Tool-Based Testing: Static Analysis Dynamic Analysis Fuzzing

John Slankas
jbslanka@ncsu.edu

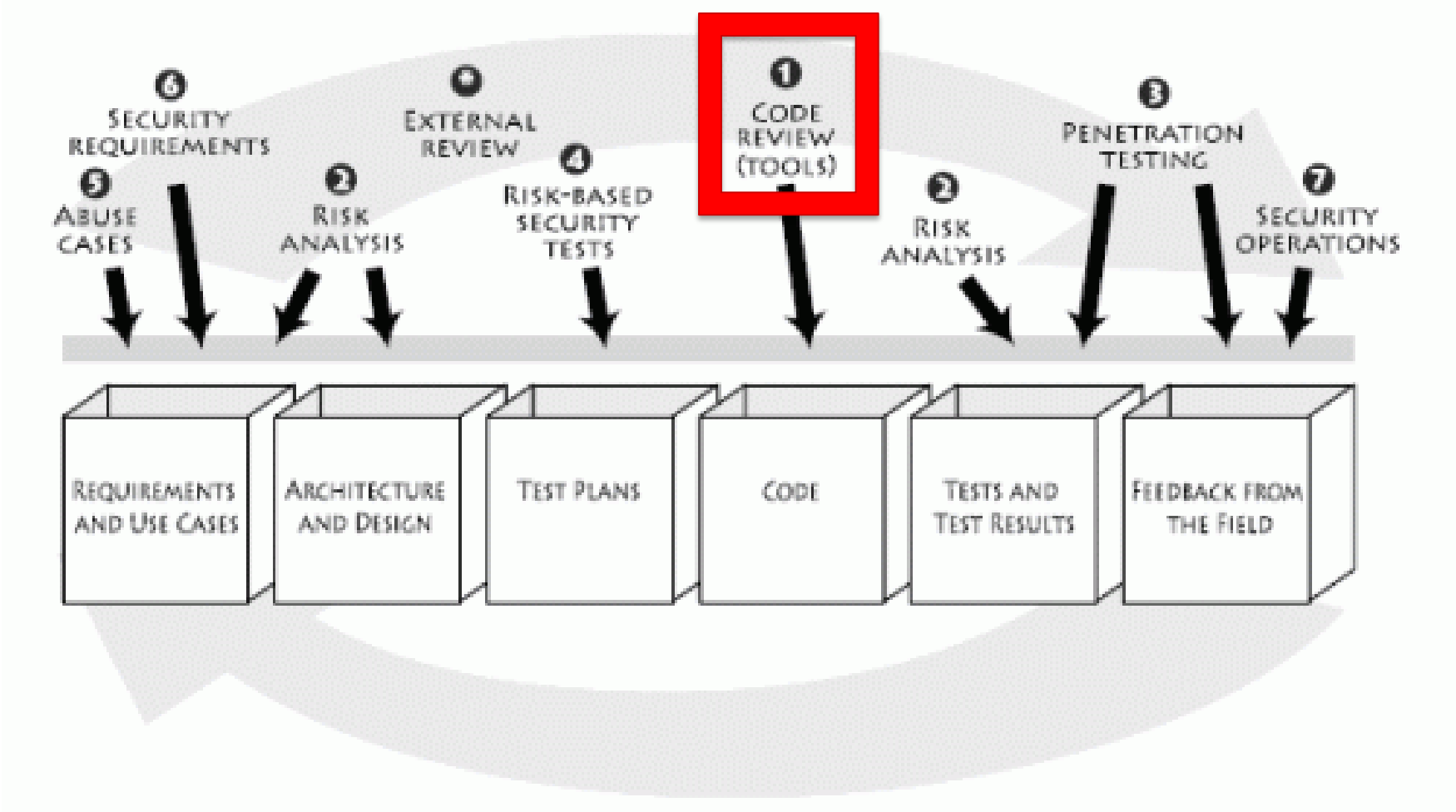
Computer Science
NC STATE UNIVERSITY

Slides adopted from Laurie Williams

Type Overview ….

- Static Code Analyzers
  - Adv: Find common patterns, low-effort to execute
  - Dis: False-positives, time-consuming to evaluate, incomplete
- Dynamic Code Analyzers
  - Adv: Low effort to execute, less false-positives
  - Dis: Incomplete
- Fuzzers
  - Adv: Low-effort to execute
  - Dis: Incomplete

Software Security Touchpoints



http://www.cigital.com/justiceleague/wp-content/uploads/2007/07/touchpoints.gif

Computer Science
NC STATE UNIVERSITY

Quote …

- Debugging is at least twice as hard as programming. If your code is as clever as you can possibly make it, then by definition you're not smart enough to debug it.

  – Brian Kernighan, *The Elements of Programming Style*, 1978

Static Analysis Tools

- Search through code to detect bug patterns (error prone coding practices that arise from the use of erroneous design patterns, misunderstanding of language semantics, or simple and common mistakes).
- Increasingly being used to identify security vulnerabilities
- "can peer into more of a program's dark corners with less fuss than dynamic analysis"

[Hovemeyer/Pugh, Chess/McGraw]

Why not manual code review?

- Time/resource consuming
- Auditors needs to know about forms of security vulnerabilities
- Tools:
  - Faster, so can be done more frequently
  - Don't require tool operator to have high level of security expertise
  - Can be done early

What will static analysis tools find?

