Evolving mutation from objects to the cloud

MUTATION workshop, Berlin, March 2011

Benoit Baudry
Outline

• A perspective on testing in evolving software construction paradigms

• A methodological pattern: question-learn-test-feedback

• An illustration: aspect-oriented programming
1. The scope of software testing practice and research is broadening
The scope of software testing practice and research is widening
1. The scope of software testing practice and research is widening
1. The scope of software testing practice and research is widening.
1. The scope of software testing practice and research is widening.
The scope of software testing practice and research is widening.
The scope of software testing practice and research is widening.
1. The scope of software testing practice and research is widening.
1. The scope of software testing practice and research is widening.

1.1. Purpose:
- Beta
- System
- Unit
- Integration
- Usability
- Performance
- Security

1.2. Target:
- Statement
- Scenario
- MC/DC
- Environment
- Mutation
- Requirements

1.3. Techniques:
- Test plan
- Prioritization
- Impact analysis
- Selection
- Conformance
- Data generation
1. The scope of software testing practice and research is widening.
The scope of software testing practice and research is widening.
1. The scope of software testing practice and research is widening.

- Techniques: beta, system, unit, integration, usability, performance, security, acceptance, regression, scenario, state, environment, mutation, requirements, statement.

- Target: beta, system, unit, integration, usability, performance, security, acceptance, regression, scenario, state, environment, mutation, requirements, statement.

- Purpose: integration, usability, performance, security, acceptance, regression, scenario, state, environment, mutation, requirements, statement.

- Jobs: dev-tester, engineer, trainer, designer, QA architect, test plan, prioritization, impact analysis, selection, data generation, conformance.

- Our goal is to write bug-free software. I'll pay a ten-dollar bonus for every bug you find and fix.

- YAHOO! WE'RE RICH! YES!!! YES!!! YES!!!

- I hope this drives the right behavior.

- I'm gonna write me a new minivan this afternoon.
The scope of software testing practice and research is widening. Various testing techniques are employed for different purposes such as unit, integration, system, and beta testing. Acceptance testing is used to verify that the software meets the requirements. Security, usability, and performance testing are also critical. Beta testing involves users to find bugs before release.

For system testing, scenario-based testing is utilized. Testing statements such as MC/DC are applied to ensure code coverage. Environment variables and state mutation are also considered. Requirements are prioritized based on impact analysis.

Tools like JMeter and EasyMock are used to automate testing. Test plans are created to guide the testing process. Prioritization of tests is important to ensure that critical issues are addressed first. Selection of test cases is a crucial step in the testing process.
The scope of software testing practice and research is widening.

**Purpose**
- beta
- acceptance
- regression
- security
- usability
- performance
- unit
- integration
- system

**Target**
- statement
- state
- scenario
- environment
- mutation
- MC/DC
- requirements

**Tools**
- test plan
- prioritization
- impact analysis
- selection
- data generation
- conformance

**Jobs**
- dev-tester
- integration
- engineer
- designer
- QA architect
- trainer

**Techniques**
- MC/DC
- test plan
- prioritization
- impact analysis
- selection
- data generation
- conformance
2. because of (i) new ways of building software systems to (ii) face the growing diversity of applications and requirements for software systems
2. because of (i) new ways of building software systems to
2. because of (i) new ways of building software systems to

processes
2. because of (i) new ways of building software systems to
2. because of (i) new ways of building software systems to
2. because of (i) new ways of building software systems to
2. because of (i) new ways of building software systems to

languages
2. because of (i) new ways of building software systems to
2. because of (i) new ways of building software systems to economics
2. because of (i) new ways of building software systems to

innovation

cloud

outsourcing

agility

economics

end-user

security

scalability

critical

end-user

processes

constraints

V-cycle

innovation factory

CMMI

SCRUM

sprint

distributed

security

cloud

languages

OOP

AOP

Procedural

UML

CP

CBSE

end-user

DSL

DSML

innovation

outsourcing

agility

innovation}

outsourcing

agility

cloud
because of (i) new ways of building software systems to...

