

2003 FAA National Software Conference

Tutorial on Static Verification

Tutorial on Static Verification

Rod Chapman
Praxis Critical Systems
rod.chapman@praxis-cs.co.uk

John Joseph Chilenski
Boeing Commercial Airplanes
john.j.chilenski@boeing.com

September 17, 2003

Agenda

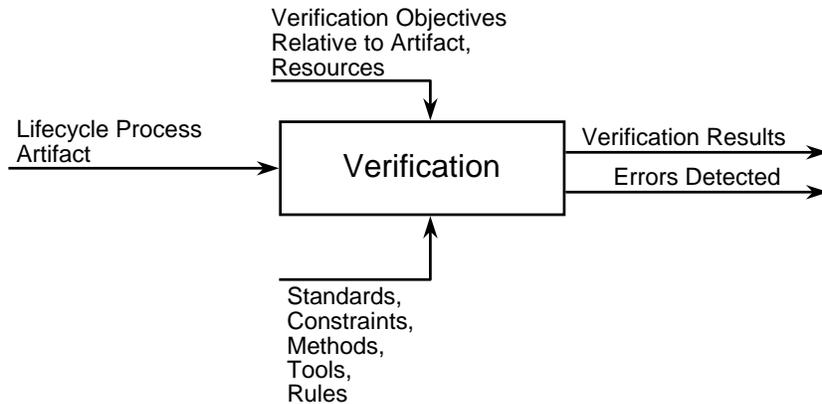
- **Introduction**
- The Catch
- Static Verification and DO-178B Objectives
- Types of Extended Static Verification
- Some Static Verification Languages and Tools
- Static Verification Projects
- Conclusions

2003 FAA National Software Conference

Tutorial on Static Verification

Introduction (1 of 10)

- Verification – The evaluation of the results of a process to ensure correctness and consistency with respect to the inputs and standards provided to that process. [DO-178B Glossary]



Page 3

Change Identifier in View - Header and Footer

Introduction (2 of 10)

- Verification objectives are satisfied through a *combination* of Reviews and Analyses.
- Review – Provides a qualitative assessment of correctness.
- Analysis – Provides repeatable evidence of correctness.
 - Static – Evaluation of a component based on its form, structure, content or documentation.
 - Note that the form, structure or content can be modeled.
 - Models can be informal or formal (mathematically based).
 - Properties about the models can be approximate or exact.
 - Exact properties imply Approximate ones.

Page 4

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Introduction (3 of 10)

- Analysis – (continued).
 - Dynamic (aka Testing) – Evaluation of a component based on its behavior during execution of test cases against the implementation in the target environment, or a high-fidelity simulation of the target environment.
 - Note that DO-178B partitions the testing activity.
 - Test Preparation (Static)
 - Test Execution (Dynamic)
 - Also note that DO-178B asks for Reviews and Analyses of the artifacts of Test Preparation and Test Execution.
 - Verification of verification

Page 5

Change Identifier in View - Header and Footer

Introduction (4 of 10)