- Implementation bugs: code-level vulnerabilities
  - Yes
- Design flaws
  - Not so much …
  - Though maybe static analysis alerts can be used to predict design flaws*
  - Hypothesis: *Programmers who don't know enough about security so they inject implementation bugs will also not know enough about design flaws related to security*

* Gegick, M.; Williams, L., "Toward the Use of Automated Static Analysis Alerts for Early Identification of Vulnerability- and Attack-prone Components," in *Internet Monitoring and Protection, 2007. ICIMP 2007. Second International Conference on* , vol., no., pp.18-18, 1-5 July 2007 doi: 10.1109/ICIMP.2007.46

Different Tools, Different Rulesets

- Different tools encode different bug patterns
- May consider using multiple tools

Computer Science
NC STATE UNIVERSITY

https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

Problems with static analysis tools

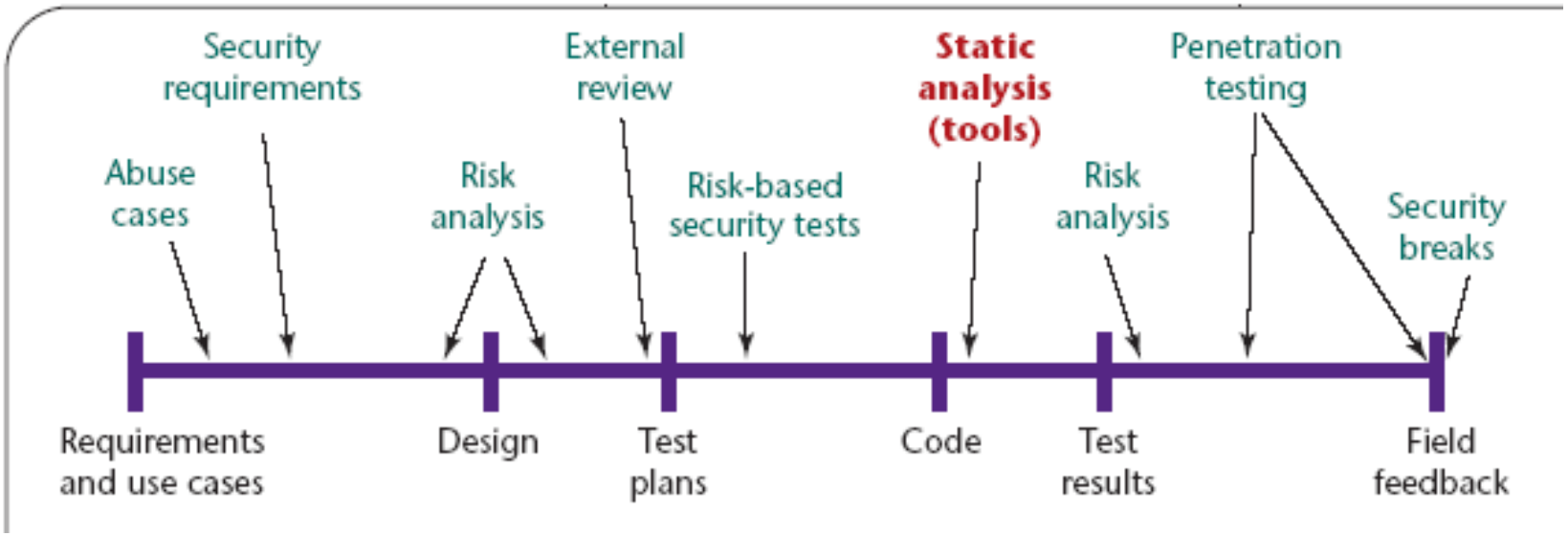
- False positive: the tool reports bugs the program doesn't contain
  - A static analysis tool will brag about having only 50% false positives.
  - Need to manually review and decide whether to fix or ignore. Some tools allow you to create filters of the types of bugs you don't want to see.

- False negative: the software contains bugs the tool doesn't report
  - May increase as static analysis tool develop works to reduce false positives

- May also detect "harmless bugs" which need human judgment to sort out

When?

- Early … once code is written



http://www.cigital.com/papers/download/bsi5-static.pdf

Computer Science
NC STATE UNIVERSITY

Key Characteristics of Tool

- Be designed for security
  - When you are doing security review
  - Tools designed with security in mind will embody more critical security knowledge
- Support multiple tiers
  - Many software projects are written in more than one language or on a single platform
- Be extensible
  - New rules can be added
- Useful and easy for both developers and security analysts
- Support existing development process/IDE

Static Analyis Exercise

- Run Fortify Analysis against BodgeIT

Kinds of Fuzzing

- Black Box
  - Easy to use
  - Explore only shallow states
- Grammar Based
  - Input informed by a grammar
  - More work to create the necessary grammar, but can explore state space much more exhaustively
- White Box
  - New inputs informed by underlying source code
  - Can be easy to use, but computational expensive

Fuzzing Inputs

- Mutation
  - Take legal input and mutate it
- Generational
  - Generate the input from scratch (e.g., from a grammar)
- Combinations
  - Generate initial input, mutate, generate new inputs
  - Generate mutations according to a grammar

Fuzzing: Dealing With Crashes

- A crash occurs ….
  - What was the root cause?
  - Can the input be smaller -> more understandable
  - Is it reproducible?
  - Does the crash signal an exploitable vulnerability?

Exercise 2: Fuzzing