5th International Workshop on Quantitative Approaches to Software Quality

Automatically Identifying Dead Fields in Test Code by Resolving Method Call and Field Dependency

Presented by

Abdus Satter

Institute of Information Technology

University of Dhaka, Dhaka 1000, Bangladesh

Outline

- Broad Domain – Test Smell, Dead Field
- Problem Specification
- Literature Review
- Research Question
- Proposed Technique
- Result Analysis
- Conclusion

Test Smell

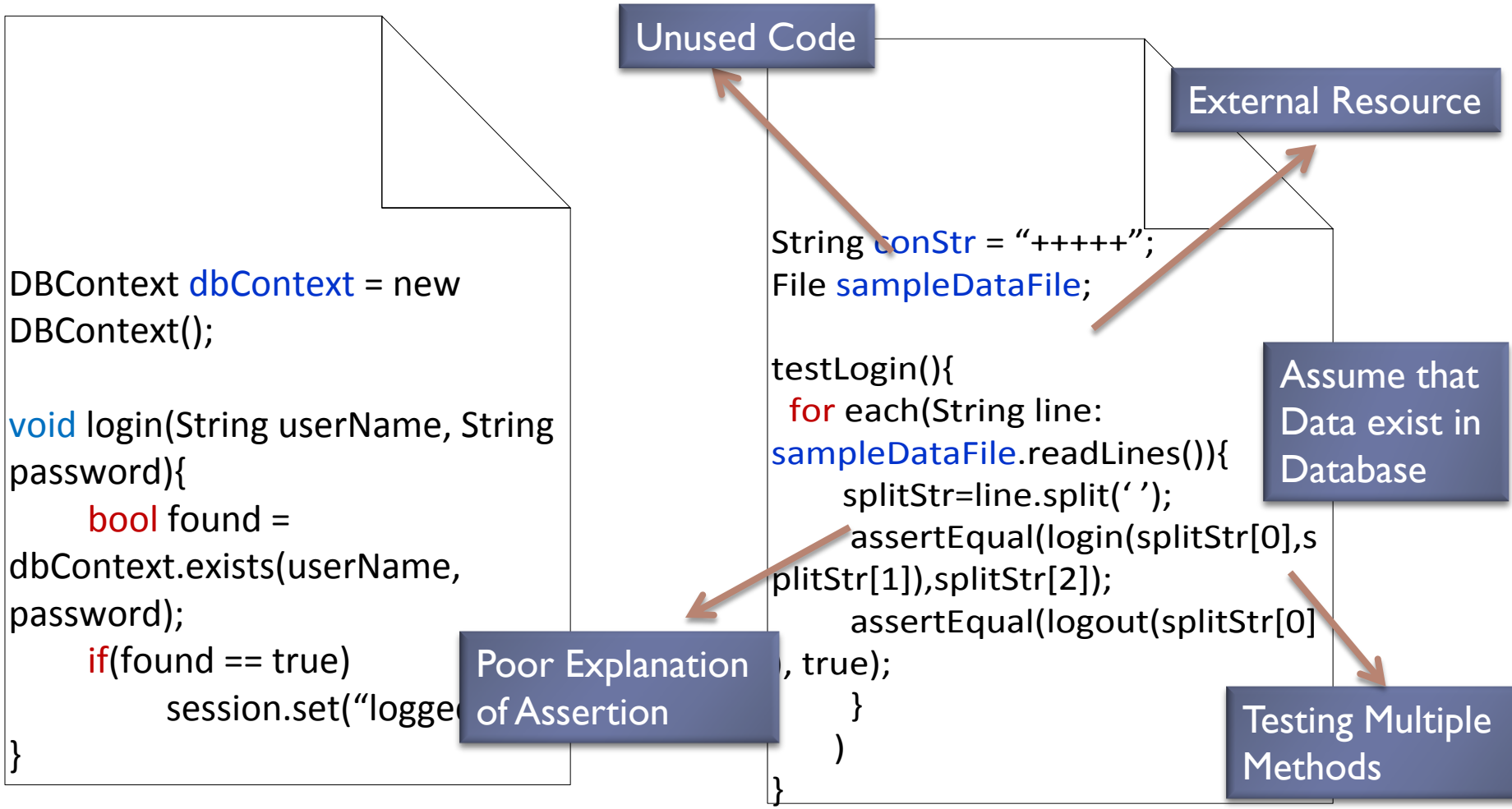
```
DBContext dbContext = new
DBContext();

void login(String userName, String
password){
        bool found =
dbContext.exists(userName,
password);
        if(found == true)
                session.set("logged_in");
}
```

