Incremental Life-Cycle of Cyber-Physical Systems

Julien Delange <jdelange@sei.cmu.edu>
Peter Feiler <phf@sei.cmu.edu>

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Software issues: facts and fiction

Why do system level failures still occur despite best design methods & fault tolerance techniques being deployed in systems?

Where faults are introduced
Where faults are found
The estimated nominal cost for fault removal

Requirements Engineering

System Design

Software Architectural Design

Component Software Design

Code Development

Unit Test

Integration Test

System Test

Acceptance Test

Actual Methods and Proposed Approach

Where faults are introduced
Where faults are found
The estimated nominal cost for fault removal

Requirements Engineering
System Design
Software Architectural Design
Component Software Design
Code Development
Integration Test
Unit Test
System Test
Acceptance Test
Actual Methods and Proposed Approach

Where faults are introduced

Where faults are found

The estimated nominal cost for fault removal

- Requirements Engineering
- System Design
- Software Architectural Design
- Component Software Design
- Code Development
- Unit Test
- Integration Test
- System Test
- Acceptance Test

70%, 3.5%
1x
20%, 16%
5x
40x
110x

Where faults are introduced

Where faults are found

The estimated nominal cost for fault removal
Where faults are introduced
Where faults are found
The estimated nominal cost for fault removal

Incremental Life-Cycle of CPS Systems
June 09, 2015
© 2015 Carnegie Mellon University
Certification & Recertification Challenges

Certification: assure the quality of the delivered system
- **Sufficient evidence** that a **system implementation** meets **system requirements**
- **Quality of requirements and quality of evidence** determines quality of system

Certification related rework cost
- Currently 50% of total system cost and growing

Recertification Challenge
- Desired cost of recertification in proportion to change
## Requirement Quality Challenge

<table>
<thead>
<tr>
<th>Requirements error</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incomplete</td>
<td>21%</td>
</tr>
<tr>
<td>Missing</td>
<td>33%</td>
</tr>
<tr>
<td>Incorrect</td>
<td>24%</td>
</tr>
<tr>
<td>Ambiguous</td>
<td>6%</td>
</tr>
<tr>
<td>Inconsistent</td>
<td>5%</td>
</tr>
</tbody>
</table>
## Requirement Quality Challenge

<table>
<thead>
<tr>
<th>Requirements error</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incomplete</td>
<td>21%</td>
</tr>
<tr>
<td>Missing</td>
<td>33%</td>
</tr>
<tr>
<td>Incorrect</td>
<td>24%</td>
</tr>
<tr>
<td>Ambiguous</td>
<td>6%</td>
</tr>
<tr>
<td>Inconsistent</td>
<td>5%</td>
</tr>
</tbody>
</table>
There is more to requirements quality than “shall”’s and stakeholder traceability.

IEEE Std 830-1998 Recommended Practice for SW Requirements Specification

<table>
<thead>
<tr>
<th>Requirements error</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incomplete</td>
<td>21%</td>
</tr>
<tr>
<td>Missing</td>
<td>33%</td>
</tr>
<tr>
<td>Incorrect</td>
<td>24%</td>
</tr>
<tr>
<td>Ambiguous</td>
<td>6%</td>
</tr>
<tr>
<td>Inconsistent</td>
<td>5%</td>
</tr>
</tbody>
</table>

IEEE Std 830-1998 characteristics of a good requirements specification:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable
Requirement Quality Challenge

<table>
<thead>
<tr>
<th>Requirements error</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incomplete</td>
<td>21%</td>
</tr>
<tr>
<td>Missing</td>
<td>33%</td>
</tr>
<tr>
<td>Incorrect</td>
<td>24%</td>
</tr>
<tr>
<td>Ambiguous</td>
<td>6%</td>
</tr>
<tr>
<td>Inconsistent</td>
<td>5%</td>
</tr>
</tbody>
</table>

IEEE Std 830-1998 characteristics of a good requirements specification:
- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

There is more to requirements quality than “shall”’s and stakeholder traceability

*IEEE 830-1998 Recommended Practice for SW Requirements Specification*

System to SW requirements gap [Boehm 2006]

*How do we verify low level SW requirements against system requirements?*

When StartUpComplete is TRUE in both FADECs and SlowStartupComplete is FALSE, the FADECStartupSW shall set SlowStartupIncomplete to TRUE.
## Industry Practice in DO-178B Compliant Requirements Capture

