DA-IICT

IT314: Software Engineering

Static Techniques
-with Source Code

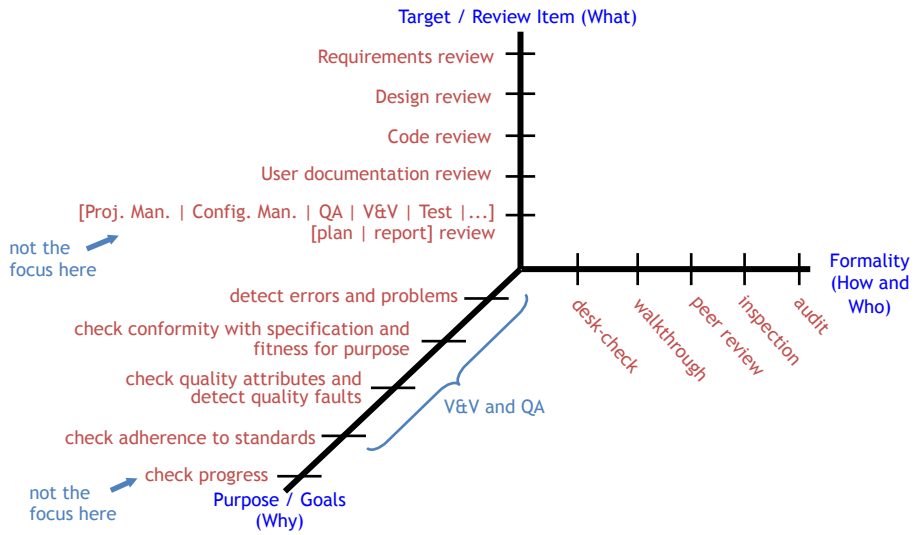
Saurabh Tiwari

1

Static Techniques

- Powerful way to improve quality and productivity of software development
- Finds defects early in the development process
- Complementary with dynamic approach
  - Finds defects rather than failures
- Performed manually or using static analysis tools
- Includes
  - Peer review
  - Walkthrough
  - Code Inspection
  - Static analysis

Types of reviews



Comparison of Defect-Detection Approaches

Table 20-2 Defect-Detection Rates

Removal Step	Lowest Rate	Modal Rate	Highest Rate
Informal design reviews	25%	35%	40%
Formal design inspections	45%	55%	65%
Informal code reviews	20%	25%	35%
Formal code inspections	45%	60%	70%
Modeling or prototyping	35%	65%	80%
Personal desk-checking of code	20%	40%	60%
Unit test	15%	30%	50%
New function (component) test	20%	30%	35%
Integration test	25%	35%	40%
Regression test	15%	25%	30%
System test	25%	40%	55%
Low-volume beta test (<10 sites)	25%	35%	40%
High-volume beta test (>1,000 sites)	60%	75%	85%

Source: Adapted from *Programming Productivity* (Jones 1986a), "Software Defect-Removal Efficiency" (Jones 1996), and "What We Have Learned About Fighting Defects" (Shull et al. 2002).

Code Analysis

Static Analysis

- Meant to review the source code
- Checking compliance of specific rules
- Usage of arguments and so forth

Dynamic Analysis

- It is executing the code, running the program and dynamically checking for inconsistencies of the given results.

