

The Road from Software Testing to Theorem Proving

A Short Compendium of my Favorite Software Verification Techniques

Frédéric Painchaud
DRDC Valcartier / Robustness and Software Analysis Group

December 2010



The Road...



Copyright © 2009 Dimension Films, 2929 Productions, Nick Wechsler Productions, Chockstone Pictures

The Road...




The Road...

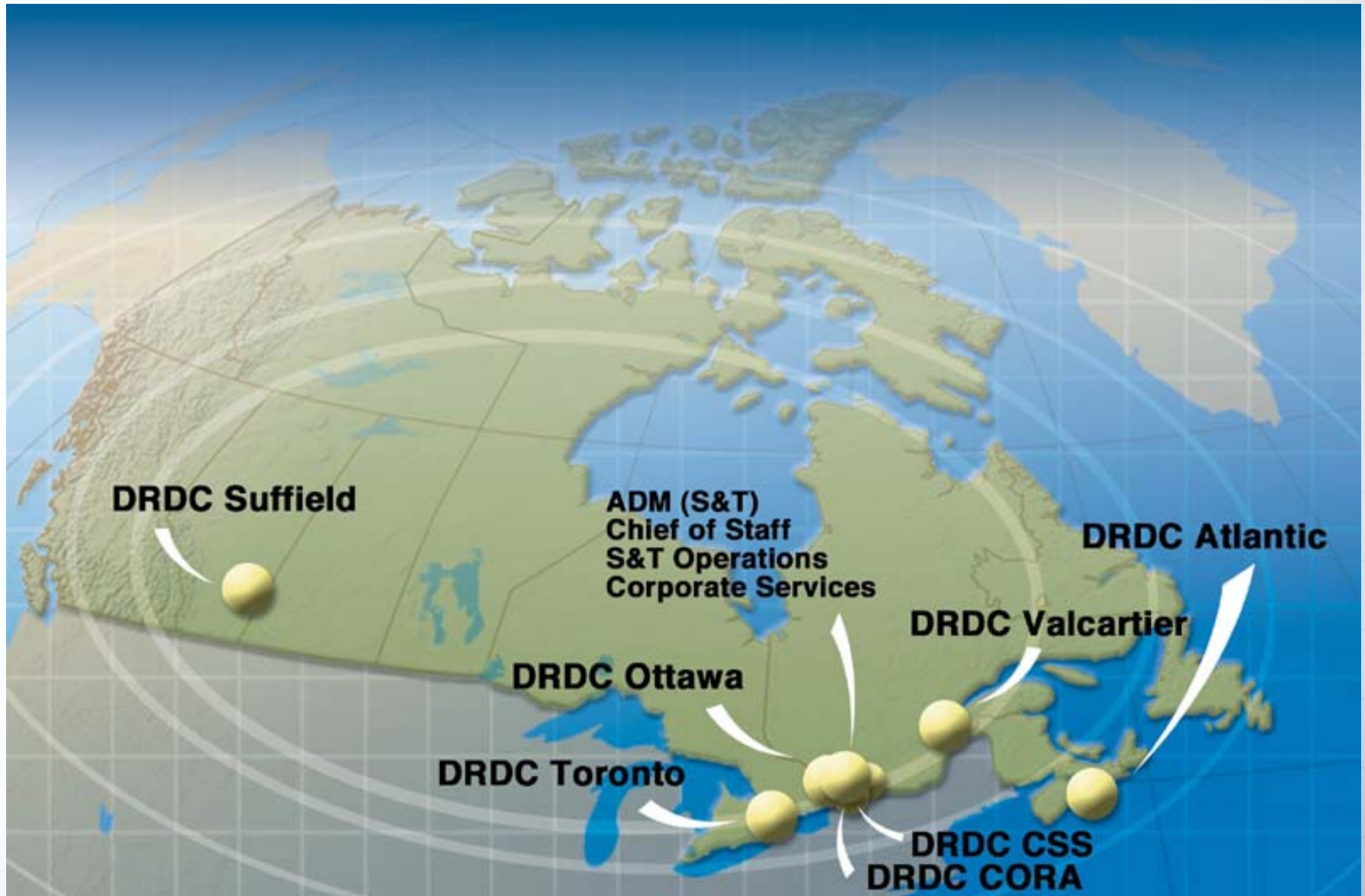


“The first principle is that you must not fool yourself - and you are the easiest person to fool.”
- Richard P. Feynman

