement**
Essential Process Elements

1) Requirements Capture
2) Architecture Specification
3) Behavior Specification
4) Code Generation
5) Certification
6) User Guide Publication
7) Configuration Management
8) Change Management
9) Metrics Collection and Analysis
Primary Work Product Relationships

1a. Derive Specification
1b. Invent Architecture
2. Map
3a. Implement Structure
3b. Implement Function
4. Model Usage & Create Oracle
5. Describe Usage

Preliminary Requirements

Architecture Specification
Behavior Specification
User Guide

Code
Test Plan
Preliminary Requirements

- Individual requirements tagged for traceability.
- Initial requirements assumed to be incomplete, inconsistent, and possibly incorrect.
Tagged Requirements Example

<table>
<thead>
<tr>
<th>Tag</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The TMC shall compute an accurate, but conservative, static estimate of DNBR every 100 ms. Sensor inputs for the DNBR calculation are pressurizer pressure ((p)), neutron flux at selected locations ((\phi_i)), primary coolant mass flow rate ((F)), cold leg temperature ((T_c)), and hot leg temperature ((T_h)).</td>
</tr>
<tr>
<td>2</td>
<td>The DNBR, (D_d), is compared to a trip setpoint, (D_t), each time it is computed. If (D_d) is less than the setpoint, then the trip contact output is set.</td>
</tr>
<tr>
<td>3</td>
<td>All sensor inputs are checked for validity when sampled. If a sensor value is invalid, the channel failure indicator is set.</td>
</tr>
</tbody>
</table>
| 4   | The operator interface has the following features:  
  - Numeric displays for the point ID and value of sensor inputs, calculated values, calibration constants, and setpoints.  
  - An alphanumeric display of the current point ID.  
  - A numeric keypad and associated function keys for entry of point ID selections and new values for calibration constants and setpoints.  
  - Indicator lights for trip status and channel failure status.  
  - A keyswitch to turn power on and off. |
| 5   | If a non-existent or invalid point ID is entered, the point ID and value displays revert to their previous values. |
| 6   | If an invalid calibration or setpoint value is entered, the value display reverts to the original value. |
| 7   | A watchdog timer must be reset at least every 100 ms. Otherwise the channel failure indication is set and a reboot command is issued. |
| 8   | When a TMC is powered up or rebooted, all sensor inputs are read and used to initialize the DNBR calculation, the watchdog timer is reset, and the trip contact output is set. |
| 9   | If a TMC is powered down or rebooted the trip contact output must be set. |
Architecture Specification Contents

- Define components and their responsibilities
- Specify relationships among components
- Specify intra-system and external interfaces
- Ensure unidirectional use hierarchy
- Specify assumptions regarding platform/environment
DANSE Architecture Specifications

- **Framework** – interfaces, services, and rules for well-behaved components
- **Each Component** – class diagrams and API design
- **Each Application** – target environment, participating components, and their relationships
Behavior Specification Basics

- Basic Premise
  - A *software program* is a rule for computing a *mathematical function* that maps *all* possible stimulus histories to *all* possible responses

- Approach
  - Derive mathematically *rigorous specification* and design from requirements
Behavior Specification Objectives

- **Completeness**
  - a response is defined for every stimulus history

- **Consistency**
  - each stimulus history maps to only one response

- **Correctness**
  - the specification is explicitly traceable to the requirements
The Simple Case: Static Calculations or Processes

1) Partition input space into domains bounded by discontinuities
2) Specify response function for EVERY domain
The General Case:
Response Depends on Stimulus Sequence

1) Establish **system boundary**.
2) Define human/software/hardware **interfaces**.
3) Itemize **stimuli**.
4) Itemize **responses**.
5) Perform **sequence enumeration**.
6) Perform **canonical sequence analysis**.
7) Generate **state machine specification**.
Derived Requirements

The discovery and documentation of derived requirements is a natural and desirable part of the specification process.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>If a second 100 ms clock interrupt is received before task completion, declare a channel failure and initiate a reboot.</td>
</tr>
</tbody>
</table>
DANSE Behavior Specifications

- DANSE Framework imposes a component Life-Cycle.

- Sequence is important and enumeration is necessary at the component level and above.
Map Behavior Specification to Architecture
DANSE Code Generation

- Define component-level architecture according to the framework.
  - Define stimulus gathering classes.
  - Define response generation classes.
  - Define classes to handle state transitions.
- Map state data to specific classes.
- Map implementation of State Box table entries to class methods.

