Agile Development for Safety Critical Embedded Systems

Bruce Powel Douglass, Ph.D.
Chief Evangelist, IBM/Rational
Bruce.Douglass@us.ibm.com
Twitter: @BruceDouglass
http://tech.groups.yahoo.com/group/RT-UML

Contact me if you want the slides!
Misconceptions about Agile

- Agile is undisciplined
- Agile is unplanned
- Agile doesn’t document
- Agile does “stories” not “requirements”
- Agile doesn’t use modeling
- Agile doesn’t test
- Agile is “hacking”
- Agile only applies to software
- Agile only applies to IT, not Systems
- Agile doesn’t apply to safety critical systems
Agile for Systems Engineering?

- Systems engineering reasons about system functionality, performance, technologies to specify requirements, architectures, interfaces, and allocations
- Agile can assist by
  - Improving quality early
  - Providing a means for incrementally specifying systems with use cases/user stories
  - Support high-fidelity hand off to downstream engineering
  - Defining means for planning and governance of system engineering work
Agile for Embedded Software?

- Embedded software must live within tight resource and performance constraints while delivering predictable and reliable services in cyber-physical systems
- Agile can assist by
  - Improving quality early
  - Providing continuous feedback as to current progress and quality
  - Ensuring system flows with continuous integration
  - Measure true project progress and quality
Agile for Safety-Critical Systems?

- Safety-critical systems identify hazardous conditions and events and both identify requirements for safety measures and provide safety assessments throughout the development process.

- Agile can assist by:
  - Avoiding defects
  - Automatically generating and/or managing needed documentation, such as:
    - Traceability records
    - Test evidence
    - Quality assurance records
    - Change and configuration management records
Safety in a Regulated Environment

- **Initial Safety Analysis**
  - Function, Failure & Safety Information
  - System Requirements & Design

- **System Requirements & Design**
  - Functions & Requirements

- **Hardware Development Life-Cycle**
  - Downstream engineering

- **Software Development Life-Cycle**

**Standards and Regulations**

- IEC 61508: Functional Safety
- IEC 60880: Nuclear
- IEC 60601: Medical
- IEC 62304: Medical
- RTCA DO-178B: Airborne Systems
- ISO 26262: Automotive
- EN 50128: Rail
- SAE ARP 4754: Complex Systems Design
- SAE ARP 4671: Safety Assessment
Some specific challenges of agile methods for Safety-Critical Systems

- Most agile literature is for IT software not embedded software or systems

- Agile de-emphasizes documentation but safety standards require strong documentation

- Agile “travels light” and doesn’t have independent Q/A and V/V

- Agile literature has no discussion of safety, reliability, or security

- Agile requirements are typically very lightweight with no traceability

- Agile testing is continuous but lightweight

- Process must address larger systems context incl
  - Systems Engineering
  - HW/SW co-development

- Use tooling to automatically generate documentation for certification approval

- Process must be documented, with standards (checklists) that can be verified by QA

- V&V testing must generally be done by an independent team

- Process steps must include dependability analysis and assessment

- Requirements must be specified, reviewed and mapped to design, tests and code

- Testing must be thorough and coverage mapped back to requirements
Welcome to IBM® Rational® Harmony™ for Embedded RealTime Development

Harmony for Embedded RealTime Development is a member of the The IBM® Rational® Harmony™ Library of Best Practices specifically for embedded software development.

Harmony for Embedded RealTime Development connects the dots between people, process, tools and best practices to provide a complete solution for embedded software development teams.

The Harmony for Embedded RealTime Development Process is generally applicable to software and systems development, but is optimized for the development of software-intensive real-time and embedded systems. Harmony for Embedded RealTime Development is directly derived from the Rapid Object-oriented Process for Embedded Systems (ROPES), authored by Dr. Bruce Ponder Douglas (DOUG, DOU9, DOU04).

