Model-Based Development Research at ISU

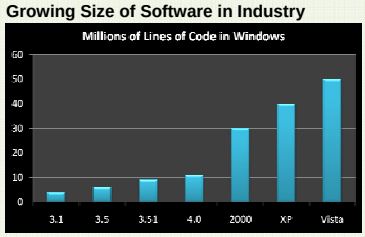
Software Reliability

Expansion of Capabilities

F-4 (1960): 8% of capabilities provided by software

F-22 (2000): 85% of capabilities provided by software

Growing Size of Software in Industry



Importance of Software Verification

Mars Polar Lander – Powered down 100ft above the Martian surface – most likely a software bug.

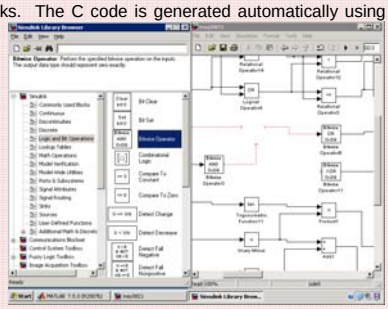
Complexity of software grows – reliability an important issue. Infamous Ariane 5 disaster, arguably one of the most expensive software bugs in history.

Problem Statement

Developing tools and technology for producing safety-critical software.

This research is aimed at developing a new tool to facilitate efficient development of control software for safety-critical applications. Like Computer-Aided Design (CAD), Model-based development (MBD) tools offer a graphical programming environment where the user develops the program as a graphical model from which the code is generated automatically. Our research focuses on Simulink, a tool by The MathWorks. The C code is generated automatically using another tool called Real-time Workshop.

- The generated code must be checked to ensure correct translation
- Existing avionics systems have as many as 750,000 lines of code.
- Develop an automated tool for auditing
  - Reconstruct the model from the generated code,
  - Compare extracted model and original with graph differencing tool (existing grad student project)



Model-Based Development in Simulink

Requirements

Functional Requirements

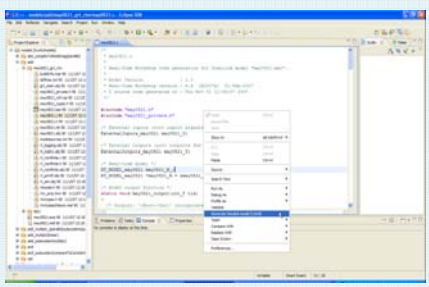
•Parse C code generated from Simulink model by Real-Time Workshop
•Produce a graph representation comparable to the original model
•Allow easy extensibility to support additional Simulink blocks

Non-Functional Requirements

•Usable on any system running the Eclipse Platform
•Process large model with over 100 blocks in under a minute

Deliverables

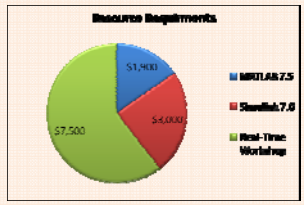
•Requirements Specification
•Engineering Specification
•Source Code
•Project Poster
•Project Plan
•Design Documents
•User Manual
•Website

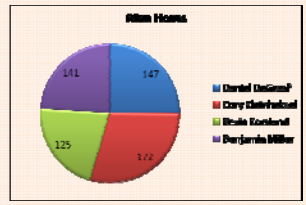


Project Plan

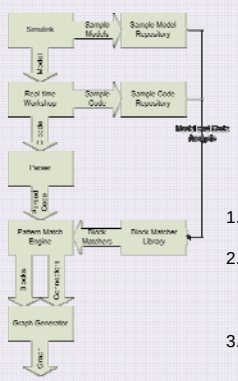
Work Breakdown

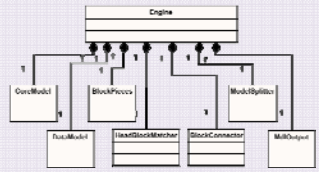
1.Create Simulink Models and C Code
2.Manually Identify Simulink Blocks In C Code
3.Write Code to Obtain an Abstract Syntax Tree
4.Write Code to Detect Blocks
5.Write Code to Output to DOT File
6.Verify Generated Model is Functionally Equivalent
7.User Interface
8.Testing and Debugging





Design Overview





1. Code segmentation – create segments that can be matched with code patterns.
2. Matching – developed a block matcher to perform the matching. The block matcher is written to handle possible mutations of the code, such as the substituting of sub-expressions for an input variable.
3. Linking – link the matched blocks to reconstruct the model.

Test Results

White Box Testing

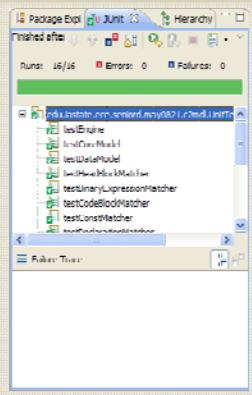
•Testing of individual methods and classes
•Uses JUnit framework

System-Level Testing

•Testing of system as a whole
•Input is a single set of files
•Intermediate output from each stage can be saved
•The source of a bug can be tracked to the component

Test Results

•Revealed a number of bugs in the code
•Identified blocks which are not supported by our code
•Test succeeds in all models using only supported blocks



JUnit Test Results

Conclusion

Our project
•demonstrates that it is possible to automatically test generated code for errors.
•is capable of matching nearly all types of blocks.
•could be developed into a robust commercial solution to save test engineers a great deal of time when auditing automatically generated code.

Faculty Advisor
Dr. Suraj Kothari

Team Members
Daniel De Graaf
Cory Kleinheksel
Kevin Korslund
Benjamin Miller