Production Code

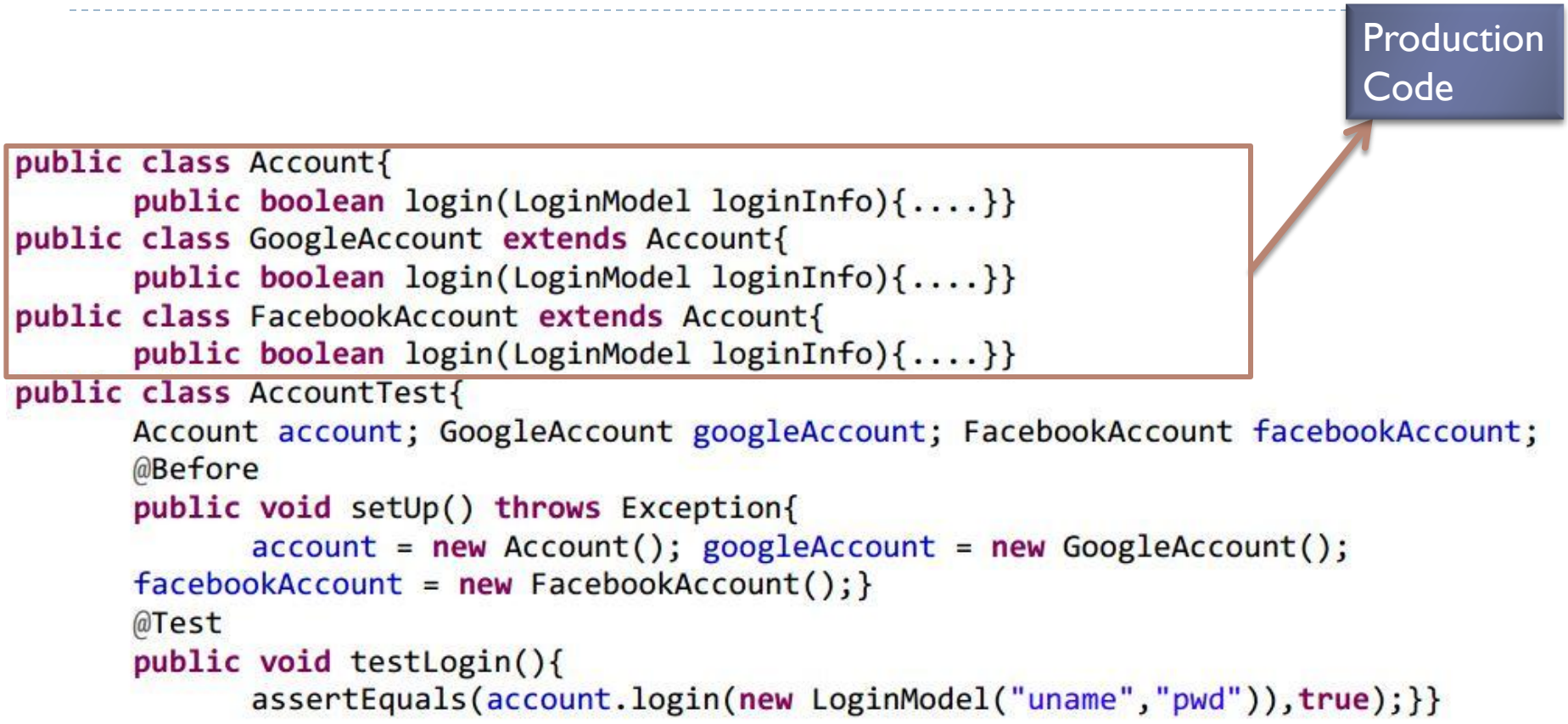
```
String conStr = "+++++";
File sampleDataFile;

testLogin(){
        for each(String line:
sampleDataFile.readLines()){
        splitStr=line.split(' ');
        assertEqual(login(splitStr[0],s
plitStr[1],splitStr[2]));
        }
}
```

Test Code

Test Smell


DBContext dbContext = new DBContext();
void login(String userName, String password){
bool found = dbContext.exists(userName, password);
if(found == true)
session.set("logge
}
String conStr = "+++++";
File sampleDataFile;
testLogin(){
for each(String line: sampleDataFile.readLines()){
splitStr=line.split(' ');
assertEqual(login(splitStr[0],splitStr[1]),splitStr[2]);
assertEqual(logout(splitStr[0]
, true);
}
)
}
Unused Code
External Resource
Assume that Data exist in Database
Poor Explanation of Assertion
Testing Multiple Methods