Agenda

- Context
 - A Selection of my Favorite Software Verification Techniques
 - Implementation Level
 - Testing
 - Security Vulnerability Scanner Tools
 - Specification/Design Level
 - Architectural Risk Analysis
 - Model Checking
 - Lightweight Formal Methods
 - Theorem Proving
 - Discussion
- 
- Decreasing level of familiarity among software developers

Context – Where is DRDC?



Context – DRDC's Mandate

- Defence R&D Canada ensures that the Canadian Forces are technologically prepared and operationally relevant by:
 - Providing expert S&T advice to the Canadian Forces and Department of National Defence;
 - Conducting research, development and analysis to contribute to new and improved defence capabilities;
 - Anticipating and advising on future S&T trends, threats and opportunities;
 - Engaging industrial, academic and international partners in the generation and commercialization of technology;
 - Providing S&T for external customers to enhance Defence S&T capacity.

Context – So What?

- Our main clients: Canadian Forces, Public Safety Canada, OGDs
 - They use very little formal methods, no theorem proving
 - Approximately ten years behind, and it is normal
- Applied research
 - Little fundamental, academic research
- My group: two projects a year involving software development
 - Few opportunities to experiment with various verification techniques
- Few people in DRDC know about formal methods
 - Not much traction inside the organization
- The road I am presenting is a *summarized* compendium of the views I am promoting to my clients when I have an opportunity
 - Not something I do everyday
 - Comes from a lot of “formal methods watch”
 - My usual focus is *security*

A Selection of my Favorite Software Verification Techniques – Implementation Level

- Testing
 - Key point: making sure that the test suite is “large enough” and contains “effective test cases”
 - “Large enough”: Code coverage
 - **Statement coverage:** implies function/method coverage
 - Tricky for object-oriented code and exception handlers, requires fault injection
 - **Loop coverage:** have all loops been executed at least twice?
 - Helps to focus more specifically on loop conditions correctness
 - **Condition/decision coverage:** when the correctness of the implemented tests in the application is *important*, e.g., control applications

A Selection of my Favorite Software Verification Techniques – Implementation Level

- Testing (cont'd)
 - “Large enough”: Code coverage (cont'd)
 - **Modified condition/decision coverage**: when the correctness of the implemented tests in the application is *paramount*, e.g., safety-critical applications (DO-178B Level A)
 - “Effective test cases”:
 - **Requirements-based testing**: defining test cases from requirements *helps* to ensure that requirements are implemented correctly and completely by:
 - having 100% success and
 - 100% statement coverage after executing test suite
 - **Mutation testing**: ensures that modifying the tested application’s code, even very slightly, also modifies the end result of at least one test case in the test suite

A Selection of my Favorite Software Verification Techniques – Implementation Level

- Testing (cont'd)
 - Pros
 - Very well-known and studied
 - Intuitive for programmers
 - Many supporting tools available for code coverage, test cases generation, mutation testing, etc, even freely
 - Cons
 - Often not performed adequately (code coverage is not measured, no mutation testing, only unit testing, etc)
 - False sense of correctness, fails during integration
 - **Important:** testing often fails to show the *presence* of bugs
 - Culture of “Never design, write once, debug many”

