Research Topics in Software Quality

Introduction
## Practical Information

<table>
<thead>
<tr>
<th><strong>goal</strong></th>
<th>study advanced techniques for evaluating, assuring and improving the quality of software</th>
</tr>
</thead>
</table>
| **lectures** | initial focus on foundations  
later on how these are evolving to cope with contemporary software |
| **practical sessions** | applying, evaluating and extending existing techniques |
| **exam** | 3 written reports on assignments (20% each)  
assigned throughout the year, deadline start of exam session  
1 presentation synthesizing at least 2 recent publications (40%)  
selected from a list to be made available on the website |
Reading Software Engineering Papers

what is the motivation for the work?
why is there no trivial solution?
what are the shortcomings of the previous solutions?

what is the proposed solution?
why is it believed to work?
how is it believed to improve upon existing ones?

how is the solution technically achieved?
how is the solution evaluated?
what are the results and their threats to validity?

what is your analysis of the problem, solution and its evaluation?
how solid, novel, applicable are the ideas?
how do the authors and you perceive the contributions and shortcomings of the work?

what is the message to take away?
why will this work still be relevant in the next decade?
are there future directions for this research?

what techniques surveyed in the course can this work be combined?
why would this be a good idea?
are there other application areas for the work?
A Whirlwind Tour of Topics

Topic: Evaluating Design using Metrics

"You cannot control what you cannot measure"

design metrics
- quantify size, complexity, quality of product
to evaluate/control/predict (1) maintainability
- extensibility
- ... utilities of ISO standard

cyclomatic complexity: paths through procedure
coupling of class A: (8) accesses field/method of B

[How to determine these aspects?]
E.g., static binding, dynamic typing, closures in arguments, program evolution?

Topic: Detecting Clones and Plagiarism

copy & paste usually frowned upon

problem: consistently maintaining clones
but sometimes understandable

forking because of platform variation,
boilerplate code in inexpensive language,
templates because of API protocols...
tools can help: linked editors (timed), semantic patches (timed), ...

categorizing clones
- type 1: identical except for layout variations
- type 2: additional variations in identifiers, types, ...
- type 3: inserts, removals and updates

Topic: Debugging and Impact Analysis using Program Slicing

slicing Java requires determining additional dependencies related to threading

available in e.g., Indus tool:

[SCAPM4 keywords]

Topic: Detecting Mechanical Bugs using Static Analysis

e.g., FindBugs: scans methods for bug patterns
- if (name != null && name.length > 3) return;
public String getName()

return name;

String destabilize = getUnmodifiableName();
destabilizing instanceof;

deep or (intra-procedural), but effective

FindBugs hits at Google May 2009:
500 engineers provided 5,000 reviews of 4,000 issues detected by FindBugs
75% were marked should fix or must fix

Topic: Run-time Monitoring of Assertions and Invariants

from simple assertions

assert(balance > 10);

over IML contracts
/\ requires amount A:
\ensure (balance + \text{\texttt{amount}}) \geq \text{\texttt{balance}}
\text{\texttt{amount}}; \needs \text{\texttt{true}} \geq \text{\texttt{balance}}
\text{\texttt{amount}};

public abstract class X {
  boolean b = true;
  if (b) throw new IllegalArgumentException();
  if (b) assignable; // IML assignable
  //
  b = new boolean[1];
}

method pre/post conditions
- caller must ensure precondition
- and assume postcondition upon return

class invariants
- must hold for all publicly visible states

Topic: Generating Test Suites using Symbolic Execution

execute program with symbols for unknown input values
collect constraints over symbols on each executable path
solutions to constraints are values for input that force a test to traverse that path

[How are constraints solved?
path condition to be solved by SAT solver

solution is test that leads to failing assertion]

Plenty of opportunity for your favorite topics
Topic: Evaluating Design using Metrics

"You cannot control what you cannot measure"

[Tom de Marco]

design metrics
quantify size, complexity, quality of product
to evaluate/control/predict(?) maintainability
extensibility
...ilities of ISO standard

cyclomatic complexity: #paths through procedure
coupling of class A: # {B|A accesses field/method of B}

how to determine these accesses?
e.g., late binding, dynamic typing, closures as arguments
program analysis!
interpret metrics

classifying design: Overview Pyramid [Lanza, Marinescu 06]