- Innovation factory
- SCRUM
- V-cycle
- CMMI
- spiral
- distributed
- embedded
- critical
- end-user
- security
- scalability
- innovation
- outsourcing
- economics
- lean
- agility
- cloud
- SOA
- OOP
- AOP
- Procedural
- UML
- CP
- CBSE
- end-user
- DSL
- DSML
because of (i) new ways of building software systems to
processes
Innovation factory
SCRUM
V-cycle
CMMI
spiral

constraints
end-user
distributed
security
scalability
critical
embedded

end-user economics
cloud
innovation
outsourcing
lean
agility

environments
maven
eclipse
spring
GWT
sonar
Cradle
MATLAB
Simulink

languages
OOP
AOP
Procedural
UML
CP
CBSE
end-user
DSL
DSML

V-cycle
CMMI
spiral
because of (i) new ways of building software systems to process innovation
Innovation factory
V-cycle
SCRUM
CMMI
spiral
distributed
embedded
critical
security
end-user
scalability

HOW'S THE CODING COMING ALONG?
NO PROBLEM UNLESS . . .

... SOME MORON TRIES TO STANDARDIZE ON A NEW PROGRAMMING METHODOLOGY IN THE MIDDLE OF THE PROJECT.

WHAT IF IT'S ME INSTEAD OF SOME MORON GUY?

cloud
innovation
economics
economics
outsourcing
agility

OOP AOP Procedural
CP CBSE UML
DSL
end-user DSML
(ii) face the growing diversity of applications and requirements for software systems
(ii) face the growing diversity of applications and requirements for software systems
(ii) face the growing diversity of applications and requirements for software systems.
(ii) face the growing diversity of applications and requirements for software systems
face the growing diversity of applications and requirements for software systems
face the growing diversity of applications and requirements for software systems
face the growing diversity of applications and requirements for software systems
3. The identification of core principles underlying this apparent seething landscape is a major challenge for research in software engineering.
A visual representation

Key principles


• Rigor and formality
• Separation of concerns
• Modularity
• Abstraction
• Anticipation of change
• Generality
• Incrementality
Research in Software Engineering

Research in Software Engineering

Research in Software Engineering

4. The confrontation of these core principles with the broadening scope of software leads to the emergence of new software construction paradigms.
Software construction paradigms

• Object-oriented programming
• Aspect-oriented programming
• Model-driven development
• Aspect-oriented modelling
• Multi-paradigm modelling
Software testing research for emerging paradigms extends beyond error detection to analyze and include the context imposed by these paradigms.
Agility and Test Driven Development

- Write unit test
- Run test
  - fail
- refactor
- Run test
  - pass
- Write code

Write unit test
Run test ➔ fail
Run test ➔ pass
Write code
refactor

MDD and MBT

- Requirements
- Design models
- System
- Test model
- Test cases
QLTF pattern

Questions

Software construction paradigm

Learn

Test

Feedback
Mutation in QLTF

what can go wrong?
what type of errors are we looking for?
what can mitigate/increase error risks?

Software construction paradigm
Mutation in QLTF

Questions

what are the frequent faults?
what usages trigger faults?
what practices to detect faults?

Software construction paradigm

Test
Feedback
Questions

Software construction paradigm

how can we assist fault detection?

do our techniques detect faults?

are new adequacy criteria adequate?
Mutation in QLTF

Questions

Software construction paradigm

how can we prevent faults?
how can testability be improved?
how can test be integrated in dev. processes?
Aspect oriented programming
An example: AOP

class Point implements FigureElement {
    private int x = 0, y = 0;

    int getX() { return x; }
    int getY() { return y; }
    void setX(int x) {
        this.x = x;
    }
    void setY(int y) {
        this.y = y;
    }
    void moveBy(int dx, int dy) {
        ...
    }
}

class Line implements FigureElement{
    private Point p1, p2;

    Point getP1() { return p1; }
    Point getP2() { return p2; }
    void setP1(Point p1) {
        this.p1 = p1;
    }
    void setP2(Point p2) {
        this.p2 = p2;
    }
    void moveBy(int dx, int dy) {
        ...
    }
}

aspect DisplayUpdating {
    pointcut move():
        call(void FigureElement.moveBy(int, int)) ||
        call(void Line.setP1(Point)) ||
        call(void Line.setP2(Point)) ||
        call(void Point.setX(int)) |
        call(void Point.setY(int));
    after(): move() {
        Display.update();
    }
}
An example: AOP

class Point implements FigureElement {
    private int x = 0, y = 0;

    int getX() { return x; }
    int getY() { return y; }
    void setX(int x) {
        this.x = x;
    }
    void setY(int y) {
        this.y = y;
    }
    void moveBy(int dx, int dy) {
        ...
    }
}

class Line implements FigureElement {
    private Point p1, p2;

