Flight Software Process Gap Analysis for Model Based Development

MINI-WORKSHOP ON
1ST INTERNATIONAL WORKSHOP ON SOFTWARE HEALTH MANAGEMENT (SHM 2009)
http://www.isis.vanderbilt.edu/workshops/smc-it-2009-shm

Submissions:
Anticipated submissions are brief position papers (extended abstracts) and not more than 3 pages long.

Topics of interest include, but are not restricted to:
- Foundations and principles of software health management
- Software anomaly detection
- Fault isolation for software
- Fault mitigation techniques, ranging from pre-defined to fully autonomic
- Early, promising technical directions and results
- Challenge problems from real-life systems
- Novel research directions and approaches, including model-based techniques

Luis C. Trevino, Ph.D.
The Charles Stark Draper Laboratory
Central Aim of Gap Analysis

Focus is on Design & Code using Model Based Methods for Software Health Management

GNC (ex.) → Preliminary Requirements Analysis → Software Requirements → Design → Code → Certification Process

Gap Analysis

Discovery

Guidelines

Roundtrip Engineering
Unit Testing
Peer Reviews

GNC
- Kalman Filter
- Controls Analysis
- Equations
- Modeling

Preliminary Requirements Analysis
- Analysis
- Text Base
- Notation based on Discrete Math & Traceable to Logical Reasoning - (special domains)

Software Requirements
- Baselined Text Docs

Design
- SDD
- MBM
- Model Checkers (e.g., Spin)
- Theorem Proving

Code
- Code Generation
- Static & Dynamic Analysis Tools
- Hybrids

Certification Process

Gap Analysis

Guidelines

Roundtrip Engineering
Unit Testing
Peer Reviews
Software Health Management and Model Based Methods

- **OMG UML Profile for Modeling Quality of Service & Fault Tolerance Characteristics & Mechanisms Specification (Version 1.1, April 2008).**
- Suggests some UML extensions to these areas
  - QoS: Fault & Failure Characteristics
  - FT: Focus is on Technical Solutions in UML software architectures
    “The Goal of [software] fault tolerance methods is to include safety features in the software design or Source Code to ensure that the software will respond correctly to input data errors and prevent output and control errors. The need for error prevention or fault tolerance methods is determined by the system requirements and the system safety assessment process.” [22]
- Policies and Profiles for MetaModels for Fault Tolerant Core, Fault Detection, Object Group Properties, & Replication Styles
  - Fault Detector Deployment Policy addresses how to monitor software faults: e.g., Software Fault Detectors (for HM) could be implemented as daemon processes that are installed with the FT infrastructure and registered in a manner internal to the FT infrastructure, allowing the infrastructure to include them in every fault-tolerant application within the fault tolerance domain in a transparent manner.
    - Addresses stuff like dangling pointers, buffer overflow in heap & stack, null pointer dereference, pointer arithmetic, dead & deactivated code, loop control variable, unchecked array indexing, pre-processor directives, pointer casting
    - Equally applicable to generated code
Software Health Management “Type” Initiatives for Ares 1 US

- Three Flight Computers within Instrument Unit for integrated Ares 1 automated/autonomous control/monitor
  - FCOG
  - FC: Byzantine Fault Resilient 3 String Voting Architecture

- Arinc 653 Compliant Partitioned Environment - has its own Health Monitor specifications for process, partition, and modules, specifically for Error Levels, Fault Detection and Response, and Recovery Actions

- Software Common Cause Failure Trade Study
  - Focus on mitigation of/Protection against software common cause failures (for Flight Computer). Fault tolerance & Trade Tree developed.
  - Other NASA related fault tolerant studies identified (methods & architectures in back up slides with reference)
# Gaps - Challenge Problems for Model Based Methods for Software Health Management