NOC/NOP: high for Java programs

Inheritance

Size & Complexity

close to high

close to average

close to low

Coupling

how to determine these thresholds?
mining large-scale software repositories!

detecting bad smells: Cultivate [Jancke 10]
Topic: Detecting Clones and Plagiarism

copy & paste usually frowned upon

problem: consistently maintaining clones but sometimes understandable

forking because of platform variation, boilerplate code in inexpressive language, templates because of API protocols, ...

tools can help: linked editors [Toomim04], semantic patches [Padoleau06], ...

categorizing clones

**type 1:** identical except for layout variations

**type 2:** additional variations in identifiers, types, ...

**type 3:** inserts, removals and updates

```java
public int getSoLinger() throws SocketException{
    Object o = impl.getOption(SocketOptions.SO_LINGER);
    if (o instanceof Integer) {
        return ((Integer) o).intValue();
    } else return -1;
}
```
Topic: Detecting Clones and Plagiarism

type 4: semantically similar, independently developed or plagiarized

---

Line 1: 01  static void
Line 2: 02  make_blank (struct line *blank, int count)
Line 3: 03  {
Line 4: 04      int i;
Line 5: 05      unsigned char *buffer;
Line 6: 06      struct field *fields;
Line 7: 07      blank->fields = count;
Line 8: 08      blank->buf.size = blank->buf.length = count + 1;
Line 9: 09      blank->buf.buffer = (char*) malloc (blank->buf.size);
Line 10: 10      buffer = (unsigned char *) blank->buf.buffer;
Line 11: 11      blank->fields = fields =
Line 12: 12          (struct field *) malloc (sizeof (struct field) * count);
Line 13: 13      for (i = 0; i < count; i++){
Line 14: 14          ...  
Line 15: 15      }

Line 1: 01  static void
Line 2: 02  fill_content(int num, struct line* fill)
Line 3: 03  {
Line 4: 04      (*fill).store.size = fill->store.length = num + 1;
Line 5: 05      struct field *tabs;
Line 6: 06      (*fill).fields = tabs = (struct field *)
Line 7: 07          malloc (sizeof (struct field) * num);
Line 8: 08      (*fill).store.buffer = (char*) malloc (fill->store.size);
Line 9: 09      unsigned char *pb;
Line 10: 10      pb = (unsigned char *) (*fill).store.buffer;
Line 11: 11      int idx = 0;
Line 12: 12      while(idx < num){ // fill in the storage
Line 13: 13          ...
Line 14: 14          for(int j = 0; j < idx; j++)
Line 15: 15              ...
Line 16: 16          idx++;
Line 17: 17      }

---

[ how to detect these? ]

subgraph isomorphism in program dependency graphs! (see next topic)

---

[ Liu06 ]
Topic: Debugging and Impact Analysis using Program Slicing

Program dependency graph

- nodes are statements, directed edge from s1 to s2 if
  - control dependency: execution of s2 depends on outcome of s1
  - data dependency: s2 uses value of variable computed in s1

Slicing = reachability in program dependence graph

---

**Example Program**

```plaintext
read(n);
i := 1;
sum := 0;
product := 1;
while i <= n do
    sum := sum + i;
    product := product * i;
i := i + 1
end;
write(sum);
write(product)
```

**Slicing Criterion**

- In grey: all statements that could explain an erroneous value for product at the slicing criterion = backward slice

---

**Algorithm Examples**

1. Bergeretti and Carr
2. Weiser's definition of branch statements with indirect relevance to a slice contains an error
3. Variants of Horwitz, Reps, and Binkley
4. Weiser's original definition of program slicing is based on iterative solution of dataflow equations.
Topic: Debugging and Impact Analysis using Program Slicing

slicing Java requires determining additional dependencies related to threading

available in e.g., Indus tool:

```java
int a = 0;
int b = 0;
int c = 0;

if (_result1 <= 50) {
    a = a + 1;
    b = b + 1;
    c = c + 1;
}

_int_result2 = _f.nextInt(100);

int z = 1;
int y = 1;
int x = 1;

if (_result2 > 50) {
    z = z + 2;
    y = y + 2;
    x = x + 2;
}
```
Topic: Detecting Behavioral Defects using Static Analysis

