Software Productivity and Reliability
Tools and Techniques

Microsoft Presentation, February 5, 2007

S. C. Kothari
Iowa State University & EnSoft Corp.
Phone: 515-294-7212
Email: kothari@iastate.edu
SimDiff 2.0
Automatically find differences between Simulink models.
Try it out! Request a trial of SimDiff.

New in 2.0: Stateflow and TargetLink Support.

Rubric
Rubric checks C/C++ code for compliance to your coding standards. Rubric identifies violations of conventions such as:

- Names of variables must begin with a lower case letter.
- Switch clauses must align with a line.

SimDiff
SimDiff compares two Simulink model files for additions, deletions, and changes to:

- blocks
- lines
- configuration parameters
- model properties
- annotations

Total Insight
COBOL developers spend substantial time finding program, file, SQL, and data interactions. COBOL Total Insight's easy-to-use interface instantly reveals interactions in multi-million line codes.

Products
- Rubric
- SimDiff
- Total Insight

By Platform
- Simulink: SimDiff
- C/C++: Rubric
- COBOL: Total Insight
Bygone Era

...New Reality
The IBM 360 finally reached the Palouse wheat fields of Eastern Washington. It cost the university two million dollars in 1966. I began using it and the world opened up. Right away, I solved a differential equation numerically and found myself staring at a Butterfly Effect. It’d be decades before that term entered our vocabulary. These new machines led us into a brave new world.

– from *Engines of Our Ingenuity* by John H. Lienhard
IBM 360

Figure 1. IBM System/360 Model 40 Data Processing System
Operating System/360

- The not-unexpected passing away of OS/360 in its 21st release – August 2, 1972.
- Obituary:

  The offspring first saw the light of day in December 1965 and the birth announcement recorded a weight of 64K. It rapidly became apparent that OS, in spite of its unusual size, was more than normally subject to childhood diseases. For a long period, this weak and sickly baby hovered close to death despite almost continuous transformations and major transplants of several vital organs. Many experts are of the opinion that the huge weight of OS at birth contributed greatly to its early ill health. OS is survived by two lineal descendants, OS/VS1 and OS/VS2. It will be mourned by its many friends and particularly by the over 10,000 system programmers throughout the world who owe their jobs to its existence.
Bygone Era

- Expensive computers.
- Limited CPU power and small memory.
- Computer’s time more valuable than programmer’s time.
New Reality

- *Human Complexity* – programmer’s time and effort – now far more important and expensive than the computer.

- *Human Complexity* is the bottleneck to achieving significant improvements in software productivity and quality.
Software evolves ...
History of Microsoft Windows

November 10, 1983, Microsoft announced Microsoft Windows®, an extension of the MS-DOS® operating system.

This marked the beginning of the graphical user interface (GUI) era at Microsoft.

Vista
Boundaries of our technologies?
Testing

- Testing - laborious and unreliable to guard against defects due to *unspecified behaviors*.
- A brute-force coverage of all paths is not possible – the number of execution paths is typically an astronomically large number.
- A test run implies traversing of one execution path at a time – defective paths may not be covered during the test runs.
Software Metrics

- The quantification problem is wide open.
- History tells us that it may be many years before we solve the problem.
- There were 35 different temperature scales before the Celsius.
- Starting with the earliest paper in 1968, we have more than 1000 software metrics – it will be a long time before we have metrics that we can agree on.
Documentation

- One guarantee we will always have – documentation of any useful software will never be complete.
- One guarantee we will never have – the documentation will always be consistent with the software.
Program Writing Technologies

- Examples: Programming languages, components and libraries, design patterns..
- For writing new software: Program writing technologies have advanced a lot, but their focus is primarily on implementation rather than design and prototyping; they tend to obfuscate the domain knowledge.
- For evolving existing software: still a long way to go ...
Software production

... a lot more besides writing it
Software – besides writing it

- Maintenance and evolution of software typically accounts for 65 to 70 percent of the total effort.
- Inspection and certification – critical for safety and mission critical applications.
- Reliability of software – the need to understand and address many complex issues.
- Preservation, propagation, and application of domain knowledge – critical for improving productivity and quality of software.
In an operating system course project, a student spends:

- 40 hours in identifying and understanding the relevant parts of code.
- 2 hours in making the actual code changes to incorporate the specified functionality.
- 10 hours in testing and debugging the code.

The required code changes are small and the bugs are usually due to incomplete or incorrect understanding.
Inspection and Certification

- Coding standards:
  - The MISRA C for automotive industry
  - Ellemtel C++ for telecommunication industry
- DO-178B – the standard for safety-critical avionics software.
Software Reliability

- A hard problem because of the intrinsic domain complexity, the cost-performance tradeoffs, an ever broadening scope of applications, and the ever-changing implementation technologies.
- Critically important due to safety, security, and financial risks associated with software failures.
Domain Knowledge - The Key

- Imagine how difficult it would be to understand OS code without knowing the underlying OS design principles.