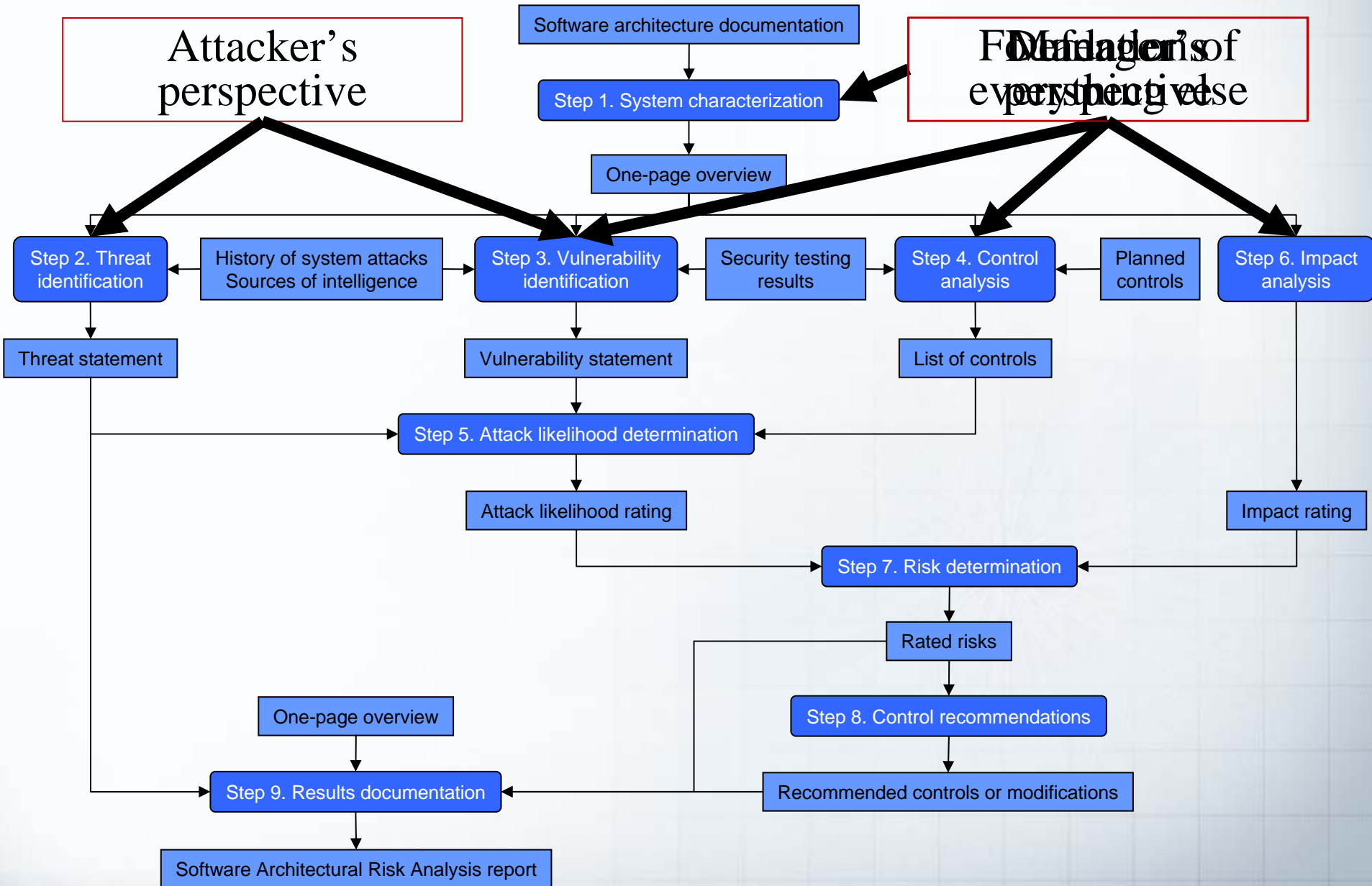
A Selection of my Favorite Software Verification Techniques – Implementation Level

- Security Vulnerability Scanner Tools
 - Key points:
 - Increasing adoption, more than for other static analysis tools
 - “Scan often, use many”
 - Pros
 - Intuitive for programmers, fit very well in current “build environments”
 - Minimal impact on management and development processes
 - Overall quality of these tools increased significantly since 2005
 - Cons
 - Tendency to select only one tool judged as the best one, mostly from the tool’s marketing
 - Secure development and C&A processes slowly become centered around tools

A Selection of my Favorite Software Verification Techniques – Specification/Design Level

