Effective Reduction of Avoidable Complexity in Embedded Systems
Dr. Julien Delange

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Introduction and Background

Embedded Systems are moving towards Model-Based Engineering

A380 control and display system implemented with SCADE
Reduction of development costs as much as 57% for highest criticality levels

Software Complexity spans along the software lifecycle
Impacts development & maintenance activities
Maintenance = 70% of Total Cost

Need for software complexity management
How to detect complexity in models?
How to improve model design?
SEI Approach for Software Complexity in Models

How to identify complexity in software models?
   Application on functional (SCADE) and runtime (AADL) models
   Reuse existing metrics vs. develop model-specific ones?

Understand why/when/how users introduce complexity
   Establish user vision of complexity
   Tool support to detect and automatically avoid complexity

Estimate the cost of complexity for safety-critical systems
   Impact on development and maintenance activities
Defining and Detecting Software Complexity

Define inappropriate modeling patterns & complexity metrics

Runtime models (AADL): complexity in configuration and deployment
Functional models (SCADE): complexity in software implementation

Identify root causes of high complexity

Software re-use (aka copy/paste), data scope, etc.
How to address and fix it (e.g. re-factoring, resources allocation)

Application to existing models

Stepper Motor (Rolls-Royce/AEC)
Flight Control Guidance (SCADE)
Software Architecture Complexity (AADL)

Resources usage/dimension, configuration issue
- Two inter-dependent subprograms deployed on different processors
- Communication queues dimensions wrt timing requirements
- Inconsistent timing requirements with connected component

Suggest architecture changes
- Change deployment strategy (e.g. relocate a task)
- Modify timing requirements/communication policy

Tested on industrial example
- Discover timing issue from industrial models
Software Functional Complexity (SCADE)

Tailor existing metrics to a modeling language

- System States (cyclomatic)
- Operators and Operands (Halstead)
- Connectedness (Zage)

Develop new model-specific metrics

- Data flow oriented (#operator, #output per flow)
- Help to reason about the impact of a flow
- Provide hints to re-architect the software architecture
Part 1: Model Review and Tool Usage Critique
   Existing model to review (elevator model)
   Evaluate the modeling tool (SCADE)

Part 2: Model Design
   Implement a system (microwave) from a textual specification

<table>
<thead>
<tr>
<th></th>
<th>Student</th>
<th>Professional</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Experience</td>
<td>Group 1</td>
<td>Group 2</td>
</tr>
<tr>
<td>Moderate Experience</td>
<td>Group 3</td>
<td>Group 4</td>
</tr>
</tbody>
</table>

Understand Model Complexity
Modeling Tool Experiment - Results

Basic complexity issues are not understood
- **50%** of participants have issues with data abstraction
- **50%** of participants have issues with data scope

Appropriate training is key when transitioning to a MDE approach
- **25%** of users experienced comprehension and understanding issues
- **80%** of participants did not find any comments (and there were some!)

Experience level does not explain performance
Estimate Cost Savings

Overall impact of Model-Driven Engineering on software quality

Automatic production of certified code
Testing and Validation activities, especially for DO178-C
Software reuse and maintenance

Impact of Models Complexity

Limited in development efforts
Significant in maintenance costs

Model-Driven Engineering with managed complexity reduces up to 30% of total TCO

Adapted from incoming SEI technical report
“Evaluating and Mitigating Complexity in Software Models”
Conclusion and Perspectives

Impact of complexity management with Model-Driven Engineering

Costs savings of more than 30% on the total life cycle
Deliver better quality, faster at an affordable cost
Require **appropriate training** and understanding of modeling techniques

Need to refine complexity metrics on models

Calibration on several models
Estimate potential savings per metrics

Transition and impact of SEI research

Release of tools, integration in modeling software
Research results published in an SEI technical report, blog and podcast
Benefits for other domains (e.g. medical, automotive, etc.)
Contact Information

Julien Delange
Member of the Technical Staff
Software Architecture Practice
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.sei.cmu.edu
www.sei.cmu.edu/contact.cfm
https://github.com/cmu-sei/eraces

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