derive run-time program properties at compile-time

e.g., FindBugs: scans methods for bug patterns

```java
public String foundType() {
    return foundType();
}

String dateString = getHeaderField(name);
dateString.trim();
```

simple (intra-procedural), but effective

FindBugs fixit at Google May 2009:
300 engineers provided 8,000 reviews of 4,000 issues detected by FindBugs
75+% were marked should fix or must fix
Topic: Detecting Behavioral Defects using Static Analysis

e.g., PQL: [Martin05]
looks for sequences of calls that could lead to SQL injection

HttpServletRequest req = /* ... */;
java.sql.Connection conn = /* ... */;
String query = req.getParameter("QUERY");
conn.execute(query);

requires analyzing inter-procedural flow of information

can strings flow unchecked from the user (source) to the database engine? (sink)

computed through points-to analysis!
Topic: Detecting Structural Anti-Patterns using Program Queries

using code templates:
specify characteristics of code through template, leave search to template matcher

e.g., returning null instead of empty collection
**Topic:** Detecting Structural Anti-Patterns using Program Queries

**SQL Antipatterns**
Avoiding the Pitfalls of Database Programming

**J2EE Antipatterns**
R ealistic Patterns in Real World Applications

**Rails Antipatterns**
Get Rails Right, Rails Getting Right a Book by Remove the

*constraints on values for ?inv and ?exp*

pairs of values satisfying constraints

---
e.g., lapsed listeners preventing subject from being reclaimed

**using logic program queries:**
specify characteristics of code in terms of logic conditions, leave search to logic engine
Topic: Run-time Monitoring of Assertions and Invariants

from simple assertions

assert(balance > 10)

over JML contracts

/*@ requires amount>=0;
 @ ensures balance == \old(balance) - amount;
 @ ensures \result == balance */
public int debit(int amount) {...}

public abstract class X {
    boolean[] b;
    //@ invariant b != null && b.length == 6;
    //@ assignable b;
    Invariant() {
        b = new boolean[6];
    }
}
Topic: Run-time Monitoring of Assertions and Invariants

to typestate properties

e.g., from [Bodden10] “After closing a connection c, don’t write to c until c is reconnected.”

monitoring properties at run-time

very precise

congeate sequence of program states is known

even with unknowns in source code (e.g., reflection)

Object unknown = Class.forName(getUserInput()).newInstance();

but incomplete

based on a single run of the program

interesting property might only be exhibited in another run
execute program with symbols for unknown input values
collect constraints over symbols on each executable path
solutions to constraints are values for input that force a test to traverse that path

1: int x,y;
2: if (x>y) {
3:   x = x+y;
4:   y = x-y;
5:   x=x-y;
6:   if (x>y) {
7:     assert false;
}  
8: return;
}

solution is test that leads to failing assertion

path condition to be solved by SAT solver

x=-1, y=54
**Topic: Assessing Test Suites using Mutation Analysis**

**how good is a given test suite?**

usually checked by computing how #statements or #paths of the program are covered
but coverage does not tell anything about the suite’s ability to discern incorrect behavior

**mutation analysis**

seed faults into the system under test (SUT)

through mutation operators that simulate earlier or common mistakes

e.g., change \((x \leq 5)\) into \((x < 5)\)

- AMC - Access Modifier Change
- HVD - Hiding Variable Deletion
- HVI - Hiding Variable Insertion
- OMD - Overriding Method Deletion
- OMM - Overridden Method Moving
- OMR - Overridden Method Rename
- SKR - Super Keyword Deletion
- PCD - Parent Constructor Deletion
- ATC - Actual Type Change
- DTC - Declared Type Change
- PTC - Parameter Type Change
- RTC - Reference Type Change
- OMC - Overloading Method Change
- OMD - Overloading Method Deletion
- AOC - Argument Order Change
- ANC - Argument Number Change
- TKD - this Keyword Deletion
- SMV - Static Modifier Change
- VID - Variable Initialization Deletion
- DCD - Default Constructor 2

**how to do this efficiently?**

analyze behavior of SUT!

analyze how many mutants are killed by the test suite
**Topic: Automating Changes**