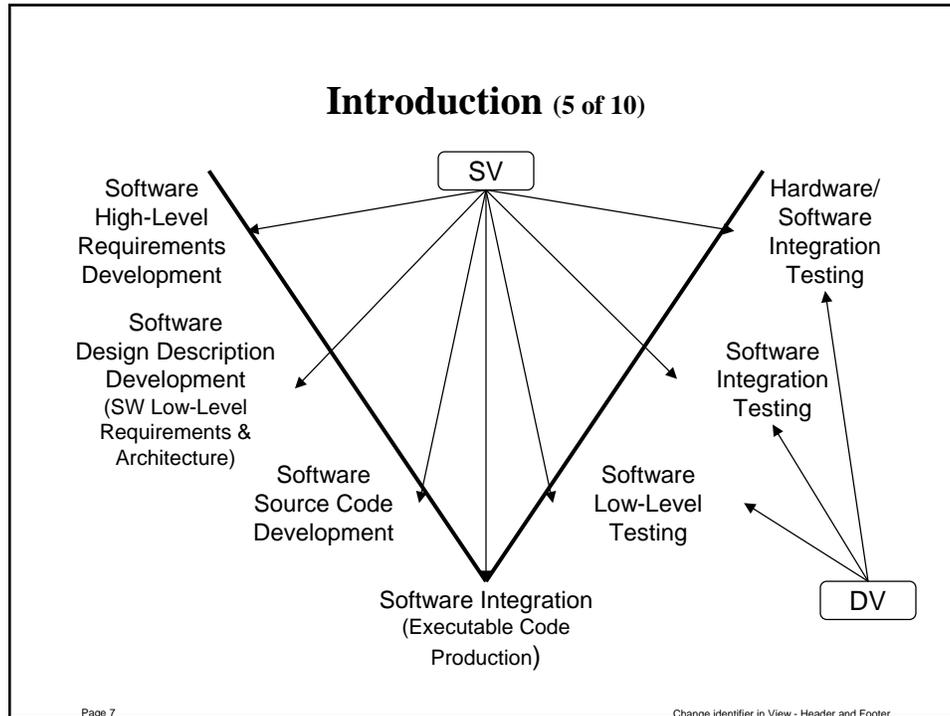
Software Requirements	Software Design	Software Coding	Integration
	Reviews		
	Analyses		
	Test Preparation		
	Test Execution		

Page 6

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification



Introduction (6 of 10)

Static Verification	Dynamic Verification
Concerned with analysis of a (restricted) (mathematical) model of the system/implementation	Concerned with observing behavior while exercising the partial or complete implementation
Can be applied to any lifecycle artifact, or a model of the artifact's properties	Can only be applied to the partial or complete implementation in the (simulated) target environment
Thoroughness accomplished with a feasibly large input or state space	Thoroughness accomplished with an infeasibly large input space
Discovers errors early in the lifecycle	Discovers errors late in the lifecycle
Discovers errors directly	Discovers symptoms of errors

Page 8 Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Introduction (7 of 10)

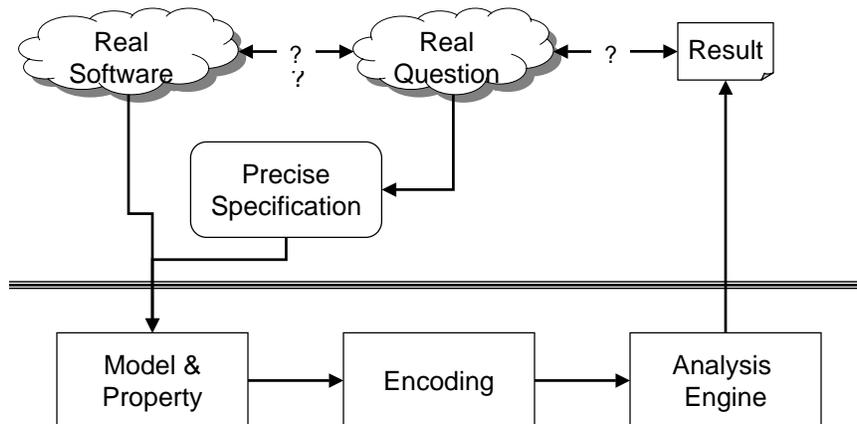
Static Verification	Dynamic Verification
Error detection (generally) not impacted by previously discovered errors	Error detection influenced by test data selection, previously discovered errors may be masking others
Error detection (generally) not impacted by other (undiscovered) errors	<ul style="list-style-type: none"> •Execution (Controllability) •Infection (Controllability) •Propagation (Observability)
Approximate (informal) SV methods can detect > 60% of all errors	Can detect 100% of all errors with the right test set
Formal SV methods may detect > 90% of all errors	
	Only method for detecting non-functional errors