- Architectural Risk Analysis
 - Key points:
 - Structured and practical process to consider security at the architectural level, either during new design or for legacy applications
 - Cigital Inc. is the main pioneer
 - My process contains Cigital's activities integrated within the usual risk analysis process
 - Cigital does not reveal its entire process

Architectural Risk Analysis Process



Common Attack Pattern Enumeration and Classification (CAPEC, see capec.mitre.org)

CAPEC Common Attack Pattern Enumeration and Classification A Community Knowledge Resource for Building Secure Software

Home > CAPEC List > VIEW GRAPH: CAPEC-1000: Mechanism of Attack (Release 1.5)

Search by ID:

- CAPEC List**
- Full CAPEC Dictionary
- Methods of Attack View Reports
- About CAPEC**
- Documents
- Resources
- Community**
- Related Activities
- Collaboration List
- News & Events**
- Calendar
- Free Newsletter
- Contact Us**
- Search the Site

CAPEC-1000: Mechanism of Attack

Definition Graph List Slice XML.zip

HasMember		225	Exploitation of Authentication		1000	
HasMember		232	Exploitation of Privilege/Trust		1000	
HasMember		255	Data Structure Attacks		1000	
HasMember		262	Resource Manipulation		1000	
HasMember		286	Network Reconnaissance		1000	

	CAPECs in this view		Total CAPECs
Total	322	out of	384
Views	0	out of	6
Categories	18	out of	67
Attack Patterns	311	out of	311

more

[Expand All](#) | [Collapse All](#)

1000 - Mechanism of Attack

- [Data Leakage Attacks - \(118\)](#)
- [Resource Depletion - \(119\)](#)
- [Injection \(Injecting Control Plane content through the Data Plane\) - \(152\)](#)
- [Spoofing - \(156\)](#)
- [Time and State Attacks - \(172\)](#)
- [Abuse of Functionality - \(210\)](#)
- [Probabilistic Techniques - \(223\)](#)
- [Exploitation of Authentication - \(225\)](#)
- [Exploitation of Privilege/Trust - \(232\)](#)
- [Data Structure Attacks - \(255\)](#)
- [Resource Manipulation - \(262\)](#)
- [Network Reconnaissance - \(286\)](#)

[BACK TO TOP](#)

Page Last Updated: April 02, 2010



CAPEC is a [Software Assurance](#) strategic initiative co-sponsored by the [National Cyber Security Division](#) of the [U.S. Department of Homeland Security](#).
This Web site is sponsored and managed by [The MITRE Corporation](#) to enable stakeholder collaboration.
Copyright 2010, The MITRE Corporation. CAPEC and the CAPEC logo are trademarks of The MITRE Corporation.
Contact capec@mitre.org for more information.

[Privacy policy](#)
[Terms of use](#)
[Contact us](#)



Common Weakness Enumeration (CWE, see cwe.mitre.org)

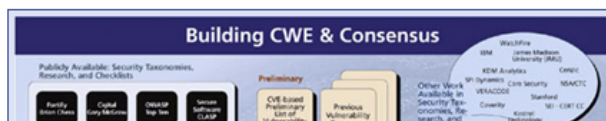


CWE and SANS Institute
TOP 25 MOST DANGEROUS SOFTWARE ERRORS

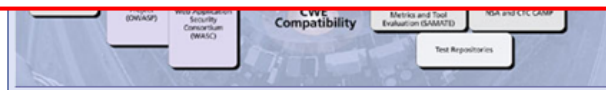
Search by ID:

- CWE List**
 - Full Dictionary View
 - Development View
 - Research View
 - Reports
- About**
 - Sources
 - Process
 - Documents
- Community**
 - Related Activities
 - Discussion List
 - Research
 - CWE/SANS Top 25
 - CWSS
- News**
 - Calendar
 - Free Newsletter
- Compatibility**
 - Program
 - Requirements
 - Declarations
 - Make a Declaration
- Contact Us**
 - Search the Site

International in scope and free for public use, CWE™ provides a unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design.