Harmony for Embedded RealTime Development is:

- Agile: embodies the use of key concepts and guiding principles of agile development
- Efficient: emphasis is on tasks and work products that add significant to the development
- High quality: stresses on continual validation of correctness and completeness throughout development
- Requirements-driven: development tasks concentrate on identifying and meeting stakeholders needs
- Architecture-centric: strong concepts of key architectural views
- Scalable: process is defined such that aspects are included only when needed

In addition, Harmony supports specific technologies and needs common to real-time and embedded systems, including:

- Timeliness, schedulability, and performance
- Level-1 (i.e. device level) development
- Early risk reduction
- Safety-critical systems
- High-reliability systems
- Hardware/software co-development

Harmony for Embedded RealTime Development can be used effectively for real-time and embedded (RT&E) systems, technical applications, and systems and software that support RT&E and technical systems.
Agile Systems Engineering Best Practices

- High-fidelity model-based engineering (Hi-MBE)
- Incremental functional analysis with use cases
- Executable requirements modeling with SysML
- Test-driven development of system specifications
- Integrated safety, security and reliability analysis
- Integrated control modeling
- Model-based handoff to downstream engineering
- Automated document generation from model artifacts

Note: a key difference between agile SW and agile SE is that the outcome of SE is specifications and the outcome of SW is implementation
Models and Viewpoints in Model-Based Systems Engineering

- **Functional Model**
  - Executable use cases
  - Functional and QoS requirements

- **Architectural Model**
  - Subsystems, interfaces
  - Subsystem use cases/Requirements

- **Dependability Model**
  - Safety, reliability, security analysis
  - FTA, FMEA, FMECA, Asset Diagram, SAD

- **Control Model**
  - Control algorithms, mathematical models

- **Subsystem Model(s)**
  - Model-based handoff

- **Software Specification**

- **Mechanical Specification**
  - Model and text

- **Electronic Specification**
  - Model and text
## Alternative approaches to Build Executable Model of UC

<table>
<thead>
<tr>
<th>Alternative 1: Scenario Based</th>
<th>Alternative 2: Activity Based</th>
<th>Alternative 3 State Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify sequences of messages between system use case and actors</td>
<td>Identify functional flow large-scale view</td>
<td>Create Model Context</td>
</tr>
<tr>
<td>Sequence Diagram</td>
<td>Activity diagram</td>
<td>Block Diagram</td>
</tr>
<tr>
<td>(optional) cluster sequences together as flows</td>
<td>Derive sequences from functional flow</td>
<td>Create executable state machine</td>
</tr>
<tr>
<td>Activity Diagram</td>
<td>Sequence diagram</td>
<td>State Diagram</td>
</tr>
<tr>
<td>Create model context</td>
<td>Create model context</td>
<td>Identify interfaces</td>
</tr>
<tr>
<td>Block Diagram</td>
<td>Block Diagram</td>
<td>Block Diagram</td>
</tr>
<tr>
<td>Identify interfaces</td>
<td>Identify interfaces</td>
<td>Derive sequences from functional flow</td>
</tr>
<tr>
<td>Block Diagram</td>
<td>Block Diagram</td>
<td>Sequence diagram</td>
</tr>
<tr>
<td>Create executable state machine</td>
<td>Create executable state machine</td>
<td>Execute State machine</td>
</tr>
<tr>
<td>State Diagram</td>
<td>State Diagram</td>
<td>Model execution views</td>
</tr>
<tr>
<td>Execute State machine</td>
<td>Execute State machine</td>
<td>Model execution views</td>
</tr>
<tr>
<td>Model execution views</td>
<td>Model execution views</td>
<td>Repeat until all requirements and sequence variants covered</td>
</tr>
<tr>
<td>Repeat until all requirements and sequence variants covered</td>
<td>Repeat until all requirements and sequence variants covered</td>
<td></td>
</tr>
</tbody>
</table>
Test Driven Development of Executable Requirements
Model-Based Safety Analysis with the UML Profile for Safety
Model-Based Security Threat Analysis

- Security Analysis Diagram* (SAD) is like a Fault Tree Analysis (FTA) but for security, rather than safety
  - It looks for the logical relation between assets, vulnerabilities, attacks, and security violations
  - Permits reasoning about security
    - What kind?
    - How much?
    - Risk assessments

* From the Security Analysis Profile for the UML
Auto-generation of dependability-relevant summary data

• Traceability improves your ability to make your safety/security/reliability case