- Productivity and quality depends on:
  - *Preservation of knowledge* – withstanding the long software lifecycles and high turnover of personnel.
  - *Propagation of knowledge* – training new recruits.
  - *Application of knowledge* – bridging the gap between the domain knowledge and the code.
Program Reading

.. It deserves attention
Program Reading – A Critical Need

- Program Reading – analyzing, reasoning, and auditing software – overlooked by our educational system.
- We only have rudimentary program reading technologies – tools such as the open source Cscope.
Program Reading – It is not just for the developers

- Beyond developers, program reading can benefit managers.
- Get a survey of the software artifacts and their interrelationships – make informed decisions.
- Get an audit report – check compliance with quality requirements.
Program Reading

... It is not easy
Program reading is not easy

- Program reading is often difficult and challenging.
- Non-localized relationships between software artifacts.
- Complexity of semantic analysis.
- Lack of domain knowledge or the difficulty of applying it.
Cross-cutting Relationships and Complex Semantics

- As an example, we will present a series of semantically more complex matching pair (MP) defects.
- Basic pattern: to be correct, a program must have matching pairs of artifacts (e.g. parentheses) or events (e.g. locking and unlocking).
First Model of MP Defects

- Must have matching pairs of syntactic patterns – for example, parentheses.
- Must analyze *syntax* to find defects.
Second Model of MP Defects

- Must have matching pairs $f$ and $f^{-1}$ along each execution path.
- Must analyze the control flow to find defects.
Third Model of MP Defects

- Must have the same $id$ associated with $f$ and $f^{-1}$.  
- The $id$ gets passed through tokens.
- Must analyze the control and data flow to find defects.
Fourth Model of MP Defects

- $f$ and $f^{-1}$ may occur on disconnected paths due to concurrency and interrupt processing.

- Must analyze the control flow, data flow and disconnected execution paths.
Watch one non-defective path.

Matching Pairs

Watch a scenario that makes the second path also a non-defective path.

Disconnected execution path

Shared pool of tokens
Observations

- Compilers cannot catch the defects beyond the first model, because they are limited by the kind of static program analysis they can perform.
- The static program analysis required for an accurate solution for the second and third model is known to be intractable (Rice-Myhill-Shapiro theorem, 1953).
Program Reading Technologies

... Wave of the future
Program Reading Technologies

- Goal: Provide tools to assist with the reading of complex programs.
- Tools will address specific needs: program understanding, smart testing, defect analysis, impact analysis, inspection, auditing ...
“Assigning more programmers to a project running behind schedule will make it even later.”
Fred Brooks – A Legend in Computing

- Architect of the IBM 360 OS.
- Author of the classic - *The Mythical Man-Month: Essays on Software Engineering*.
Tools to amplify human intelligence

Fredrick Brooks:

“... IA > AI, that is, that intelligence amplifying systems can, at any given level of available systems technology, beat AI systems. That is, a machine and a mind can beat a mind-imitating machine working by itself.”
Program Reading Technologies

... Work at ISU and EnSoft
Knowledge-Centric Software (KCS) Tools

- **Goal:** Enable program reading to preserve, propagate, or apply domain knowledge.
- **KCS tools automate execution of intelligent program reading strategies:**
  - User designs the strategies.
  - Tools help in executing the strategies by automating extraction and modeling of information about software artifacts and their relationships.
Query-Model-Refine (QMR) Approach

- A natural way to *amplify human intelligence* by assisting in:
  - Retrieval of information by analyzing software.
  - Generation of visual models from the retrieved information.
  - Refinement of the models to manage complexity.
Tools Technology

- **eXtensible Common Intermediate Language (XCIL):** A language for referring to program artifacts – makes it easier to develop tools for different languages (Ada, COBOL, C, C++, FORTRAN, and Java).

- **eXtensible Pattern Specification Language (XPSL):** A language for referring to relations between program artifacts – makes it easier to develop tools for different domains (control software, systems software, numerical software, business software).
Program Reading Tools

… A Trailer
An Example of Architecture Extraction

- Extracting the architecture of XINU networking code.
- A model and two successive refinements:
  - View I: Call graph (CG) as a model.
  - View II: CG after one refinement.
  - View III: CG after the second refinement.
The Call Graph Model
After one refinement
After the second refinement
Program Reading Tools

… demos and discussions
Program Reading Tools From EnSoft
SimDiff – A Model Differencing Tool
Atlas – A Program Mapping Tool
References

- MISRA C Coding Standard:
  - [http://www.knosof.co.uk/misracom.html](http://www.knosof.co.uk/misracom.html)
- Ellemtel C++ Coding Standard:
  - [http://www.doc.ic.ac.uk/lab/cplus/c++.rules/](http://www.doc.ic.ac.uk/lab/cplus/c++.rules/)
- Avionics software standard:
- CsScope:
- Fredrick Brooks, *The Computer Scientist as Toolsmith*,
- Earliest paper on software complexity:
- Knowledge-Centric Software Research Laboratory at ISU
  - [http://dirac.ece.iastate.edu/sec/](http://dirac.ece.iastate.edu/sec/)
- EnSoft Corp.