Page 9

Change Identifier in View - Header and Footer

Introduction (8 of 10)

- Recipe for Static Verification (Analysis)



Page 10

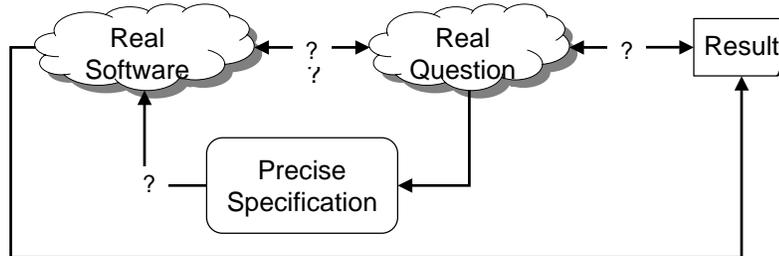
Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Introduction (9 of 10)

- Recipe for Dynamic Verification (Analysis)



Page 11

Change Identifier in View - Header and Footer

Introduction (10 of 10)

- Retrospective versus Constructive Static Verification
 - Historically, most SV has been *retrospective* - analysis after delivery of a "finished" product as part of "V & V" activity.
 - Major problems
 - Effectiveness of retrospective SV critically depends on how well the product is built in the first place!
 - Example: Chinook Mark 2 FADEC - defied static verification by all known methods and tools!
 - Often too late in life-cycle to gain full benefit.
 - There is strong evidence to support *constructive* SV - the application of SV as a *development* activity as the system is built.
 - Catch: For constructive SV to work, it must be *efficient* and *modular*.

Page 12

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Agenda

- Introduction
- **The Catch**
- Static Verification and DO-178B Objectives
- Types of Extended Static Verification
- Some Static Verification Languages and Tools
- Static Verification Projects
- Conclusions

Page 13

Change identifier in View - Header and Footer

The Catch (1 of 2)

- Languages *Do* Matter!
- *Ambiguity* in language design is the enemy of SV.
 - ISO C90 has about 200 undefined "features."
 - What's a tool to do when it encounters one of these?
 - Make an assumption? (Dangerous...)
 - Analyze every possible semantics? (Analysis time explodes...)
 - Specialize to the compiler? (Nightmare...)
 - Ignore it? (Dangerous...)
 - Certain language features defy analysis (technically, setting NP-hard or undecidable problems.)
 - For example, complete analysis of pointers and aliasing in C.
- What about language subsets?

Page 14

Change identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

The Catch (2 of 2)

- The irony of subsets and their analysis...
- To increase market share and attractiveness, most SV tools attempt analysis of the "whole language", and therefore suffer from the ambiguity problem.
 - Analysis might be
 - Unsound
 - Incomplete
 - Too slow for constructive use
- BUT...everyone uses subsets!!
 - You *do* have a language or coding standard, right?!
- Possible way out: use of well-defined *unambiguous* subsets.
 - More on this later...

Page 15

Change Identifier in View - Header and Footer

Agenda

- Introduction
- The Catch
- **Static Verification and DO-178B Objectives**
- Types of Extended Static Verification
- Some Static Verification Languages and Tools
- Static Verification Projects
- Conclusions

Page 16

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Static Verification and DO-178B Objectives

(1 of 2)

- DO-178B Table A-5 Objective 6 asks for the source code to be accurate (i.e. correct) and consistent
- Refers to section 6.3.4f that in turn calls out the following analyses
 - Stack usage (worst case memory usage)
 - Fixed point arithmetic overflow and resolution
 - Resource contention
 - Worst-case execution timing
 - Exception handling
 - Use of uninitialized variables or constants (aka "data-flow analysis")
 - Unused variables or constants
 - Data corruption due to task or interrupt conflicts

Page 17

Change Identifier in View - Header and Footer

Static Verification and DO-178B Objectives

(2 of 2)

- These analyses can be accomplished either manually or with tools.
 - How many are performing manual analyses?
 - How many are using tools?
 - Full automation?
 - Partial?
 - Qualified?
 - Effective?
- DO-178B does not preclude other analyses.
 - How many are performing other analyses?
 - What are they?