Dead Field


Production Code
public class Account{
public boolean login(LoginModel loginInfo){....}}
public class GoogleAccount extends Account{
public boolean login(LoginModel loginInfo){....}}
public class FacebookAccount extends Account{
public boolean login(LoginModel loginInfo){....}}
public class AccountTest{
Account account; GoogleAccount googleAccount; FacebookAccount facebookAccount;
@Before
public void setUp() throws Exception{
account = new Account(); googleAccount = new GoogleAccount();
facebookAccount = new FacebookAccount();}
@Test
public void testLogin(){
assertEquals(account.login(new LoginModel("uname","pwd")),true);}}

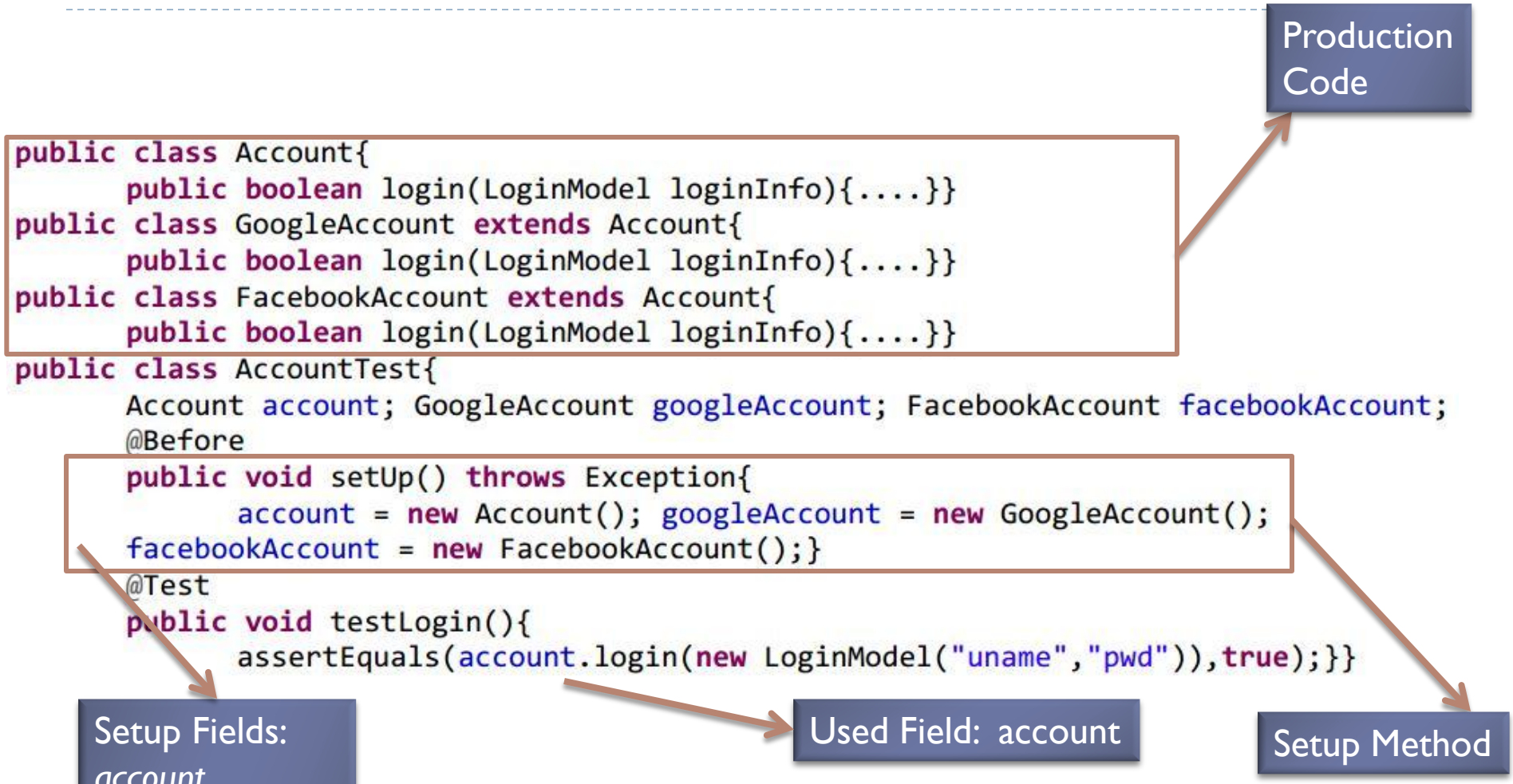
Dead Field


Production Code
public class Account{
public boolean login(LoginModel loginInfo){....}}
public class GoogleAccount extends Account{
public boolean login(LoginModel loginInfo){....}}
public class FacebookAccount extends Account{
public boolean login(LoginModel loginInfo){....}}
public class AccountTest{
Account account; GoogleAccount googleAccount; FacebookAccount facebookAccount;
@Before
public void setUp() throws Exception{
account = new Account(); googleAccount = new GoogleAccount();
facebookAccount = new FacebookAccount();}
@Test
public void testLogin(){
assertEquals(account.login(new LoginModel("uname","pwd")),true);}}
Setup Method