Dependability metadata guides
- System requirements
- Downstream engineering work
- Regulatory approval submissions
Scenario Driven Use Case Construction / Validation

Making it agile
1. Incrementally specify
   - Use cases/User stories
   - Specification nanocycles
2. Continuously verify
   - Build executable models
3. Frequently validate
   - Demonstrate to customer
4. Identify missing & incorrect requirements
   - Unspecified state transitions
   - Unspecified scenarios
   - Unspecified actions
   - Unspecified qualities of service
   - Unspecified exception/error behavior
5. Incrementally add traceability
6. Integrate with other specifications
   - Other use cases
   - Non-functional work
     - Safety
     - Security
     - Reliability
Model-based hand off to software

- Preserves engineering precision
- Traces to textual requirements
- Specifies
  - Architectural entities & responsibilities
  - Logical interfaces
    - Allocated functionality and QoS
- Hand off outputs
  - Subsystem functional/non-functional specifications
  - Physical interface specifications
  - Allocations to engineering disciplines
  - Specification of interdisciplinary interfaces
    - SW-electronics interfaces
    - Electro-mechanical interfaces
  - Model-environment for down-stream development
Agile Embedded Software Engineering Best Practices

- Analysis-Oriented Practices
  - Initial Safety Analysis
  - Continuous Safety Assessment
  - Executable Requirements Models

- Design-Oriented Practices
  - Model-Based Engineering
  - 5 Views of Architecture
  - Defensive Design

- Quality-Oriented Practices
  - Continuous Execution
  - Test-Driven Development
  - Continuous Integration
  - Incremental Development with the Harmony Microcycle

- Evidence-Oriented Practices
  - Manage Traceability Records
  - Test Coverage Analysis
Harmony Overall Lifecycle

Initial Safety analysis

Ongoing Safety analysis

Incremental Safety design / implementation

Define and deploy the development environment

Prespiral Planning

Develop Stakeholder Requirements

Control Project

Microcycle

Manage Change
Practice: Harmony Iteration Cycle

- **Analysis**: Prototype Definition
  - Defines the properties of all acceptable solutions

- **Design**: Object Analysis
  - Architectural Design
  - Mechanistic Design
  - Detailed Design
  - Prepare for Validation Testing
  - Specifies a particular “optimal” Solution - this is where we add safety architectures, patterns, and idioms

- **Test**: Validation
  - Increment Review ("Party Phase")
  - Validates against prototype mission, including safety and reliability

Typically 4-6 weeks
Practice: High Fidelity Modeling Workflow

Nanocycle
- Identify Objects and Classes
  - [Needs all functional requirements]
  - [else]

High-fidelity Modeling
- Refine Collaboration
- Create Unit Test Plan/Suite
  - [needs refinement]

Continuous Integration
- Make Change Set Available
  - [test passed]
  - [else]

Test-Driven Development
- Execute Unit Test

Code Gen
- Translation
- Execute Model
  - [else]

Debugging
- Factor Elements into Model
Practice: Harmony’s 5 Key Views of Architecture

- Harmony identifies 3 levels of design optimization
  - Architectural
  - Mechanistic
  - Detailed

- Architecture is divided into 5 primary views
  - Each view is characterized by its own set of design patterns, approaches, and technologies
  - Secondary architecture views include
    - Security & Information Assurance
    - Data Management
    - Quality of Service Management
    - Error and exception Management
## Summary

- **Agile methods provide a very disciplined approach to software development**
  - Practices provide guidance on how to achieve work goals and what work products are needed
  - Agile is inherently extensible to meet a variety of needs including safety critical systems

- **Harmony/SE applies agile practices with**
  - Incremental functional analysis with use cases using UML/SysML
  - Integrated safety/security/reliability specification and assessment
  - Model-based hand off
  - Dynamic planning and agile governance

- **Harmony/ESW is a agile process with**
  - Use of UML Modeling to capture application behavior and structure
  - Defect avoidance via a set of best practices
    - *Continuous debugging* - Continual code generation (many times/day)
    - *Test Driven Development* - Continual debugging and unit testing (many times/day)
    - *Continuous integration* (reestablish baseline >=1/day)
  - Incremental Development
  - Agile planning and governance
References