Page 18

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Agenda

- Introduction
- The Catch
- Static Verification and DO-178B Objectives
- **Types of Extended Static Verification**
- Some Static Verification Languages and Tools
- Static Verification Projects
- Conclusions

Page 19

Change Identifier in View - Header and Footer

Types of Extended Static Verification (1 of 9)

- **Static semantics and subset checking**
 - Enforcement of language subset rules and/or local coding standards.
 - Simple stuff: "Don't use language feature X".
 - More subtle:
 - "There shall be no function side-effects."
 - "There shall be no dependence on expression evaluation order."

Page 20

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Types of Extended Static Verification (2 of 9)

- **Data flow analysis**
 - Very old style of analysis (at least 30 years old now...)
 - Analysis that all variables have a well-defined value before they are referenced - a **very** common source of programming defect, which is **very** difficult to detect by testing.
 - Can be "local" (within a single function), or "global" (whole program.)
 - Lots of tool support for this for most languages. Should be mandatory!

Page 21

Change identifier in View - Header and Footer

Types of Extended Static Verification (3 of 9)

- **Information flow analysis**
 - Does all data-flow analysis, **plus**
 - Verification of required inputs-to-outputs information flow
 - i.e., dependencies of outputs on inputs.
 - Detection of invariant or "stable" expressions.
 - Detection of ineffective statements and expressions
 - E.g. writing to a variable twice without reading it in between.
 - (Aside - IFA mostly invented by the security community - it's very useful if you want to know where your (secret) data is going!)

Page 22

Change identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Types of Extended Static Verification (4 of 9)

- **Theorem Proving**
 - The generation of small theorems about a program, the proof of which verify particular program properties.
 - Start with an assertion at two program points (initial, final)
 - Show that the statements between the two statements transform the initial assertion into the final assertion, or why not
 - Automated theorem proving is now **very** good at doing the hard work for you!
 - Program properties we can verify:
 - Exception freedom (e.g. no buffer overflows!)
 - Partial correctness (w.r.t. "contracts")
 - Safety properties (e.g. invariants)
 - Examples: ESC/Java, SPARK, Microsoft SLAM.

Page 23

Change Identifier in View - Header and Footer

Types of Extended Static Verification (5 of 9)

- **Abstract Interpretation**
 - Represents selective dynamics of a software application through a static mathematical model.
 - Extracts only those properties from the source code relevant to the analysis (slicing).
 - Allows analysis and prediction of selected behavior.
 - Checks each code section against all possible inputs.
 - Still concerns about the size of that space (scaling).
 - Is a mature technology.
 - Developed about 20 years ago, but had to wait for increased computing power.
 - Commercial tool support now available.

Page 24

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Types of Extended Static Verification (6 of 9)

- **Symbolic Execution**
 - Represents the output values of a program as a symbolic (abstract) specification (function) of the inputs.
 - Use symbols instead of values to represent the inputs to the program.
 - Represent the values of program variables as symbolic expressions.
 - Can be used to analyze data states.
 - Can be used to generate test specifications or data.
 - Can be used to verify safety property constraints.
 - Branching constructs cause complexity.
 - Especially dynamic loops (and recursion)!
 - Length and number of input-to-output paths cause problems.
 - Mature technology – not widely used.
 - Commercial tools for ForTran 77

Page 25

Change identifier in View - Header and Footer

Types of Extended Static Verification (7 of 9)

- **Model Checking**
 - A mathematical model of a system as a state machine.
 - Mechanical exploration of that state machine to verify a particular property.
 - Tool either says "Yes" or "No, and here's a counter-example"

 - Main uses so far in hardware design and verification of communications protocols.
 - Some use in software now - Microsoft SLAM for instance.
 - Problem: Computation time/space tends to explode.
 - Every path through the state machine is explored.