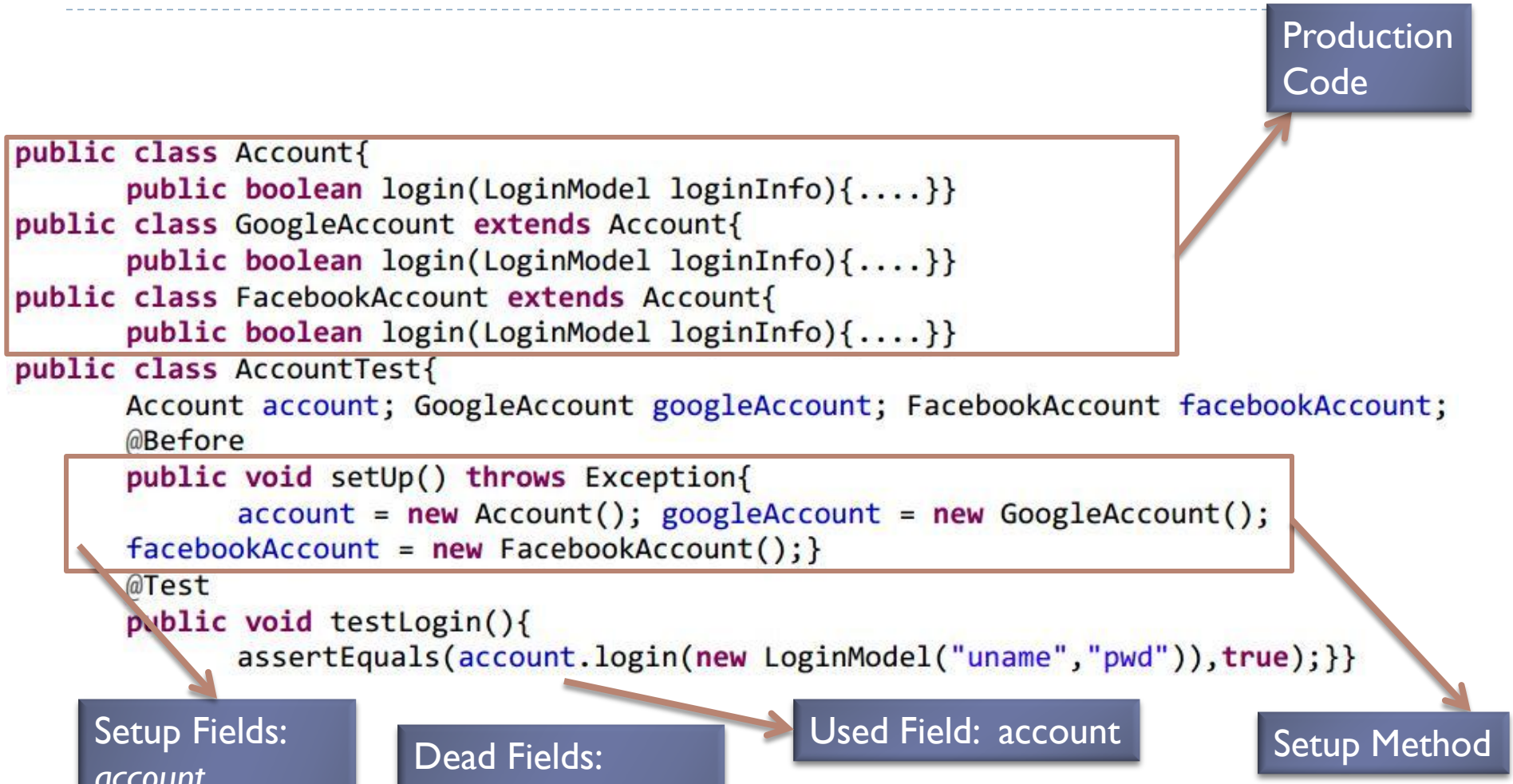
Dead Field


Production Code
public class Account{
public boolean login(LoginModel loginInfo){....}}
public class GoogleAccount extends Account{
public boolean login(LoginModel loginInfo){....}}
public class FacebookAccount extends Account{
public boolean login(LoginModel loginInfo){....}}
public class AccountTest{
Account account; GoogleAccount googleAccount; FacebookAccount facebookAccount;
@Before
public void setUp() throws Exception{
account = new Account(); googleAccount = new GoogleAccount();
facebookAccount = new FacebookAccount();}
@Test
public void testLogin(){
assertEquals(account.login(new LoginModel("uname","pwd")),true);}}
Setup Fields:
account
facebookAccount
googleAccount
Setup Method

Dead Field


Production Code
public class Account{
public boolean login(LoginModel loginInfo){....}}
public class GoogleAccount extends Account{
public boolean login(LoginModel loginInfo){....}}
public class FacebookAccount extends Account{
public boolean login(LoginModel loginInfo){....}}
public class AccountTest{
Account account; GoogleAccount googleAccount; FacebookAccount facebookAccount;
@Before
public void setUp() throws Exception{
account = new Account(); googleAccount = new GoogleAccount();
facebookAccount = new FacebookAccount();}
@Test
public void testLogin(){
assertEquals(account.login(new LoginModel("uname","pwd")),true);}}
Setup Fields:
account
facebookAccount
googleAccount
Used Field: account
Setup Method

Dead Field


Production Code
public class Account{
public boolean login(LoginModel loginInfo){....}}
public class GoogleAccount extends Account{
public boolean login(LoginModel loginInfo){....}}
public class FacebookAccount extends Account{
public boolean login(LoginModel loginInfo){....}}
public class AccountTest{
Account account; GoogleAccount googleAccount; FacebookAccount facebookAccount;
@Before
public void setUp() throws Exception{
account = new Account(); googleAccount = new GoogleAccount();
facebookAccount = new FacebookAccount();}
@Test
public void testLogin(){
assertEquals(account.login(new LoginModel("uname","pwd")),true);}}
Setup Fields:
account
facebookAccount
googleAccount
Dead Fields:
facebookAccount
googleAccount
Used Field: account
Setup Method

Impact of Dead Field

- Increase line of code
- Decrease maintainability
- Make the code difficult to comprehend
