Static Code Analysis

* What?
  + Simply put, analysis of computer code without executing it
  + Works by looking at the raw source code (or bytecode in the case of Java) and interpolates flaws using a variety of techniques
  + Often uses data flow analysis, taint analysis, and lexical analysis
    - Data flow analysis
      * Gets a sense of what values variables can have at different points in the code by maintaining a Control Flow Graph
      * Gives you a framework to prove facts about a program
      * Often used by compilers for optimization
    - Taint analysis/taint checking – finding variables and sections of code that can be modified by an outside users – identifies security risks
    - Lexical analysis – tokenizes source code syntax to abstract it make it easier to check and manipulate – combines well with the methods I just mentioned
* When?
  + Can use static code analysis anywhere, but used most often in safety-critical software
    - Nuclear, Aviation, Medical industries
* Why?
  + Performance scales well – can be run often and on lots of code – no execution
  + Finds certain issues like SQL injection flaws and buffer overflows with very high confidence
  + Can check for bugs when run-time processes are slow or unavailable – server calls and large data manipulations
  + Can maintain code quality for simple programming errors, sloppy coding practices, and anti-patterns
  + Better code coverage than dynamic analysis – catches bugs in code that is never executed but might be executed in the future with more implementation
  + Disadvantages
    - Biggest – produces high numbers of false positives
      * FindBugs boasts a less than 50% false positive rate
    - Extremely difficult to find certain vulnerabilities and flaws that can’t be verified until run-time or are just too complicated – authentication problems, access control violations, misuse of cryptographic functions
    - In general it’s difficult to absolutely prove that a security issue is actually an issue without extensive run-time testing
* To summarize – my interpretation based on research
  + Static code analysis is best used to help find security flaws in large, sensitive systems with high security or safety-critical requirements
  + Not meant to replace traditional testing techniques such as unit and integration testing, but meant to be used as an additional testing layer
  + Because of the high false positive rate it offers a “shotgun approach” to bug finding – you’re meant to use your own judgement to determine whether the found bugs are legitimate or not instead of taking them as fact

FindBugs

* Overview
  + Open source static code analysis plugin designed by the University of Maryland
  + First release was in June 2006
  + Most popular static code analysis tool for Java – downloaded over a million times
  + Many ways to view FindBugs data
* Eclipse IDE Integration
  + Easy to install – just go to the Eclipse store and download it for free
  + Shows line-by-line breakdown
  + SHOW WINDOW -> PREFERENCES
  + FINDBUGS -> FIND BUGS
* Traditional views
  + Command line interface and a GUI
  + GUI is very nice – mvn findbugs:gui
  + PROJECT 1 – iBox
  + Static Code Analysis often doesn’t produce substantial results for small-sized projects
* HTML view
  + FindBugs can also produce a boiled-down HTML view
  + SHOW REPORTS for older versions of JDK and Eclipse
  + No details provided about the offending code
  + Provides a holistic view – might show problem areas and packages
  + When you have a ton of code Static Code Analysis is at its best
    - It could reveal important metrics to improve coding best practices and reveal gaps in employee training
    - Shows you where the problem code is and where to focus debugging
* Questions?