    Point getP1() { return p1; }
    Point getP2() { return p2; }
    void setP1(Point p1) {
        this.p1 = p1;
    }
    void setP2(Point p2) {
        this.p2 = p2;
    }
    void moveBy(int dx, int dy) {
        ...
    }
}

aspect DisplayUpdating {
    pointcut move():
        call(void FigureElement.moveBy(int, int)) ||
        call(void Line.setP1(Point)) ||
        call(void Line.setP2(Point)) ||
        call(void Point.setX(int)) ||
        call(void Point.setY(int));

    after(): move() {
        Display.update();
    }
}
An example: AOP

class Point implements FigureElement {
    private int x = 0, y = 0;

    int getX() { return x; }
    int getY() { return y; }
    void setX(int x) {
        this.x = x;
    }
    void setY(int y) {
        this.y = y;
    }
    void moveBy(int dx, int dy) {
        ...
    }
}
class Line implements FigureElement{
    private Point p1, p2;
    Point getP1() { return p1; }
    Point getP2() { return p2; }
    void setP1(Point p1) {
        this.p1 = p1;
    }
    void setP2(Point p2) {
        this.p2 = p2;
    }
    void moveBy(int dx, int dy) {
        ...
    }
}

aspect DisplayUpdating {
    pointcut move():
        call(void FigureElement.moveBy(int, int)) ||
        call(void Line.setP1(Point)) ||
        call(void Line.setP2(Point)) ||
        call(void Point.setX(int)) ||
        call(void Point.setY(int));
    after(): move() {
        Display.update();
    }
}
An example: AOP

class Point implements FigureElement {
    private int x = 0, y = 0;
    
    int getX() { return x; }
    int getY() { return y; }
    void setX(int x) {
        this.x = x;
    }
    void setY(int y) {
        this.y = y;
    }
    void moveBy(int dx, int dy) {
        ...
    }
}

class Line implements FigureElement{
    private Point p1, p2;
    
    Point getp1() { return p1; }
    Point getp2() { return p2; }
    void setp1(Point p1) {
        this.p1 = p1;
    }
    void setp2(Point p2) {
        this.p2 = p2;
    }
    void moveBy(int dx, int dy) {
        ...
    }
}

class DisplayUpdating {
    aspect Pointcut move() {
        pointcut move():
            call(void FigureElement.moveBy(int, int)) ||
            call(void Line.setP1(Point)) ||
            call(void Line.setP2(Point)) ||
            call(void Point.setX(int)) ||
            call(void Point.setY(int));
    after(): move() {
        Display.update();
    }
}
An example: AOP

class Point implements FigureElement {
    private int x = 0, y = 0;
    
    int getX() { return x; }
    int getY() { return y; }
    void setX(int x) {
        this.x = x;
    }
    void setY(int y) {
        this.y = y;
    }
    void moveBy(int dx, int dy) {
        ...;
    }
}
class Line implements FigureElement {
    private Point p1, p2;
    
    Point getP1() { return p1; }
    Point getP2() { return p2; }
    void setP1(Point p1) {
        this.p1 = p1;
    }
    void setP2(Point p2) {
        this.p2 = p2;
    }
    void moveBy(int dx, int dy) {
        ...
    }
}

aspect DisplayUpdating {
    pointcut move():
        call(void FigureElement.moveBy(int, int)) ||
        call(void Line.setP1(Point)) ||
        call(void Line.setP2(Point)) ||
        call(void Point.setX(int)) ||
        call(void Point.setY(int));
    after(): move() {
        Display.update();
    }
}
public aspect AccessControl {
    pointcut controlledAccess():
        execution(* Account.*(int));

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
Withdraw

(a)

Account.withdraw

if(amount>=balance)

T
balance -= amount;

F
return false;

return true;

(b)

Account.withdraw

if(checkAccess(thisjoinpoint.getTarget()))

T
throw(new DeniedAccessException());

F
if(amount>=balance)

T
balance -= amount;