<table>
<thead>
<tr>
<th>Gap (Concern)</th>
<th>Category-Issue</th>
<th>Reference Process</th>
<th>Notes</th>
<th>More Notes &amp; Applicable Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstraction interpretation</td>
<td>Design &amp; Abstraction</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of detail</td>
<td>Abstraction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domain Specificity-Applicability</td>
<td>PSM</td>
<td>2,3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code generation &amp; Specification</td>
<td>DSL scaling &amp; analysis</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sources for Language-Domain Specific Language (DSL)</td>
<td>DSL</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Limit Expressiveness</td>
<td>domain interpretation</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Notation</td>
<td>domain interpretation</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphical-vs-Textual Notation</td>
<td>development</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Semantics</td>
<td>domain interpretation</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gap (Concern)</th>
<th>Category-Issue</th>
<th>Reference Process</th>
<th>Notes</th>
<th>More Notes &amp; Applicable Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viewpoints</td>
<td>domain interpretation</td>
<td>3</td>
<td>Studies</td>
<td></td>
</tr>
<tr>
<td>Partitioning</td>
<td>development</td>
<td>3</td>
<td></td>
<td>partitioning not completely transparent in many tools avoid accidently creating unintended dependencies on other partitions</td>
</tr>
<tr>
<td>Evolution</td>
<td>language</td>
<td>3</td>
<td></td>
<td>If language needs to be changed, have a way of adapting model processors including existing models caution: don’t shoehorn domain concepts into existing language two different styles of language see sufficient far from sufficient existing language libraries see embedded comment</td>
</tr>
<tr>
<td>Fallacy of Generic Languages</td>
<td>language</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Who are the First Class Citizens</td>
<td>language design</td>
<td>3</td>
<td>design</td>
<td></td>
</tr>
<tr>
<td>Libraries</td>
<td>development</td>
<td>3</td>
<td></td>
<td>constraints required to validate models putting constraint checks in templates is bad, overly complicated</td>
</tr>
<tr>
<td>Checks First &amp; Separate</td>
<td>development</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Don’t Modify Generated Code</td>
<td>development</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control Manually Written Code</td>
<td>development</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Care About Generated Code</td>
<td>development</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Make the Code True to the Model</td>
<td>development</td>
<td>3</td>
<td>code generation</td>
<td>Maintain manual coding standards</td>
</tr>
<tr>
<td>Viewpoint-Aware Processing</td>
<td>development</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Gaps - Challenge Problems for Model Based Methods for Software Health Management

<table>
<thead>
<tr>
<th>Gap (Concern)</th>
<th>Category-Issue</th>
<th>Reference Process</th>
<th>Notes</th>
<th>More Notes &amp; Applicable Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Care About Templates</td>
<td>development</td>
<td>3 generation</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>Use M2M Transformations to Simplify Generation</td>
<td>development</td>
<td>3 generation</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>Allow for Adaptations</td>
<td>development</td>
<td>3 generation</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>Cascading</td>
<td>development</td>
<td>3 development</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>Annotation Models</td>
<td>development</td>
<td>3 development</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>Classify Behavior</td>
<td>development</td>
<td>3 design</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>Don’t forget Testing</td>
<td>testing</td>
<td>3 testing</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>Iterate!</td>
<td>development</td>
<td>3 testing</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>Co-Evolve Concepts &amp; Language</td>
<td>development</td>
<td>3 preliminary</td>
<td>design, trade, &amp; feasibility</td>
<td>in work</td>
</tr>
<tr>
<td>Documentation Is Necessary</td>
<td>development</td>
<td>3 development</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>Reviews</td>
<td>development</td>
<td>3 process</td>
<td>improvement</td>
<td>in work</td>
</tr>
<tr>
<td>Let People Do What They Are Good at</td>
<td>development</td>
<td>3 process</td>
<td>improvement</td>
<td>in work</td>
</tr>
<tr>
<td>Domain Experts Programming</td>
<td>development</td>
<td>3 development</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>Metaware as a Product</td>
<td>development</td>
<td>3 development</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>Compatible Organization</td>
<td>development</td>
<td>3 process</td>
<td>improvement</td>
<td>in work</td>
</tr>
<tr>
<td>Forget Published Case Studies</td>
<td>development</td>
<td>3 development</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>XML - XMI issues (Lou)</td>
<td>development</td>
<td>3 development</td>
<td>in work</td>
<td></td>
</tr>
<tr>
<td>Modeling Checking (Lou)</td>
<td>development</td>
<td>3 in work</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process &amp; Organization (Lou)</td>
<td>development</td>
<td>in work</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML - Class Diagram attributes &amp; properties</td>
<td>design &amp; development</td>
<td>see embedded comment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XMI-XML Metadata Interchange Format</td>
<td>design &amp; Abstraction</td>
<td>1 design &amp; development</td>
<td>see embedded comment</td>
<td></td>
</tr>
</tbody>
</table>