Manual

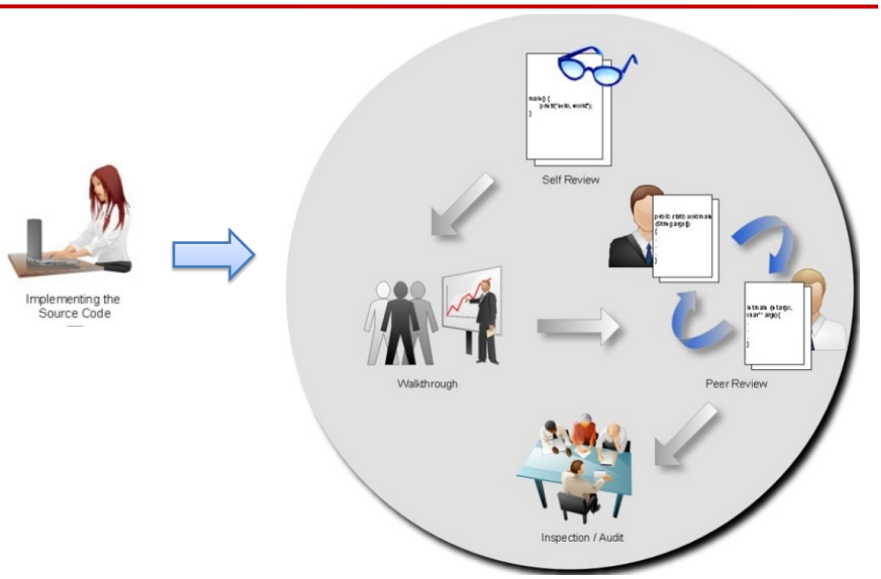
Code Inspection

Walkthrough

Peer Review

Automated

Manual – Static Analysis



Inspection

- A general verification approach
- Earlier applied to code, later to design and requirements
- Can start early in the SDLC
- Complementary to testing
  - Non conformance of the artifact, missing requirements, design defects, inconsistent interface specifications
- Other advantages
  - Increase communication
  - Better understanding
  - Increase productivity
  - Improve quality

Code Inspection (CI)

- Aim is to identify defects in the code
- Generally applied when code is successfully compiled and other for static analysis is performed
- Approaches
  - Check-list based
  - Perspective based
  - Scenario based
  - Stepwise abstraction

An Error Checklist for Inspections

Data Reference Errors (e.g., Does a referenced variable have a value that is unset or uninitialized?)

Data-Declaration Errors

Computation Errors

Comparison Errors

Control-Flow Errors (e.g., Will every loop eventually terminate?)

Interface Errors (e.g., Does the number of parameters received by this module equal the number of arguments sent by each of the calling modules? Also, is the order correct?)

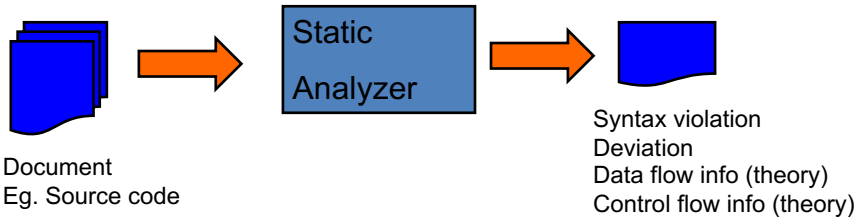
Input/output Errors

9

Automated - Static analysis

- Automated code review techniques
- Focuses on detecting errors in the code without knowing about what it is supposed to do
- Examine the program code and also reasons over all possible behaviors that might arise at the run time.
- This helps the programmers to improve the code and correct the errors before an actual execution of the code.
- Usually performed using software tool

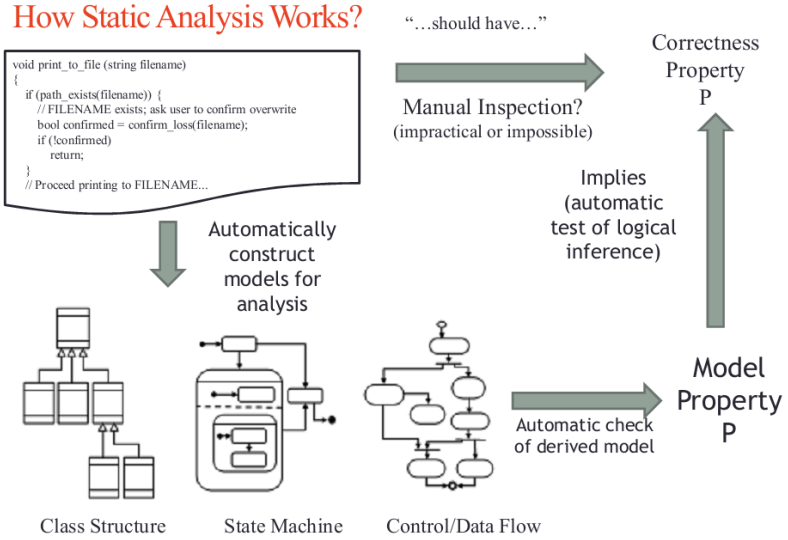
Static analyzers: General form



Static analysis

- Who and when used static analysis tools?
  - Developers
  - Before (Unit testing) and during component or integration testing
    - To check if guidelines or programming conventions are adhered to
  - During integration testing: analyze adherence to interface guidelines
- What are produced by static analysis tools?
  - List of warnings and comments
    - Syntax violation
    - Deviation from conventions and standards
    - Control flow anomalies
    - Data flow anomalies
    - Metrics

How Static Analysis Works?