- Increase the amount of production code in Test Driven Development (TDD) approach

Refactoring: improving the design of existing code[1]

- Martin Fowler first introduced code smell
- Discussed impact of code smells in software maintainability
- Provided refactoring mechanism to remove code smells

Refactoring Test Code[2]

- van Deursen first introduced term test smell
- Identified eleven test smells based on the concept of code smell
- Proposed different techniques to remove test smells from test code
- However, no technique was provided to detect dead field because this smell was not discovered at that time

An Empirical Analysis of the Distribution of Unit Test Smells[3]

- Two studies were carried out by Bavota et al. on 18 software systems with twenty masters students
- Explained the distribution of test smells in test code
- Described the impact of test smells in test code comprehension in the controlled experiment
- However, they did not provide any information regarding dead field

Testq: Exploring structural and maintenance characteristics of unit test suites[4]

- Tool presented by Bart Van Rompaey for exploring structural maintenance characteristics of test code
- Eleven different test smells like assertionless, eager test and so on could be identified by the tool
- Uses test smell characteristics for test smell detection
- Visualization feature helps to explore test suites visually
- The tool could not identify dead field as no metric was defined for the detection

Rule-based assessment of test quality[5]

- This rule based tool proposed by Stefan Reichart for assessing test code quality
- The tool parses the source code, analyzes the source tree, detects pattern and identifies smell
- Pattern detection involved test smell characteristics
- The tool could not detect dead fields as no rule was provided for it

Automated Detection of Test Fixture Strategies and Smells[6]

- Introduced five new test smells
  - Test Maverick
  - Dead Fields
  - Lack of Cohesion of Test Methods
  - Obscure In-Line Setup
  - Vague Header setup
- Implemented a tool named TestHound to detect these smells
- Could identify test smells well but could not identify dead field correctly due to not considering field dependency and usage of setup fields properly

Research Question

- How to develop a process to identify dead fields automatically by analyzing method call and field dependency in test methods?