F
return false;

return true;
Faults in PCD
Faults in PCD

correct PCD

- Intended
- Matched
Faults in PCD

Correct PCD

Both neglected and unintended

- Intended
- Matched
Faults in PCD

- Correct PCD
- Both neglected and unintended
- Neglected joinpoints

Legend:
- Intended
- Matched
Faults in PCD

Correct PCD

Both neglected and unintended

Neglected joinpoints

Unintended joinpoints

Intended

Matched
both neglected and unintended

pointcut controlledAccess(): execution(* Bank.*(int));
matches deleteAccount() and getAccount()

pointcut controlledAccess(): execution(boolean Bank.*(int));
matches deleteAccount() and login()
neglected joinpoints

pointcut controlledAccess(): execution(* Account.*(int)); matches withdraw() and deposit()

pointcut controlledAccess(): execution(boolean Account.*(int)); matches withdraw()
Mutant PCD

• A PCD where a fault has been inserted
  • Selects a different set of joinpoints

• Equivalent mutant
  • Mutant that matches the same set of joinpoint
  • Equivalent mutants can be detected statically
## Mutation Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCCC</td>
<td>Replaces a <code>cflow</code> by a <code>cflowbelow</code>, or the contrary</td>
</tr>
<tr>
<td>PCCE</td>
<td>Replaces a <code>call</code> by an <code>execution</code>, or the contrary</td>
</tr>
<tr>
<td>PCGS</td>
<td>Replaces a <code>get</code> by a <code>set</code>, or the contrary</td>
</tr>
<tr>
<td>PCLO</td>
<td>Changes the logical operators in a composition of PCDs</td>
</tr>
<tr>
<td>PCTT</td>
<td>Replaces a <code>this</code> by a <code>target</code>, or the contrary</td>
</tr>
<tr>
<td>POEC</td>
<td>Adds, removes or changes throwing clauses</td>
</tr>
<tr>
<td>POPL</td>
<td>Changes the parameter list</td>
</tr>
<tr>
<td>PSWR</td>
<td>Removes wildcards</td>
</tr>
<tr>
<td>PWAR</td>
<td>Removes annotation from type, field or method patterns</td>
</tr>
<tr>
<td>PWIW</td>
<td>Adds wildcards</td>
</tr>
</tbody>
</table>
Testing the PCD: an example – 1

```java
public aspect AccessControl {
    pointcut controlledAccess(): execution(* Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if (!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
```
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser,password);
    try {
        account.withdraw(30);
        fail("Access should not be authorized");
    } catch(DeniedAccessException) {}
}

Bank
+ login(String,String): boolean
+ deleteAccount(int): boolean

Account
+ withdraw(int): boolean

accounts *
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    try {
        account.withdraw(30);
        fail("Access should not be authorized");
    }
    catch (DeniedAccessException() {})
}
public aspect AccessControl {
    pointcut controlledAccess(): execution(void Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    try {
        account.withdraw(30);
        fail("Access should not be authorized");
    } catch (DeniedAccessException) {}
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    try {
        account.withdraw(30);
        fail("Access should not be authorized");
    } catch (DeniedAccessException e) {
    }
}
Testing the PCD: example – 3

```java
public aspect AccessControl {
    pointcut controlledAccess(): execution(* Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if (checkAccess(thisJoinPoint.getTarget())) {
            throw new DeniedAccessException();
        }
    }
}
```
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    try {
        account.withdraw(30);
        fail("Access should not be authorized");
    } catch (DeniedAccessException() {})
}
Testing the PCD: example – 3

```java
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    try {
        account.withdraw(30);
        fail("Access should not be authorized");
    } catch (DeniedAccessException) {}
}
```
public aspect AccessControl {
    pointcut controlledAccess(): execution(* *(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget())) {
            throw new DeniedAccessException();
        }
    }
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    try {
        account.withdraw(30);
        fail("Access should not be authorized");
    } catch (DeniedAccessException() {})
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    try {
        account.withdraw(30);
        fail("Access should not be authorized");
    } catch (DeniedAccessException) {}
Crosscutting aspects

38 open source projects:
479 aspects, 522 advices in 21245 join-points
AdviceTracer: writing PCD specific Oracles