Compiler as a Static analysis Tool

- Detection of violation of the programming language syntax; reported as a fault or warning
- Further information and other checks
  - Generating a cross reference list of the different program elements (e.g., variables, functions)
  - Checking for correct data type usage by data and variables in programming languages with strict typing
  - Detecting undeclared variables
  - Detecting code that is not reachable
  - Detecting overflow or underflow of field boundaries
  - Checking of interface consistency
  - Detecting the use of all labels as jump start or jump target

Common Coding Errors

Memory Leaks

```
...
try {
   int* pValue = new int();
   if (someCondition) { throw 42; }
   delete pValue;
} catch (int&) { }
...
```

```
void func() {
   char *p = new char[10];
   some_function_which_may_throw(p);
   delete [ ] p;
}
```

```
char* foo(int s)
{
   char *output;
   if (s>0)
      output=(char*) malloc (size);
   if (s==l)
       return NULL; /* if s==l then memory leaked */
    return(output);
}
```

When an application dynamically allocates memory, and does not free that memory when it is finished using it, that program has a ***memory leak***.

Common Coding Errors

Freeing an already freed resource

```
main ()
{
        char *str;
        str = (char *)malloc (10);
        if (global==0)
                free(str) ;
        free (str) ; /* str already freed
}
```

Common Coding Errors

Null dereferencing

```
char *ch=NULL;
if (x>0)
{
        ch='c ' ;
}
printf ("\%C" , *ch);        /* ch may be NULL
*ch=malloc(size);
ch = 'c';            /* ch will be NULL if malloc returns NULL
```

```
switch(i)
{
case 0: s=OBJECT_l; break;
case 1: s=0BJECT_2;break;
}
return(s); /* s not initialized for values other than 0 or 1 */
```

Common Coding Errors

- Synchronization errors
  - Deadlocks, Race conditions
- Array index out of bound
- Arithmetic exceptions
  - Divide by zero, floating point
- String handling errors
- Use of & in place of &&

```
if (object != null & object.getTitle( ) != null)
/* Here second operation can cause a null
dereference */
```

Common Coding Errors

- Buffer Overflow

```
...
char A[8] = {};
unsigned short B = 1979;

strcpy(A, "excessive");
...
```

variable name	A								B	
value	[null string]								1979	
hex value	00	00	00	00	00	00	00	00	07	BB

variable name	A								B	
value	'e'	'x'	'c'	'e'	's'	's'	'l'	'v'	25856	
hex value	65	78	63	65	73	73	69	76	65	00

Discussion

- Why is limiting the scope of a variable a good thing for testing?
- What do you do when you want to share data among many modules of your code?
- What is your opinion of the programming language feature 'declaration upon use'?

Confusion Matrix

A **false positive** is an error in some evaluation process in which a condition tested for is mistakenly found to have been detected.

A false negative is an error in some evaluation process in which wrongly indicates that a particular condition or attribute is absent.
For example, "the test says you aren't ill when you are"

		Predicted Class	
		Yes	No
Actual Class	Yes	TP	FN
	No	FP	TN

Static analysis Tools

- Language Dependent
- e.g., Java (Open Source Tools)
- FindBugs (University of Maryland)
  - http://findbugs.sourceforge.net/
- Google's CodePro Analytics
- PMD
  - http://pmd.sourceforge.net/
- UCDetector
- CheckStyle
  - http://checkstyle.sourceforge.net/

FindBugs

- FindBugs, a program which uses static analysis to look for bugs in Java code.
- It is free (open source) software, distributed under the terms of the Lesser GNU Public License.
- The name FindBugs™ and the FindBugs logo are trademarked by The University of Maryland.
- FindBugs has been downloaded more than a million times.

Example of errors identified by FindBugs

Zero Length Array
```
Int [] zero1 = new int[0];
```

Negative Length Array
```
int [] zero2 = new int[-5];
```

Divide by ZERO
```
int a, b = 9,c = 3;
 a = b / (b%c);
```

Integer Overflow
```
int a2 = 1234567809, b2 = 1234567890;
    int c2 = a2 + b2;
```

Out of bound array indexing
```
int[] array2 = new int[5];
 int b7;
 for(int i = 0 ; i<=array2.length; i++){}
```

Never Executed for Loop
```
for(int i = 2; i <= 1 ; i++ ){}
```

Unexpected behavior of the loop
```
for(int i = 2; i <= 3 ; i-- ){}
```

/* Dead code */
```
for (cl; c2; c3) {
if (C) {
break; }
else {
break; }
stmt; /*this is unreachable*/
```

and
many more.....