Security Vulnerability Scanner Tools



Similar Standards

<ul style="list-style-type: none">Attack Patterns (CAPEC)Vulnerabilities (CVE)Configurations (CCE)Platforms (CPE)Malware (MAEC)	<ul style="list-style-type: none">Assessment Language (OVAL)Checklist Language (XCCDF)Log Format (CEE)Security Content Automation (SCAP)Making Security Measurable
---	--

- ### News
- [New ISO/IEC Report Lists the 51 Most Common Vulnerabilities in Programming Languages](#)
 - [CWE and Making Security Measurable panel at Rethinking Cyber Security: A Systems-Based Approach Conference](#)
 - [CWE/CAPEC/MAEC panel at 11th Annual Security Conference](#)
 - [CWE/Making Security Measurable and CAPEC briefings at AppSec DC 2010](#)
 - [CWE/CAPEC Keynote presentation at SecureSDLC Conference](#)

- ### Upcoming Events
- [Software Assurance panel at CTF Congress, November 30-December 2](#)
 - [CWE/Making Security Measurable briefing at ITU-T Security Workshop, December 6-7](#)
 - [CWE/CAPEC/MAEC briefings at DHS/DoD SwA Working Group Meeting Session, December 14-16](#)

Status Report

Version 1.10 posted September 27, 2010. There are 7 new entries, mostly for synchronization and "memory safety" issues; changes to 115 entries; modified mitigations for 34 entries; and updates to 43 relationships. There were no schema changes. The CWE/SANS Top 25 was updated to reflect the new version.

More Information
cwe.mitre.org

Page Last Updated: November 23, 2010

CWE is a [Software Assurance](#) strategic initiative co-sponsored by the [National Cyber Security Division](#) of the [U.S. Department of Homeland Security](#). This Web site is sponsored and managed by [The MITRE Corporation](#) to enable stakeholder collaboration. Copyright 2010, The MITRE Corporation. CWE and the CWE logo are trademarks of The MITRE Corporation. Contact cwe@mitre.org for more information.



[Privacy policy](#)
[Terms of use](#)
[Contact us](#)



A Selection of my Favorite Software Verification Techniques – Specification/Design Level

- Architectural Risk Analysis (cont'd)
 - Pros
 - Practical: even though it is framed inside a process, developers are really finding security flaws
 - Helps to find higher-level security issues, not only buffer overflows, format strings, etc
 - Compatible with usual Software Development Life Cycles
 - Cons
 - In immature organizations, it requires efforts to modify current, usually *ad hoc*, development processes
 - Requires trained personnel in security to take the attacker's perspective

A Selection of my Favorite Software Verification Techniques – Specification/Design Level

- Model Checking
 - Key point: given a formal model of a system, it verifies automatically whether this model meets a given formal specification
 - Pros
 - Generic: theories have been developed for many types of models (timed, stochastic, etc) and specification logics (temporal, modal, etc)
 - Most tools now handle the state explosion problem to an acceptable extent
 - Cons
 - In software model checking, most useful logics in practice (more expressive than propositional and monadic predicate logics) are hardly tractable, semidecidable or undecidable so it can fail to prove or disprove a given property
 - It can be very difficult to produce a correct but yet abstract model of a system
 - Requires trained and dedicated personnel to analyze the results

A Selection of my Favorite Software Verification Techniques – Specification/Design Level

- Lightweight Formal Methods
 - Key points:
 - Coined by Prof. Daniel Jackson at the MIT (see alloy.mit.edu)
 - Benefit from the precision of mathematical thinking and linguistic advantages of formal methods while sacrificing enough language expressiveness to get simplicity and a fully-automated mechanical analysis that is effective at finding errors
 - In some sense, they could be named “Pragmatic Formal Methods” (but sounds pedant) or “Formal Testing” (but sounds pejorative and is too restrictive)

A Selection of my Favorite Software Verification Techniques – Specification/Design Level