 - A very active research field, so keep an eye on this one.

Page 26

Change identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Types of Extended Static Verification (8 of 9)

- **Timing and Schedulability analysis**
 - WCET Analysis - to find worst-case execution time of single tasks or threads.
 - Theory is well-developed, but complexity of modern CPUs has made tool support very hard, and therefore little use in industry so far..
 - Schedulability
 - Analysis of "whole program" (tasks, interrupt handlers, scheduler etc.) to determine end-to-end response times, deadline satisfaction etc.
 - "Rate monotonic" family of analyses are the best known.
 - Mature tool support exists now.
 - Catch: adoption of an analyzable (subset) concurrency model. e.g Ada95 Ravenscar tasking profile.

Page 27

Change Identifier in View - Header and Footer

Types of Extended Static Verification (9 of 9)

- **Memory use analysis**
 - Analysis to determine "no memory leaks" or maximum bound on memory usage.
 - Depends heavily on whether you use pointers/malloc/free/garbage collection etc. etc.
 - In simple languages, this reduces to an analysis of worst-case stack usage in a non-recursive program. Easy.
 - Worst-case - analysis of allocation, deallocation, garbage collection etc. in a dynamic language. Very hard!
 - Obvious interaction with real-time and timing-analysis issues.

Page 28

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Agenda

- Introduction
- The Catch
- Static Verification and DO-178B Objectives
- Types of Extended Static Verification
- **Some Static Verification Languages and Tools**
- Static Verification Projects
- Conclusions

Page 29

Change Identifier in View - Header and Footer

Some Static Verification Languages and Tools

(1 of 5)

- MISRA C
 - A set of "guidelines" for the use of C developed by the automotive industry. Varied acceptance.
 - 127 rules.
 - Rules are informally defined, in "ISO English."
 - Rules basically imply: subset checking, static semantic checks, and data-flow analysis.
- The good news:
 - Probably the best (public) guidelines for the use of C ever produced.
 - Adoption by automotive industry has prompted much activity from the tool vendors to support it.
 - Now being revised to give a more formal definition of the rules.
 - Has influenced significant projects, such as JSF.

Page 30

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Some Static Verification Languages and Tools

(2 of 5)

- MISRA C
 - The bad news:
 - Informality of rules and inherent ambiguity of C90
 - "Compliance" is almost impossible to claim.
 - All tool vendors claim "100%" implementation of the rules.
 - All the tools are different!
 - Which is right?!?
 - C is very "pointer-centric" - meaning some of the rules are NP-hard or even undecidable to implement - oh dear...
 - Deep analysis is *slow*, which limits constructive use.
 - Tools suffer from high *false-alarm rate*.

Page 31

Change identifier in View - Header and Footer

Some Static Verification Languages and Tools

(3 of 5)

- The Extended Static Checker for Java (ESC/Java)
 - Advanced research tool from Compaq/HP SRC.
 - Developed from ESC/Modula 3
 - Implements data-flow analysis, theorem-proving and uses *annotations* which embody "design-by-contract" information for the tool to use.
 - Theorem proving is "under the hood", so (almost) invisible to user.
 - Problems: will this ever be a commercial product? Java is still unproven in hard real-time, safety-critical systems.
 - Watch out for: SofCheck Inc - trying to bring similar technology to commercial use.

Page 32

Change identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Some Static Verification Languages and Tools

(4 of 5)

- SPARK
 - An annotated (design-by-contract again...) subset of Ada95.
 - Subset is specifically designed for hard real-time, embedded, safety- and security-critical systems.
 - Designed to have a totally unambiguous semantics, so analysis can be *both* deep and efficient.
 - Tools do *not* attempt analysis of "full Ada" so the whole-language problem does not appear.
- Analyses available:
 - Mandatory: subset checking, static semantics, data-flow analysis.
 - Optional (stage 1): Information-flow analysis.
 - Optional (stage 2): Theorem proving for exception freedom, partial correctness, safety properties.

