

Software Engineering of Critical Software Tools

Kevin J. Sullivan
presented by David Coppit

Department of Computer Science
University of Virginia, Charlottesville VA, USA
<http://www.cs.virginia.edu>
{sullivan|coppit}@virginia.edu

Audience

- Knowledgeable in modeling and analysis
- Need supporting software tools
- Not trained in software engineering
- Need introduction to key relevant issues

Expectations

- Software systems among the most complex ever built
- The problem is like the werewolf that can't be killed
- There is "No Silver Bullet" to slay the beast -- Brooks
- There is no royal road, but there is a road -- Brooks
- Software engineering is now recognized as critical
- Real software engineers are trained expert professionals
- No one becomes a professional quickly or informally
- This tutorial is not a proven, general "methodology"
- Expect an introduction to a set of critical S.E. issues

Problem

- Tools that analyze critical systems are critical components
 - Therefore such tool should be dependable
- Need tools with high usability, interoperability, low cost
 - Therefore tools are low cost, complex software systems?
- How can we get dependability + complexity + low cost?
- The problem is a software engineering problem

Objectives

- Communicate notion that tools can be critical components
- Communicate importance of rigorous software engineering
- Introduce a set of critical software engineering concepts
- Make concepts concrete by relating them to prototype tool

Outline

- Design tools for critical systems are critical
- Quality of such scientific tools appears to be low
- Need rigorous software engineering of critical tools
- Galileo dynamic fault tree tool as case study in tool design
- Introduction to a set of especially important concepts

Critical Concepts

- Software architecture
- Formal description
- Object-oriented design
- Component-based design (time permitting)

Critical System Design Tools Are Critical

- Engineers rely on software tools to analyze critical systems
- Important design decisions can be based on tool results
 - risk of erroneous design decisions for critical systems
 - risk of sub-optimal design decisions for critical systems
- Modeling & analysis tools are critical system components
- Sophisticated modeling tools are complex software systems
 - run on complex, modern, commodity platforms (e.g., PC/NT)
 - have many sophisticated non-modeling functions (e.g., cut-&-paste)
- Complex software is prone to many errors and difficulties
 - extremely costly to build and maintain
 - errors in specification and implementation
 - poor usability and interoperability properties

Are Scientific Tools Dependable Today?

- Divergences in function actually computed by ostensibly identical seismic analysis tools [Hatton 1994]
- Divergences in function of seemingly equivalent program analysis (call graph extraction) tools [Murphy et al., 1998]
- Nuclear Regulatory alert on errors in critical finite element analysis tools used in designing pressure vessels [NRC 1996]
- The data points seem to be adding up to a consistent story

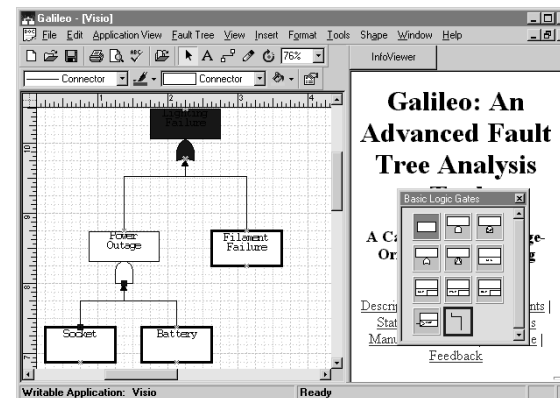
Tools Usable, Interoperable, Low Cost?

- Need desktop tools, but many tied to mainframes, Unix
- High usability, low training costs for sophisticated tools require costly usability engineering
- Interoperability with PC tools, engineering environment demands specialized and very costly program design
- A fault tree tool we know is about a million lines of code; estimated 2/3 for non-modeling issues; cost millions of \$

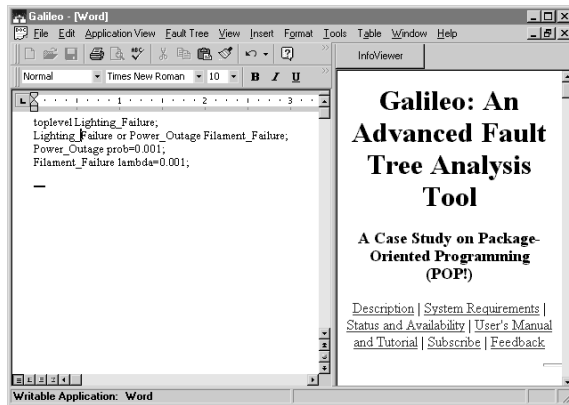
Galileo Tool

- Research prototype distributed to many industrial sites
- Supports Dugan's hierarchical dynamic fault tree analysis
- Function, usability, interoperability, dependability, cost
- Installs easily and runs on PCs with Windows 95/98/NT
- Available at <http://www.cs.virginia.edu/~ftree>
- We have not yet achieved all objectives in all dimensions