**Industry Survey in 2009 FAA Requirements Engineering Study**

### Notation

Enter an “x” in every row/column cell that applies

### Need analyzable & executable specifications

<table>
<thead>
<tr>
<th>Tool</th>
<th>System Requirements</th>
<th>Data Interconnect (ICD)</th>
<th>High-Level Software Requirement</th>
<th>Low-Level Software Requirement</th>
<th>Hardware Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOORS</td>
<td>3  4  3  3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rational ROSE®</td>
<td>23</td>
<td>13</td>
<td>22</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>RDD-100®</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requisite Pro®</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Rhapsody</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCADE Suite</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simulink</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Slate</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spreadsheet (e.g., Microsoft Excel)</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Word Processor (e.g., Microsoft Word)</td>
<td>19</td>
<td>20</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>VAPS™</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Designer's Workbench™</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proprietary Database: SCADE like pic tool</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interleaf</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>BEACON</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CaliberRM</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>XM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wiring Diagram</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Enter an “x” in every row/column cell that applies
Mixture of Requirements & Architecture Design Constraints

Requirements for a Patient Therapy System

The patient shall never be infused with a single air bubble more than 5ml volume.

When a single air bubble more than 5ml volume is detected, the system shall stop infusion within 0.2 seconds.

When piston stop is received, the system shall stop piston movement within 0.01 seconds.

The system shall always stop the piston at the bottom or top of the chamber.

Adapted from M. Whalen presentation
Requirements for a Patient Therapy System

The patient shall never be infused with a single air bubble more than 5ml volume.

When a single air bubble more than 5ml volume is detected, the system shall stop infusion within 0.2 seconds.

When piston stop is received, the system shall stop piston movement within 0.01 seconds.

The system shall always stop the piston at the bottom or top of the chamber.

1. The patient shall never be infused with a single air bubble more than 5ml volume.

2. When a single air bubble more than 5ml volume is detected, the system shall stop infusion within 0.2 seconds.

3. When piston stop is received, the system shall stop piston movement within 0.01 seconds.

4. The system shall always stop the piston at the bottom or top of the chamber.

Typical requirement documents span multiple levels of a system architecture

We have made architecture design decisions.

We have effectively specified a partial architecture

Adapted from M. Whalen presentation
Model-Based Requirements Engineering Framework

Stakeholders Goals
(“The temperature should not exceed 50F”)

System Requirements
(“The heater is equipped with a sensor”)

System Architecture
(“There is a sensor device” and “The device is connected to the temperature management software”)

Development - Implementation
(Design Model Behavior + Generate Code)

(Automated) Assurance Case
(“The Temperature does not exceed 50F”)

(Automated) System Tests
(“The SW and the heater are implemented correctly”)

(Automated) Integration Tests
(“The Software Receives the Temperature”)

Measure Quality
Derived from
Stakeholders Goals
("The temperature should not exceed 50F")

System Requirements
("The heater is equipped with a sensor")

System Architecture
("There is a sensor device" and "The device is connected to the temperature management software")

Development - Implementation
(Design Model Behavior + Generate Code)

(Automated) Assurance Case
("The Temperature does not exceed 50F")

(Automated) System Tests
("The SW and the heater are implemented correctly")

(Automated) Integration Tests
("The Software Receives the Temperature")
Stakeholders Goals
(“The temperature should not exceed 50F”)

System Requirements
(“The heater is equipped with a sensor”)

System Architecture
(“There is a sensor device” and “The device is connected to the temperature management software”)

Development - Implementation
(Design Model Behavior + Generate Code)

(Automated) Assurance Case
(“The Temperature does not exceed 50F”)

(Automated) System Tests
(“The SW and the heater are implemented correctly”)

(Automated) Integration Tests
(“The Software Receives the Temperature”)

Model-Based Engineering Domain
Stakeholders Goals
(“The temperature should not exceed 50F”)

System Requirements
(“The heater is equipped with a sensor”)

System Architecture
(“There is a sensor device” and “The device is connected to the temperature management software”)

Development - Implementation
(Design Model Behavior + Generate Code)

Requirements Engineering Domain
(Automated) System Tests
(“The SW and the heater are implemented correctly”)
(Automated) Integration Tests
(The Software Receives the Temperature”)

Model-Based Engineering Domain
(Automated) Assurance Case
(“The Temperature does not exceed 50F”)

20.5%
Stakeholders Goals
("The temperature should not exceed 50F")

Requirements Engineering Domain

(Automated) Assurance Case
("The Temperature does not exceed 50F")

(Automated) System Tests
("The SW and the heater are implemented correctly")