Dead Field Detector (DFD)

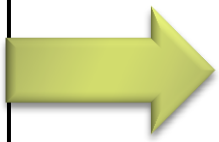
- The technique comprises four steps –
  - Call Graph Generation
  - Data Dependency Graph Generation
  - Setup Field Detection
  - Dead Field Detection

Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20;
int z;
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
   return y*y-2*y;
}
int m4(){
   return p+x-y;
}
```

Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20; int z;
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
   return y*y-2*y;
}
int m4(){
   return p+x-y;
}
```



```
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
```

```
int m2(int a){
   return m3()-2*a;
}
```

```
int m3(){
   return y*y-2*y;
}
```

```
int m4(){
   return p+x-y;
}
```

Parsing Methods

Dead Field Detector (DFD)

```
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
```

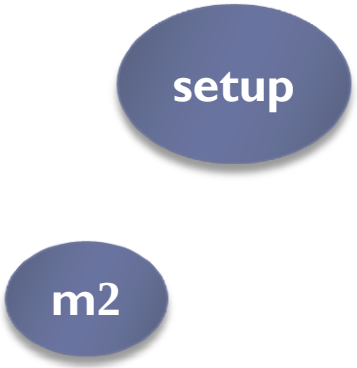
Call Graph Generation

Dead Field Detector (DFD)

```
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
```

```
int m2(int a){
   return m3()-2*a;
}
```

Call Graph Generation



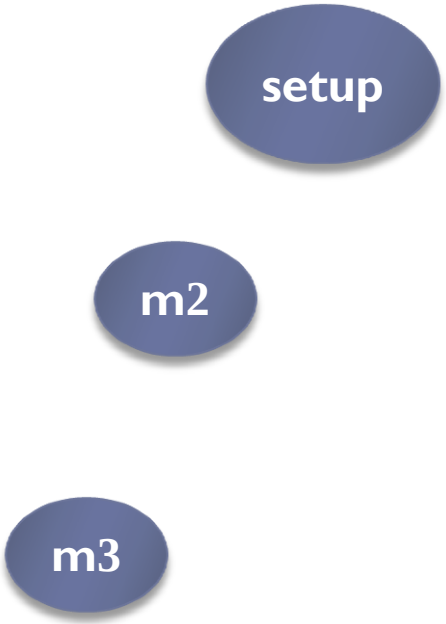
Dead Field Detector (DFD)

```
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
```

```
int m2(int a){
   return m3()-2*a;
}
```

```
int m3(){
   return y*y-2*y;
}
```

Call Graph Generation



Dead Field Detector (DFD)

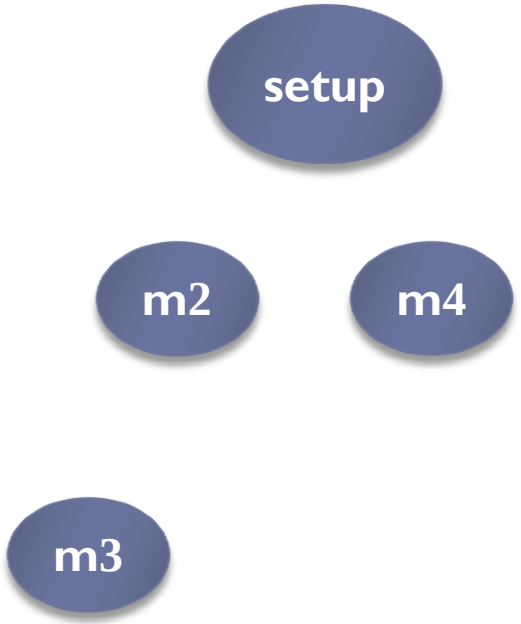
```
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
```

```
int m2(int a){
   return m3()-2*a;
}
```

```
int m3(){
   return y*y-2*y;
}
```

```
int m4(){
   return p+x-y;
}
```

Call Graph Generation



Dead Field Detector (DFD)

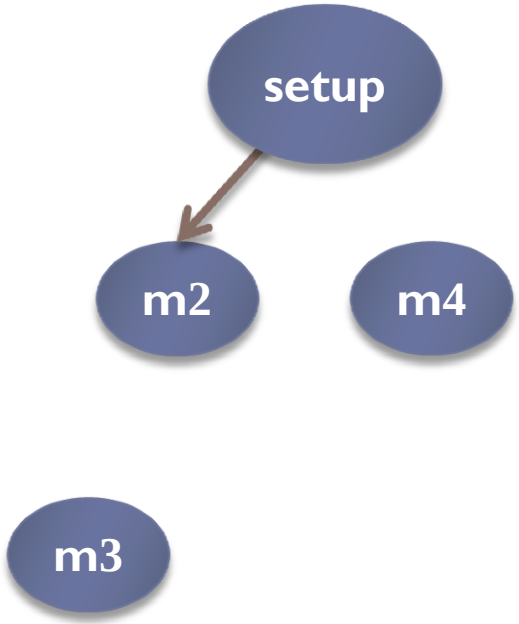
```
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
```

```
int m2(int a){
   return m3()-2*a;
}
```

```
int m3(){
   return y*y-2*y;
}
```

```
int m4(){
   return p+x-y;
}
```