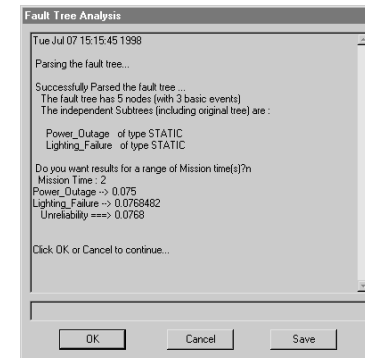
Galileo Graphical View



Galileo Text View



Galileo Analysis Window



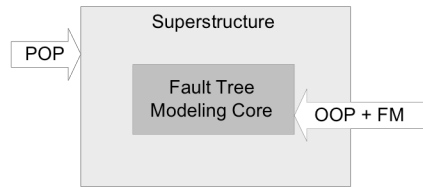
Software Engineering Aspects of Galileo

- Software architecture uses object-oriented and component-based design, emphasizing abstraction & large-scale reuse
- Partial formal description of core modeling framework
- Object-oriented design of core modeling framework
- Component-based design of tool superstructure

Software Architecture

- As software development problems increase in complexity, programming becomes less critical, and architecture more
- Architecture is decomposition of system into components that are composed by connectors under certain constraints
- Design rules, the satisfaction of which is intended to lead to certain defined, desirable performance outcomes
- Many projects come to grief by focusing too early on programming while ignoring critical architectural issues

Very High-Level Architecture of Galileo



Formal Description

- When modeling framework is well understood challenge is to build and verify the dependability of an implementation
- However, Hatton's data suggest that this isn't so easy
- When model framework is innovative and subtle, another class of error modes is present: conceptual error in design
- Precise descriptions of framework are needed to reason about the conceptual design, to specify work to be done by the software designer, and as basis for user documentation
- Natural language and informal graphical descriptions (e.g., UML) are prone to incompleteness & ambiguity, thus error
- Need mathematical descriptions as used routinely in many

Formal Description

- Many mathematical frameworks provide bases for formal (mathematical) descriptions in software engineering, e.g., predicate logic & set theory, temporal logics, trace algebra
- Predicate logic + set theory is an especially important one
- They provide an assembly language of formal description
- Structuring and abstraction mechanisms needed to build large, complex formal descriptions, such as system specifications
- Software engineering notations for formal specification, such as Z, have been designed to meet this need

Example: Gate

```

úy Gate yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
ú
úFaultTreeNode
úinputs      : ç FaultTreeNodeType
úoutput      : FaultTreeStatus
úfailTime    : £
úyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
úSumNonDummyInputs(inputs) µ 0
úyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
    
```


Object-Oriented Class Inheritance

```
Class FaultTreeNode {  
    virtual void Print();  
    ...  
};  
Class Gate : FaultTreeNode {  
    void Print();  
    ...  
};  
Class BasicEvent : FaultTreeNode {  
    ...  
};
```

Polymorphism

```
Class FaultTree {  
    private:  
        set<FaultTreeNode> nodes;  
    ...  
};  
FaultTree::Print() {  
    foreach n in nodes {  
        n.Print();  
    }  
}
```

Component-Based Design

- Buy, don't build! --Brooks
- No single design authority
- How to assure that components can be integrated?
- Integration architectures -- design rules, such as COM
- Galileo exploits Active Document Architecture

Some Benefits Achieved

- Product line architecture for reliability modeling
- Modeling framework formal description and clean design provide a basis for future progress toward dependable core
- Rich function, high usability & interoperability, low cost achieved through component-based design with large-scale components
- Low cost for users to learn to use and to use the tool

Conclusion

- Scientific problem of crafting innovative modeling and analysis techniques, such as Dugan's modular hierarchical dynamic fault tree analysis technique
- Software engineering problem of providing cost-effective tool support
- Software modeling tools for critical systems are complex and critical themselves, and therefore warrant a rigorous software engineering approach to their design
- We have discussed several elements of such an approach