System Requirements
("The heater is equipped with a sensor")

Reqspec language

System Architecture
("There is a sensor device" and "The device is connected to the temperature management software")

Model-Based Engineering Domain

Development - Implementation
(Design Model Behavior + Generate Code)

(Automated) Integration Tests
("The Software Receives the Temperature")
Stakeholders Goals
("The temperature should not exceed 50F")

System Requirements
("The heater is equipped with a sensor")

System Architecture
("There is a sensor device" and "The device is connected to the temperature management software")

Requirements Engineering Domain

(Automated) Assurance Case
("The Temperature does not exceed 50F")

(Automated) System Tests
("The SW and the heater are implemented correctly")

(Automated) Integration Tests
("The Software Receives the Temperature")

Development - Implementation
(Design Model Behavior + Generate Code)

Requirements Engineering Domain

Verify language

Model-Based Engineering Domain

Reqspec language
Stakeholders Goals
(“The temperature should not exceed 50F”)  

System Requirements
(“The heater is equipped with a sensor”)  

System Architecture
(“There is a sensor device” and “The device is connected to the temperature management software”)  

Development - Implementation
(Design Model Behavior + Generate Code)  

(Automated) Integration Tests
(“The Software Receives the Temperature”)  

(Automated) System Tests
(“The SW and the heater are implemented correctly”)  

(Automated) Assurance Case
(“The Temperature does not exceed 50F”)  

Verify language  

Assure language  

Reqspecl language  

Requirements Engineering Domain  

Model-Based Engineering Domain
Stakeholders Goals
(“The temperature should not exceed 50F”)

System Requirements
(“The heater is equipped with a sensor”)

System Architecture
(“There is a sensor device” and “The device is connected to the temperature management software”)

Development - Implementation
(Design Model Behavior + Generate Code)

Requirements Engineering Domain

Model-Based Engineering Domain

Word/Excel/ReqIF Import

D-CASE/CertWare Export

Stakeholders Goals
(“The temperature should not exceed 50F”)

System Requirements
(“The heater is equipped with a sensor”)

System Architecture
(“There is a sensor device” and “The device is connected to the temperature management software”)

Development - Implementation
(Design Model Behavior + Generate Code)

Requirements Engineering Domain

Model-Based Engineering Domain

Verify language

Assure language

Automated) Assurance Case
(“The Temperature does not exceed 50F”)

Automated) System Tests
(“The SW and the heater are implemented correctly”)

Automated) Integration Tests
(“The Software Receives the Temperature”)

Automated) System Tests
(“The SW and the heater are implemented correctly”)

Automated) Assurance Case
(“The Temperature does not exceed 50F”)

Word/Excel/ReqIF Import

D-CASE/CertWare Export

Stakeholders Goals
(“The temperature should not exceed 50F”)

System Requirements
(“The heater is equipped with a sensor”)

System Architecture
(“There is a sensor device” and “The device is connected to the temperature management software”)

Development - Implementation
(Design Model Behavior + Generate Code)

Requirements Engineering Domain

Model-Based Engineering Domain

Verify language

Assure language

Automated) Assurance Case
(“The Temperature does not exceed 50F”)

Automated) System Tests
(“The SW and the heater are implemented correctly”)

Automated) Integration Tests
(“The Software Receives the Temperature”)

Automated) System Tests
(“The SW and the heater are implemented correctly”)

Automated) Assurance Case
(“The Temperature does not exceed 50F”)

Word/Excel/ReqIF Import

D-CASE/CertWare Export

Stakeholders Goals
(“The temperature should not exceed 50F”)

System Requirements
(“The heater is equipped with a sensor”)

System Architecture
(“There is a sensor device” and “The device is connected to the temperature management software”)

Development - Implementation
(Design Model Behavior + Generate Code)

Requirements Engineering Domain

Model-Based Engineering Domain

Verify language

Assure language

Automated) Assurance Case
(“The Temperature does not exceed 50F”)

Automated) System Tests
(“The SW and the heater are implemented correctly”)

Automated) Integration Tests
(“The Software Receives the Temperature”)

Automated) System Tests
(“The SW and the heater are implemented correctly”)

Automated) Assurance Case
(“The Temperature does not exceed 50F”)

Word/Excel/ReqIF Import

D-CASE/CertWare Export

Stakeholders Goals
(“The temperature should not exceed 50F”)

System Requirements
(“The heater is equipped with a sensor”)

System Architecture
(“There is a sensor device” and “The device is connected to the temperature management software”)

Development - Implementation
(Design Model Behavior + Generate Code)