Call Graph Generation



Dead Field Detector (DFD)

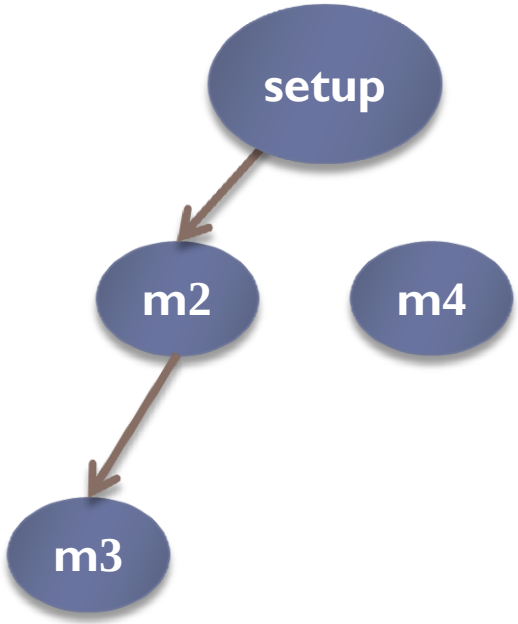
```
int setup (int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
```

```
int m2(int a){
   return m3()-2*a;
}
```

```
int m3(){
   return y*y-2*y;
}
```

```
int m4(){
   return p+x-y;
}
```

Call Graph Generation



Dead Field Detector (DFD)

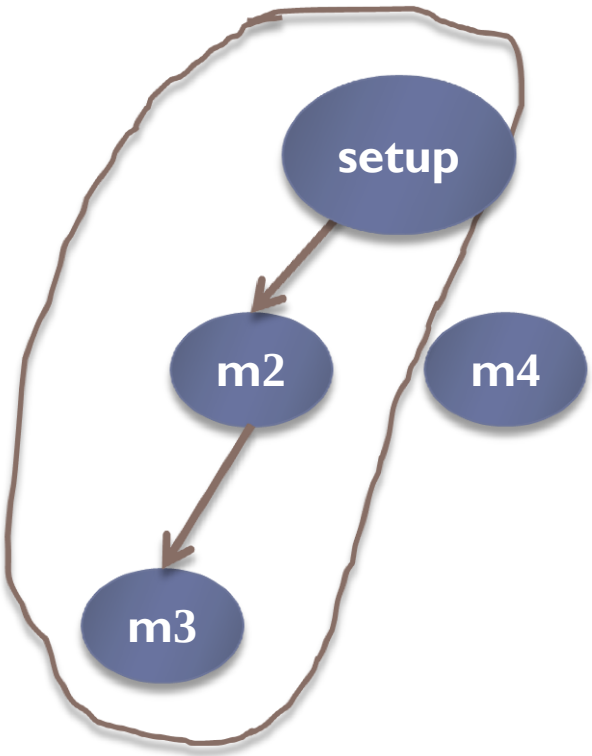
```
int setup (int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
```

```
int m2(int a){
   return m3()-2*a;
}
```

```
int m3(){
   return y*y-2*y;
}
```

```
int m4(){
   return p+x-y;
}
```