<table>
<thead>
<tr>
<th>Type</th>
<th>Eclipse JDT core</th>
<th>Eclipse SWT</th>
<th>Mozilla Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period of development</td>
<td>2001/06 ~ 2009/02</td>
<td>2001/05 ~ 2010/05</td>
<td>1998/03 ~ 2008/05</td>
</tr>
<tr>
<td>Total revisions</td>
<td>17000 revisions</td>
<td>21530 revisions</td>
<td>200000 revisions</td>
</tr>
<tr>
<td># of bugs</td>
<td>1812</td>
<td>1256</td>
<td>11254</td>
</tr>
<tr>
<td># of Type I bugs</td>
<td>1405 (77.54%)</td>
<td>954 (75.96%)</td>
<td>7562 (67.19%)</td>
</tr>
<tr>
<td># of Type II bugs</td>
<td>407 (22.46%)</td>
<td>302 (24.04%)</td>
<td>3692 (32.81%)</td>
</tr>
</tbody>
</table>

- **Figure 1**: The number of times that the same bug is fixed
- **Figure 2**: The number of days taken for the supplementary fix to appear since an initial fix

[“An empirical study of supplementary bug fixes” Park et al. MSR 2012]
Topic: Automating Changes

requires specifying

subjects of the transformation (LHS)

requires specifying

their state afterwards or change actions (RHS)

carefully ensuring

no required changes are missed

no unwarranted changes are applied
Topic: Automating Changes

e.g., generating a Visitor for class hierarchy

```java
public class ?visitedName extends [...]{
    ?bodyVisited}
}
```

```java
package be.ac.chaq.model.ast.java;
?visitedImports

package be.ac.chaq.model.ast.java.visitor;
?visitorImports

public interface IASTVisitor { }

=>

```java
public void acceptVisitor(IASTVisitor visitor){
    visitor.(str "visit" ?visitedName)(this);
}
```

```java
import be.ac.chaq.model.ast.java.IASTVisitor;

import be.ac.chaq.model.ast.java.?visitedName;

import be.ac.chaq.model.ast.java.visitor.IASTVisitor;

public void (str "visit" ?visitedName)(?visitedName o); }
```
Topic: Automating Changes

domain-specific refactorings
refactorings to eliminate or maintain clones

program renovation
convert anonymous classes to Java 8 lambda expressions

API migration
migrate an application from using SWING to using SWT

domain-specific optimizations

List listUsers := [ ]; int i, j = 0;
List users := Query(SELECT * FROM users);
List roles = Query(SELECT * FROM roles);
    while (i < users.size()) {
        while (j < roles.size()) {
            if (users[i].roleid = roles[j].roleid)
                listUsers := append(listUsers, users[i]);
                ++j;
        }
        ++i;
    }

List<User> getRoleUser () {
    List<User> listUsers = db.executeQuery("SELECT u
                                       FROM users u, roles r
                                       WHERE u.roleid = r.roleid
                                       ORDER BY u.roleid, r.roleid");
    return listUsers; }

[Cheung et al., PLDI13]
Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems
Michele Lanza, Radu Marinescu

Managing Duplicated Code with Linked Editing
VL/HCC 2004
Michael Toomim, Andrew Begel, Susan L. Graham

Semantic patches for documenting and automating collateral evolutions in Linux device drivers. PLOS06
Yoann Padioleau, René Rydhof Hansen, Julia L. Lawall, Gilles Muller

http://sewiki.iai.uni-bonn.de/research/cultivate/

GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis
KDD06
Chao Liu, Chen Chen, Jiawei Han, Philip S. Yu

A survey of program slicing techniques
JOPL Frank Tip

An Overview of the Indus Framework for Analysis and Slicing of Concurrent Java Software
SCAM06
Venkatesh Prasad Ranganath, John Hatcliff

The Google Findbugs Fixit
ISSTA10
Nathaniel Ayewah, William Pugh

Finding Application Errors and Security Flaws Using PQL: a Program Query Language
OOPSLA05
Michael Martin, Benjamin Livshits, Monica S. Lam

Clara: Partially Evaluating Runtime Monitors at Compile Time - Tutorial Supplement
RV10
Eric Bodden, Patrick Lam