Requirements Engineering Domain

Model-Based Engineering Domain

Verify language

Assure language

Automated) Assurance Case
(“The Temperature does not exceed 50F”)

Automated) System Tests
(“The SW and the heater are implemented correctly”)

Automated) Integration Tests
(“The Software Receives the Temperature”)

Automated) System Tests
(“The SW and the heater are implemented correctly”)

Automated) Assurance Case
(“The Temperature does not exceed 50F”)
Increased Confidence and Reduced Certification Cost through Incremental Life Cycle Assurance

Major cost savings through rework avoidance by early discovery

Architecture Focused Requirements Analysis
Virtual Architecture Integration & Analysis
Design Validation by Virtual Integration
Code Coverage Testing

Requirements Engineering → Requirements Validation → System & SW Architectural Design → System & SW Architecture Validation

Build the System
Build the Assurance Case

Requirements Validation → System & SW Architectural Design

Target Build → Integration Build → Integration Test → System Test → Flight Test

Deployment Build → Acceptance Test

System Integration Lab Testing

Design Validation

Unit Test → Test System → Test System

Unit Test

Increased Confidence and Reduced Certification Cost through Incremental Life Cycle Assurance

Major cost savings through rework avoidance by early discovery

Architecture Focused Requirements Analysis
Virtual Architecture Integration & Analysis
Design Validation by Virtual Integration
Code Coverage Testing

Requirements Engineering → Requirements Validation → System & SW Architectural Design → System & SW Architecture Validation

Build the System
Build the Assurance Case

Requirements Validation → System & SW Architectural Design

Target Build → Integration Build → Integration Test → System Test → Flight Test

Deployment Build → Acceptance Test

System Integration Lab Testing

Design Validation

Unit Test → Test System → Test System

Unit Test
Integrated Approach to Requirement V&V through Assurance Automation

- Requirement coverage
- Assumption evidence
- Evidence records in terms of claims that requirements have been met
- Safety hazards are part of the picture

Generated assurance cases
Linkage to automated test harnesses

Safety hazards are part of the picture
Building the Assurance Case throughout the Life Cycle

Architecture-centric Virtual Integration

Early Discovery through Architecture Analysis leads to Assurance Related Rework Reduction

Build the System

FY15/16 line funded project
Building the Assurance Case throughout the Life Cycle

**Architecture-centric Virtual Integration**

- Requirements Verification
- System & SW Architecture Verification
- Design Verification
- Code Verification
- Requirements Engineering
- Component Software Design
- Code Development
- System & SW Architectural Design
- Integration Build
- Unit Test
- Build the System
- Increased Confidence
- Cost-effective Tests
- Auto-generated Assurance Cases
- Build the Assurance Case

**Early Discovery through Architecture Analysis leads to Assurance Related Rework Reduction**

**Auto-generation from verified models**
- AADL&SCADE/Simulink
- Ada SPARK/Ravenscar
- MISRA C

**FY15/16 line funded project**
Building the Assurance Case throughout the Life Cycle

Continuous Confidence Measure throughout Life Cycle that a System Meets its Requirements

Architecture-centric Virtual Integration

Architecture Led Requirements Specification

Virtual Architecture Integration & Analysis

Design Validation by Virtual Integration

System & SW Architecture Verification

Component Software Design

Verification

Deployment Build

Build the System

Flight Test

System Integration Lab Testing

Code Coverage Testing

Auto-generation from verified models
AADL&SCADE/Simulink
Ada SPARK/Ravenscar
MISRA C

Increased Confidence

Cost-effective Tests

Auto-generated Assurance Cases

Need for Multi-valued Argumentation Logic

Confidence = Requirement Quality + Evidence Quality

CF

FY15/16 line funded project
Building the Assurance Case throughout the Life Cycle

Continuous Confidence Measure throughout Life Cycle that a System Meets its Requirements

**Architecture-centric Virtual Integration**

- Architecture Led Requirements Specification
- Virtual Architecture Integration & Analysis
- Design Validation by Virtual Integration
- System Integration Lab Testing
- Code Coverage Testing
- Flight Test

Incremental Evolution and Execution of Assurance Plans

- Incremental Architecture & Requirement Evolution
- Requirement Coverage
- Design & Req Refinement

Need for Multi-valued Argumentation Logic

Confidence = Requirement Quality + Evidence Quality

Auto-generated Assurance Cases

Build the System

FY15/16 line funded project

Build the Assurance Case

Auto-generation from verified models: AADL&SCADE/Simulink, Ada SPARK/Ravenscar, MISRA C