Example of errors NOT identified by FindBugs

Needless Leading ZEROs

```
int a4 = 00023;
```

If-else blocks without braces

```
int x = 5, y = 3;
if(x = = y)
    if (y = = 3)
            x=3;
else
    x = 4;
```

```
String four = "four: " + 2 + 2;
System.out.println(four);

int a6=2;
a6 =+ 5;
System.out.println(a6);
```

FindBugs Categories

- Bad practice
- Correctness
- Dodgy
- Experimental
- Internationalization
- Malicious code vulnerability
- Multithreaded correctness
- Performance
- Security

PMD

PMD scans Java source code and looks for potential problems like:

- Possible bugs - empty try/catch/finally/switch statements
- Dead code - unused local variables, parameters and private methods
- Suboptimal code - wasteful String/StringBuffer usage
- Overcomplicated expressions - unnecessary if statements, for loops that could be while loops
- Duplicate code - copied/pasted code means copied/pasted bugs

Different Tools find different Bugs...

```
import java.io.*;
public class foo{
  private byte[] b;
  private int length;
  Foo(){ length = 40;
    b = new byte[length]; }
  public void bar(){
    int y;
    try {
      FileInputStream x =
        new FileInputStream("Z");
      x.read(b,0,length);
      c.close();}
    catch(Exception e){
      System.out.println("Oopsie");}
    for(int i = 1; i <= length; i++){
      if (Integer.toString(50) ==
              Byte.toString(b[i]))
        System.out.print(b[i] + " ");
    }
  }
}
```

variable never used (detect by PMD)

Method result is ignored (detected by FindBugs)

Don't use '==' to compare strings (detected by FindBugs and JLint)

May fail to close stream on exception (detected by FindBugs)

Array index possibly too large (detected by ESC/Java)

Possible null dereference (detected by ESC/Java)

Static Code Checkers

Features	FindBugs	Checkstyle	PMD
Version	0.9.7	4.1	3.6
Works on	Bytecode	Source	Source
Languages	Java	Java	Java
Interface	GUI, command line, plugin	Command line, plugin	Command line, plugin
Detects security vulnerabilities	Few	No	Few
Stack overflow analysis	No	No	No
Custom checkers	Yes	Yes	Yes
Architectural analysis	No	No	No
Metrics	No	Few	No
Web-based project management	HTML reports	HTML reports	HTML reports
Size	3.6 Mbytes	6.8 Mbytes	49.1 Mbytes (of which 48.6 Mbytes is documentation)
License	GNU Lesser General Public License (LGPL)	GNU LGPL	Berkeley Software Distribution-style license

29

Test this??

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

public class CodingHorror {
   public static void main(String args[]) {

     InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    String input = null;
    try {
      input = br.readLine(); // e.g., peel
    } catch (IOException ioex) {
      System.err.println(ioex.getMessage());
    }
    input.replace('e', 'o');
    if (input == "pool") {
      System.out.println("User entered peel.");
    } else {
      System.out.println("User entered something else.");
    }  } }
```

Tool Demonstration

Eclipse plugins (Static Analysis Tools)

1. FindBugs
2. PMD

Issues...

- Tools used to identify bugs in source code often return large numbers of false positive warnings to the user.
- True positive warnings are often buried among a large number of distracting false positives.
- By making the true positives hard to find, a high false positive rate can frustrate users and discourage them from using an otherwise helpful tool.
- In order to understand an error report, users must develop a way to take the information in the report and relate it to the potential problem with the code.
- Experience and vast knowledge about the error (in terms of identifying the false positive and false negative).

Challenges & Summary

- Identify the best tool among all static analysis tools available in the market.
- Create a BIG database of the false alarms suggested by various static analysis tools
- Create your new TOOL (EFFICIENT)

- Static testing can be done to find defect and deviation using:
  - Structured group examinations
    - Reviews
      - Inspection, walkthrough, technical review, informal review
  - Static analysis
    - Compiler
    - Data flow analysis | Control flow analysis

Questions...

Next Lectures....

1. Theory of static analysis (Control flow/Data flow)
2. White box testing
3. Code Coverage