- Lightweight Formal Methods (cont'd)
 - Pros
 - Best of many worlds: smaller languages, easier to learn, concepts closer to programmers' mindsets, good tools support, application- and domain-specific
 - Con (comment)
 - I believe many formal methods fall into this “more friendly” category but are unfortunately not presented with this perspective: modern compilers (!), automatic static analysis tools, model checking, SAT solvers, first-order automatic theorem proving, etc

A Selection of my Favorite Software Verification Techniques – Specification/Design Level

- Theorem Proving
 - Key point: prove that some generic formal properties, such as consistency or completeness, are valid in a formally-defined specification
 - Pros
 - Proves the absence of errors (with respect to the verified properties), no partial error detection anymore
 - Backed by fifty years of research and development, world-wide
 - First-order theorem proving now enjoys very efficient, effective and some fully-automatic tools
 - Cons
 - Formal specification and theorem proving are very complex on mainstream software developers' scale
 - With interactive theorem proving (more expressive logics), the cost of proof (analysis) is usually an order of magnitude greater than the cost of formal specification, which is already high

Discussion – Comments Relating these Techniques to DO-178B/C Certification

- Testing
 - Until we can formally specify the entire aircraft platform systems (hardware and software), we will have to perform some software testing (at the very least integration testing) to verify that the software works correctly in its environment.
- Security Vulnerability Scanner Tools
 - Note: In the context of DO-178B/C, these tools are not security-enforcing tools but more generically correctness-enforcing tools (static and dynamic).
 - Tools must be qualified but are still incomplete (especially taken individually) and potentially incorrect (except if qualification is infallible). Cross-verifying one tool's results with one or more other tools should be discussed (if not already in DO-178C).

Discussion – Comments Relating these Techniques to DO-178B/C Certification

- Architectural Risk Analysis
 - Note: If you switch its focus from security to safety and rename it Threat and Risk Assessment (TRA), you get a process already performed in Safety Engineering which can be used to progress more smoothly towards formal methods adoption.
 - For organizations with a small or inexistent “culture of correctness”, forcing the adoption of formal methods is unrealistic.
- Model Checking/Lightweight Formal Methods/Theorem Proving
 - The qualification of these tools is at least discussed in DO-178B/C; the process is still imperfect but was refined. But the processes and culture behind formal methods must still be mapped onto and integrated into DO-178B/C required objectives by the organizations that want to get the certification.

Discussion – From the List of *Just-in-Time* Topics Prepared Ahead of the Workshop

- “Conservative Representation” (ref DO-178C): A conservative representation is *sound* with respect to the original representation but *incomplete*. Thus, every property true of the conservative representation can be mapped to a true property of the original representation (sound). But not every property true of the original representation is true of the conservative representation (incomplete).
- “Necessary Low-Level Software Requirements” (ref DO-178C): It is a requirements traceability problem. First, you need to formalize all high-level and low-level software requirements. Second, you need to find minimal sets of low-level software requirements so that each set entails one or more high-level software requirements. Finally, you need to show that every low-level software requirement is a member of at least one set. In other words, low-level requirements must constitute a correct and minimal step-wise refinement of high-level requirements.

Additionally, you could show that every high-level software requirement is entailed.

Discussion – From the List of *Just-in-Time* Topics Prepared Ahead of the Workshop

- “Avoidance of Unintended Functionality” (ref DO-178C): It is another requirements traceability problem. The implementation must be formal, i.e., you must have a complete formal semantics for its programming language. Using the same principle than in the “Necessary Low-Level Software Requirements” topic, you must show that the code is a correct and minimal step-wise refinement of low-level requirements.
- “Soundness of a Theorem-Proving Approach” (ref DO-178C): My interpretation is that the intent behind “soundness” here is in line with the “Conservative Representation” topic. A theorem-proving approach is sound if it works on conservative representations, or we might say sound representations (but not necessarily complete), with respect to the reality being modeled.

Discussion – From the List of *Just-in-Time* Topics Prepared Ahead of the Workshop

- “Assurance Cases”: Interactive theorem proving can be used to verify the validity of assurance cases already developed (say, manually). They can also support the development of new assurance cases by validating each step and performing some of the steps automatically.

DEFENCE



DÉFENSE