Increased Confidence

Cost-effective Tests

Confidence = Requirement Quality + Evidence Quality

Need for Multi-valued Argumentation Logic

Confidence = Requirement Quality + Evidence Quality

Early Discovery through Architecture Analysis leads to Assurance Related Rework Reduction

Auto-generated Assurance Cases

Build the System

FY15/16 line funded project

Build the Assurance Case
Building the Assurance Case throughout the Life Cycle

Continuous Confidence Measure throughout Life Cycle that a System Meets its Requirements

Architectural Evolution and Execution of Assurance Plans

Incremental Evolution and Execution of Assurance Plans

Incremental Architecture & Requirement Evolution

Virtual Architecture Integration & Analysis

System Integration Lab Testing

Code Coverage Testing

Design Validation by Virtual Integration

Architecture Led Requirements Specification

Early Discovery through Architecture Analysis leads to Assurance Related Rework Reduction

Auto-generation from verified models
AADL&SCADE/Simulink
Ada SPARK/Ravenscar
MISRA C

Need for Multi-valued Argumentation Logic

Confidence = Requirement Quality + Evidence Quality

Auto-generated Assurance Cases

Build the System

Build the Assurance Case

FY15/16 line funded project
Building the Assurance Case throughout the Life Cycle

Continuous Confidence Measure throughout Life Cycle that a System Meets its Requirements

Architecture-centric Virtual Integration

Incremental Evolution and Execution of Assurance Plans

Incremental Architecture & Requirement Evolution

Need for Multi-valued Argumentation Logic

Confidence = Requirement Quality + Evidence Quality

Auto-generated Assurance Cases

Build the Assurance Case

FY15/16 line funded project
Building the Assurance Case throughout the Life Cycle

Continuous Confidence Measure throughout Life Cycle that a System Meets its Requirements

Architecture-centric Virtual Integration

Architecture Led Requirements Specification

Virtual Architecture Integration & Analysis

System & SW Architecture Verification

Design Validation by Virtual Integration

Code Coverage Testing

Increased confidence

Cost-effective Tests

Auto-generated Assurance Cases

Need for Multi-valued Argumentation Logic

Confidence = Requirement Quality + Evidence Quality

Auto-generation from verified models
AADL&SCADE/Simulink
Ada SPARK/Ravenscar
MISRA C

Incremental Evolution and Execution of Assurance Plans

Incremental Architecture & Requirement Evolution

Incremental Contract-based Compositional Verification

FY15/16 line funded project

Build the System

Build the Assurance Case

Flight Test

System Integration Lab Testing

Deployment Build

Early Discovery through Architecture Analysis leads to Assurance Related Rework Reduction

Requirements Engineering

Integration Test

Integration

Component Software Design

System & SW Architectural Design

Target Build

Code Development

Component Software Design

System & SW Architectural Design

Target Build

Requirements Engineering

Verification

Build the System

Build the Assurance Case
Actual Implementation

Prototype Implementation

- Xtext for language design
- Sirius for graphical views

Bridges with Existing Tools

- Word/Excel/Doors/ReqIF bridge to import/export requirements
- Assurance Case Generation with GSN/D-CASE

Availability as an Open-Source Platform

- Release in late 2015
- Included with the AADL OSATE toolset (http://www.aadl.info)
Research Agenda

First Experimental Projects (late 2015/early 2016)
  • Avionics (Wheel Brake System) project with SAVI
  • Application with Medical Devices

ROI study (2016)
  • Measure of Requirements Improvement
  • Investment returns, development savings

Tooling Improvement (mid-2016)
  • User-Friendly Interfaces, Graphical View (Tables + Diagram)
  • Assurance Case Diagrams within Eclipse
Conclusion

Reduce risks

- Analyze system early and throughout life cycle
- Understand system wide impact
- Validate assumptions across system

Increase confidence

- Validate models to complement integration testing
- Validate model assumptions in operational system
- Evolve system models in increasing fidelity

Reduce cost

- Fewer system integration problems
- Incremental evidence through compositional verification
- Fewer verification steps through generation from single source and verified models
Contact Information

Julien Delange
Member of the Technical Staff
Telephone: +1 412-268-9652
Email: jdelange@sei.cmu.edu

U.S. Mail
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

Web
www.aadl.info
www.aadl.info/wiki

Customer Relations
Email: info@sei.cmu.edu
SEI Phone: +1 412-268-5800
SEI Fax: +1 412-268-6257