## Commercial Code Generators

- Code generation: design & development
- Note: includes and using namespace not automatically generated lots of virtual memory
Summarizing The Literature Analysis

• Many papers were reviewed
• An equal amount identified for further review
• Common themes
• Gap.xls
Conclusions: Forthcoming Activities

- Will continue researching case studies and reliable application of generated code from model based tools (e.g., SEI - Multi Aspect Model Repository & Model Bus, Modeling of Fault Tolerant Configurations)
- Will continue research case studies & tools for model checking (i.e., of UML, e.g., OCLE & USE - structural properties of UML) and populating the spreadsheet
- Explore Architectural Description Languages for Fault Tolerant UML Design for Fault Tolerant Applications (e.g., [9])
- Continue identifying gaps in selected reference material below
- Continue investigation of fault detection and mitigation methods and applicability to software health management
- Will be working further with Cambridge-Draper software working group
References


[12] Integration between AbsInt’s aiT and StackAnalyzer Tools white paper, Esterel Technologies (SCADE Mission & Sactety), 1/31/08


* References [11] through [21], already read but needs further examination
Back Up Slides
Fault Tolerant Methods

- **Horizontal Partitioning**
  - “Horizontal partitioning separates the major software functions into highly independent structural branches communicating through interfaces to control modules whose function is to coordinate communication and execution of the functions.”

- **Vertical Partitioning**
  - “Vertical partitioning (or factoring) focuses distributing the control and processing work in a top-down hierarchy, where high level modules tend to focus on control functions and low level modules do most of the processing.”

- **Atomic Action**
  - “An atomic action among a group of components is an activity in which the components interact exclusively with each other and there is no interaction with the rest of the system for the duration of the activity [Anderson 81].”

- **Timing Checks**
  - “Timing checks are applicable to systems and modules whose specifications include timing constraints, including deadlines.”
  - Look for timing that is outside these specifications
  - Watchdog timer
  - ARINC 653 process timing checks

- **Encoding**
  - “Coding checks use redundancy in the representation of information with fixed relationships between the actual and the redundant information.”
  - CRC, for example

- **Structural Checks**
  - Structural checks use known properties of data structures.

- **Inversion**
  - “Reversal checks use the output of a module to compute the corresponding inputs based on the function of the module.”
Fault Tolerant Methods

Data Diversity Models – I
- Input
- Re-Expression
- Selection
- Execution
- Error
- Retry
- Output

Data Diversity Models – II
- Input
- Re-expression
- Program Execution
- Output

Recovery Block Model
- Input
- Checkpoint Memory
- Checkpoints
- Primary Version
- Alternate Version
- Selection Switch
- Acceptance Test
- Output

N-Version Programming Model
- Version 1
- Version 2
- Version n
- Selection Algorithm
- Crew

N Self-Checking Programming Using Acceptance Tests
- Version 1
- Version n
- Acceptance Test 1
- Selection Logic
- Output

Consensus Recovery Block Model
- Version 1
- Version 2
- Version n
- Selection Algorithm
- Failure
- Switch
- Crew

Process Pairing Model
- Primary Processor
- Secondary Processor
- Selection Switch
- Error Detection
- Output

N Self-Checking Programming Using Comparison
- Version 1-A
- Version 1-B
- Version n-A
- Version n-B
- Comparison
- Selection Logic
- Output