Page 33

Change identifier in View - Header and Footer

Some Static Verification Languages and Tools

(5 of 5)

- SPARK
 - Good news:
 - Has an industrial track record in *all* of the toughest software standards in many industries:
 - Commercial Aero: DO-178B Level A
 - Defence: UK Def Stan 00-55 SIL4
 - Security: ITSEC E6, Common Criteria
 - Rail: CENELEC 50128
 - Not so good news:
 - "But it's Ada..."
 - It's British! ("Why can't we buy an American one?")

Page 34

Change identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Agenda

- Introduction
- The Catch
- Static Verification and DO-178B Objectives
- Types of Extended Static Verification
- Some Static Verification Languages and Tools
- **Static Verification Projects**
- Conclusions

Page 35

Change Identifier in View - Header and Footer

Static Verification Projects The Lockheed-Martin C130J



Page 36

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

C130J Mission Computer

- 130,000 lines of safety related code in mission computer
- Process designed to
 - reduce V&V costs (and consequent delays)
 - meet certification requirements, UK MoD, RAF, and FAA
- Based on rigorous specification and design
 - SPC CoRE (Parnas tables)
 - SPARK

Page 37

Change Identifier in View - Header and Footer

C130J Mission Computer - Timeline

- 1995 - Lockheed adoption of SPARK “encouraged” by RAF and QinetiQ Boscombe Down for Level A Mission Computer (MC) and Bus Interface Unit (BIU).
- 1996-1998 - Aircraft development and flight test. Dual certification to *both* DO-178B and Def Stan 00-55.
- 1999 - Retrospective static analysis of *all* software conducted by AeroSystems International (AEI) in the UK.

Page 38

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

C130J Mission Computer - Observations

- During adoption of SV
 - Significant drop in pre-test defect rate.
 - Subsequent saving in formal test process.
 - Some significant defects found in code that had already *passed* formal testing.
 - SPARK *forced* engineers to ask tough questions (e.g. "What inputs is this output validity flag supposed to depend on?"). Actually found specification and requirements defects.

Page 39

Change Identifier in View - Header and Footer

C130J - Lockheed on SPARK

- Some errors immediately uncovered by formal analysis, such as conditional initialization errors may only emerge after very extensive testing.
- The technology for generating and discharging the proof obligations, based on the SPARK components of Ada, was crucial, in binding the code to the initial requirements.
- SPARK provides an extremely robust and efficient basis for formal verification.
- The process has proven effective with typical software developers and did not necessitate an inordinate amount of additional training.
- Experience has shown that SPARK coding occurs at near typical Ada rates.
- Code written in SPARK is deterministic and inherently statically analysable.
- Very few errors have been found in the software during even the most rigorous levels of FAA testing, which is being successfully conducted for less than a fifth of the normal cost in industry.
- Correctness by construction is no longer a theoretical abstraction; it is now a practical way to develop software that exceeds its technical goals while delivering sterling business performance.

Page 40

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

C130J - Lockheed on SPARK

- Some errors immediately uncovered by formal analysis, such as conditional initialization errors may only emerge after very extensive testing.
- The technology for generating and discharging the proof obligations, based on the SPARK components of Ada, was crucial, in binding the code to the initial requirements.
- SPARK provides an extremely robust and efficient basis for formal verification.
- *Very few errors have been found in the software during even the most rigorous levels of FAA testing, which is being successfully conducted for **less than a fifth** of the normal cost in industry.*
- practical way to develop software that exceeds its technical goals while delivering sterling business performance.