Document class relationships and API inline via epydoc, Doxygen, etc.
Application Scalability Issues

- Partition system according to high-level architecture
- Derive rigorous specifications at the component level
- Verify integrated behavior
  - I/O audit
  - Formal verification
  - Incremental integration and test
Release Package Certification

- Evidence that approved software engineering practices were used to produce the software
- Results of inspection and tests provide confidence that the as-built software will perform its intended function.
Certification Protocol

- Architecture and Behavior Specs updated to as-built design – review action items resolved
- Code inspection complete – review action items resolved
- Verification testing complete – defect correction for all failures resolved
- Beta test complete – reported issues resolved
- Closure of all items in CR package
Independent Review

- Domain Expert Review
  - Initial Requirements
  - Behavior Specification
  - Usage Model

- Development Team Peer Review
  - Architecture Specification
  - Test Plan

- Code Inspection
  - Automated Enforcement of Coding Standards
  - Manual Verification of Functional Correctness
Manual and Automated Code Inspection

1. Identify defects related to semantic errors and hazardous coding practices.
2. Verify correct code functionality with respect to specification.
3. Iterate until defect volume is acceptable for testing.
Testing is Always Sampling

Population (All Uses)

What to test: a statistically appropriate sample

How much to test: to an acceptable reliability estimate
Model-Based Statistical Testing

1. Build Model → TML Files
2. Analyze/Revise Model → HTML Report
3. Generate Test Cases → Test Cases/Scripts
4. Analyze/Revise Test Suite → HTML Report
5. Execute Tests → Test Results
6. Reliability Analysis → HTML Report
7. Retest
8. Release Decision → Release Criteria
9. Release

Usage Modeling and Test Planning
Test Case Generation and Automation Support
Results Management and Quantitative Analysis

Copyright © 2005 Software Quality Research Laboratory
Usage Model: All Possible Uses and their Likelihood

STATE MACHINE
Nodes are states-of-use
Arcs are possible stimuli
Probabilities define expected usage
Test case is path from initial to terminal state
Test Automation

Parametric Data Generation for Abstract Stimuli
  - Partition and Sample Input Space

Test Case Generation
  - Usage Model
    - Generate Test Cases
  - Export Test Scripts
  - Test Record
  - Analyze Results

Test Execution
  - Compare Results to Oracle
  - Record Pass/Fail
  - Result Checking and Comparison

Release?
Results Analysis / Release Decision

Record all failures by test case number and transition where failure occurs, then:

- Compute **estimated reliability** based on test results
- Evaluate **stopping criteria**
- **Release or fix and retest** based on semi-quantitative risk assessment
DANSE Testing Strategy

- Independent Test Leader for Framework and Each Science Team
- Automated Model-Based Testing at Component Level
- Incremental Integration and Test for Each Application
Work Products Supporting Certification

- Peer Review Findings and Resolution
  - Specifications
  - Code
  - Test Plan
- Test Results and Resolution of Failures
- Release Decision Meeting Minutes
(1) A seamless, traceable transformation from requirements to code specification.

(2) Mapping specified code to a robust, maintainable architecture.

(3) Peer review of key work products including 100% code inspection.

(4) Quantitative reliability assessment via model-based statistical testing.
User Guide

- **Content**
  - How to install and/or configure the system
  - How to initiate and control all functions
  - How to interpret output

- **Format**
  - HTML for use with browser
  - Online “Help” – integral to system
Configuration Items

- Project Plan (Cat 3 only)
- Software Requirements*
- Software Behavior Specification
- Software Architecture Specification
- User Guide
- Software Test Plan
- Source Code
- Test Record

*superceded by Behavior Spec
Software Configuration Management
Configuration/Release Control

Key Practices

- Source control tool (e.g., CVS or Subversion)
- Centrally managed release tagging
- Repeatable automated build procedure
- Formal release procedure
- Change tracking
- Audit trail
Change Request Processing

SQET ≈ Testing and Release Control
Productivity and Reliability Metrics

- **Software volume**
  - Function points based on the Behavior Specification
  - Source lines of executable code (SLOEC) added, modified, or deleted

- **Development and test effort**
  - Staff time spent producing a software increment or
  - Actual dollar cost to produce a software increment or
  - Calendar time required to produce a software increment

- **Reliability**
  - Total failures reported by users vs. total number of uses based on system logs
  - Failures reported by users per unit time
Process Summary

- Rigorous specification of code derived from informal requirements
  - Architecture
  - Behavior
- Certification
  - Independent Work Product Review
  - Quantitative Testing
- Configuration management
  - Centralized build/release control
  - Comprehensive change tracking
- Metrics to support process improvement