• AdviceTracer: AspectJ library to write oracles that specifically target the PCD
• Oracles written with AdviceTracer for one PCD are:
  • Independent from the advices
  • Independent from other PCDs
public aspect AccessControl {
    pointcut controlledAccess(): execution(* Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser,password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch(DeniedAccessException() {})
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl","Account.withdraw:47");
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch (DeniedAccessException() {})
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl","Account.withdraw:47");
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch (DeniedAccessException() {})
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl", "Account.withdraw:47");
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch (DeniedAccessException() {})
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl", "Account.withdraw:47");
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch (DeniedAccessException() {}) 
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl","Account.withdraw:47");
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser,password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch (DeniedAccessException() {})
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl","Account.withdraw:47");
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser,password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch (DeniedAccessException() {}) 
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl","Account.withdraw:47");
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser,password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch(DeniedAccessException() {})
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl","Account.withdraw:47");
}
public aspect AccessControl {
    pointcut controlledAccess(): execution(void Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if (!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch (DeniedAccessException() {}
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl", "Account.withdraw:47");
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch (DeniedAccessException() {})
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl","Account.withdraw:47");
}
public aspect AccessControl {
    pointcut controlledAccess(): execution(* Account.*(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch (DeniedAccessException() {})
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl", "Account.withdraw:47");
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch (DeniedAccessException() {})
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl","Account.withdraw:47");
}
public aspect AccessControl {
    pointcut controlledAccess(): execution(* *(int))

    @AdviceName("AccessControl")
    before(): controlledAccess() {
        if(!checkAccess(thisJoinPoint.getTarget()))
            throw new DeniedAccessException();
    }
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser,password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch (DeniedAccessException() {})
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl","Account.withdraw:47");
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser,password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    try {
        account.withdraw(30);
    } catch(DeniedAccessException() { }
    setAdviceTracerOff();
    assertAdviceExecutionEquals(1);
    assertExecutedAdvice("AccessControl","Account.withdraw:47");
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser,password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    // System Test
    setAdviceTracerOff();
    assertAdviceExecutionEquals(4);
}
public void testAccessControlDelete() {
    bank.login(nonAuthorizedUser, password);
    addTracedAdvice("AccessControl");
    setAdviceTracerOn();
    // System Test
    setAdviceTracerOff();
    assertAdviceExecutionEquals(4);
}
QLTF for AOP

which faults?

Software construction paradigm

usage of PCD
AdviceTracer mutation tool
contracts for AOP
Cloud computing

• Build order of magnitude bigger systems with order of magnitude less efforts
  • http://boom.cs.berkeley.edu/
• Distribution, communication and synchronization are hidden
  • establish trust in these mechanisms
• Design applications as atoms and bits
  • reason on pieces of programs
  • understand the interactions between these pieces and the cloud framework
MapReduce
/**
 * Counts the words in each line.
 * For each line of input, break the line into words and emit them as
 * (<b>word</b>, <b>1</b>).
 */

public static class MapClass extends MapReduceBase
  implements Mapper<LongWritable, Text, Text, IntWritable> {

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(LongWritable key, Text value,
                   OutputCollector<Text, IntWritable> output,
                   Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}
/**
  * A reducer class that just emits the sum of the input values.
  */

public static class Reduce extends MapReduceBase
  implements Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator<IntWritable> values,
                     OutputCollector<Text, IntWritable> output,
                     Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
MapReduce application

• Two independents parts
  • Mapper processes a bit of information; 1-to-1 input/output sources
  • Reducer processes n mapped bits of information in a global result; n-to-1 I/O sources
• + configuration, dispatch modules
• **Map and reduce modules rely on**
  • **high level languages**
    • low level arithmetic, control, etc. are hidden from the programmer
  • **that have been heavily tested**
    • i.e., we can trust the iterator Java library
  • **framework that abstracts I/O and distribution operations**
    • all errors are handled by the framework (e.g., Hadoop)
MapReduce faults

• Some faults in map and reduce
  • logic, arithmetic

• Most of them elsewhere
  • ‘cutting’ a function in map and reduce is a hard design task
  • understanding the MapReduce framework is hard
Conclusion

- New ways of building software
- require new ways of testing it
- Mutation analysis can play a central role
  - understanding faults in the methods and techniques
  - focusing testing on relevant defects
  - tailoring testing research questions