Page 41

Change identifier in View - Header and Footer

C130J - The AeroSystems Study

- Static analysis of all the software on the aircraft, *after* the certification of the aircraft.
- On the MC and BIU, L-M had only performed static semantics and information-flow analysis - no proof.
- AEI did proof on the MC and BIU SPARK code - exception freedom and partial correctness with respect to Parnas tables.
- All anomalies recorded and classified.
- C. 10000 anomalies found. Approx 1% had safety impact.

Page 42

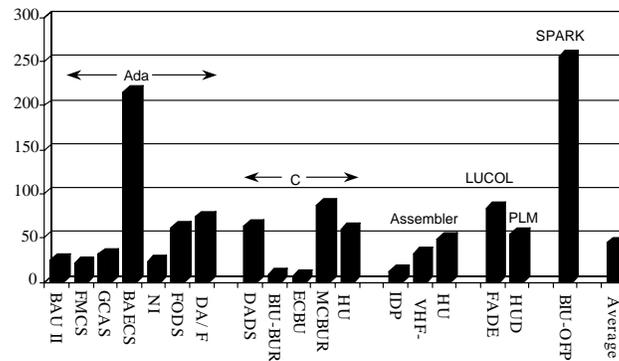
Change identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

C130J - AeroSystems Results

- Lines of code per anomaly by subsystem and programming language:



Page 43

Change Identifier in View - Header and Footer

Agenda

- Introduction
- The Catch
- Static Verification and DO-178B Objectives
- Types of Extended Static Verification
- Some Static Verification Languages and Tools
- Static Verification Projects
- **Conclusions**

Page 44

Change Identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

Conclusions (1 of 2)

- There is strong technical and commercial evidence to support the use of SV, *regardless* of safety/integrity level.
- SV directly addresses DO-178B objectives, and (perhaps more importantly) can indirectly ease integration, testing and subsequent lifecycle phases.
- "Extended static analysis" such as abstract interpretation, model checking, and theorem proving are now used on an industrial scale.
 - These may not be *required* by DO-178B, but that's no reason not to use such technology if they make your project better and/or cheaper!
- There are strong signs of a "new golden age" for SV:
 - New tools (e.g. Polyspace, RavenSPARK, SofCheck...)
 - New markets (e.g. automotive, security...)
 - New languages (e.g. Java, Microsoft Vault)

Page 45

Change identifier in View - Header and Footer

Conclusions (2 of 2)

- Static Verification is generally applied to a model of some property of either the intended or actual implementation.
 - But we can't model everything.
- Dynamic Verification is generally applied to the implementation in either a simulated or actual environment.
 - But we can't test for everything.
- Therefore, we need both.
 - Best if they are used in a complementary fashion.
 - Use strengths of one to cover the weaknesses of the other.
- We need to design and implement for verifiability!
 - Design for testability (DV) well established in hardware.
 - Design and implementation for SV is needed.
 - Languages really *do* matter!

Page 46

Change identifier in View - Header and Footer

2003 FAA National Software Conference

Tutorial on Static Verification

SV Resources

- Some background information, papers and so on for the languages and technologies mentioned in this tutorial:
- General
 - "Software Static Code Analysis: Lessons Learnt" by Andy German. CrossTalk Journal, November 2003 (to appear).
- MISRA C - www.misra.org.uk
- ESC/Java - research.compaq.com/SRC/esc/Esc.html

Page 47

Change Identifier in View - Header and Footer

SV Resources

- SPARK
 - www.sparkada.com
 - "High Integrity Software: The SPARK Approach to Safety and Security" by John Barnes. Addison Wesley, 2003. ISBN 0-321-13616-0.
- Microsoft SLAM - research.microsoft.com/projects/slam/main.htm
- SofCheck - www.sofcheck.com
- Abstract Interpretation: Polyspace - www.polyspace.com
- The C130J
 - "Correctness by Construction: Better can also be Cheaper" by Peter Amey. CrossTalk Journal, March 2002. www.stsc.hill.af.mil

Page 48

Change Identifier in View - Header and Footer