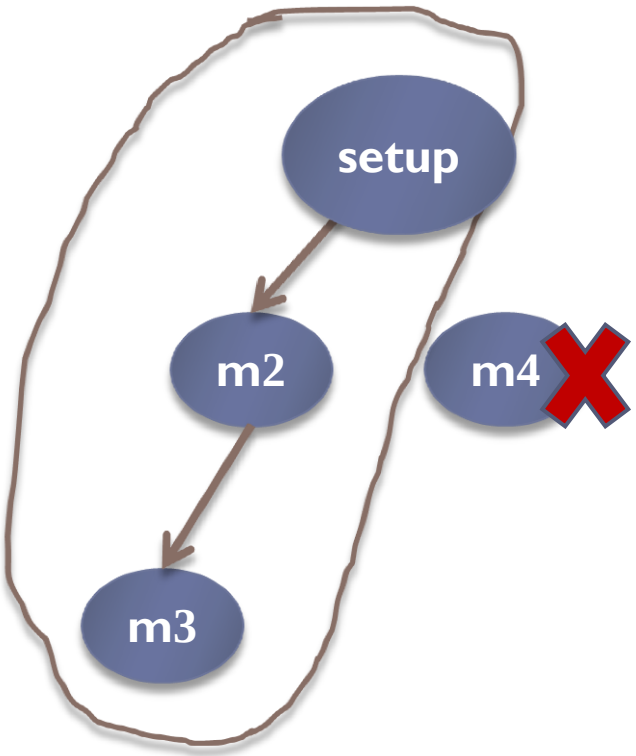
Method Call Resolution



Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20;
int z;
int setup (int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
   return y*y-2*y;
}
int m4(){
   return p+x-y;
}
```

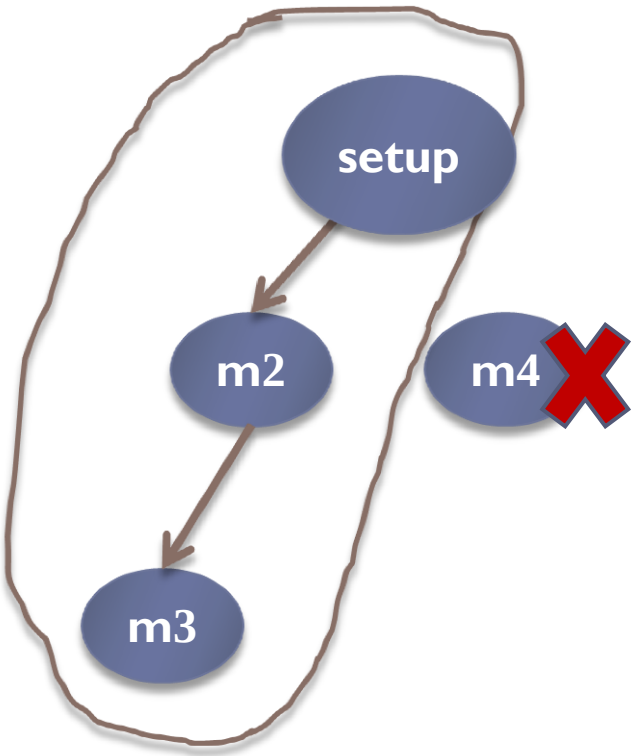
Method Call Resolution



Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20;
int z;
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
   return y*y-2*y;
}
int m4(){
   return p+x-y;
}
```

Method Call Resolution



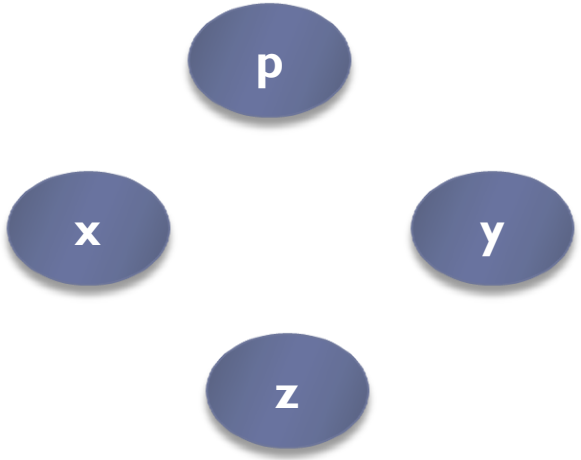
Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20;
int z;
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
   return y*y-2*y;
}
```

Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20;
int z;
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
   return y*y-2*y;
}
```

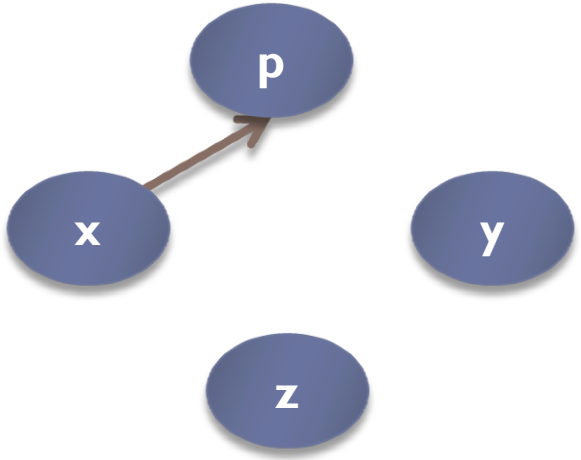
Data Dependency Graph Generation



Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20;
int z;
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
   return y*y-2*y;
}
```

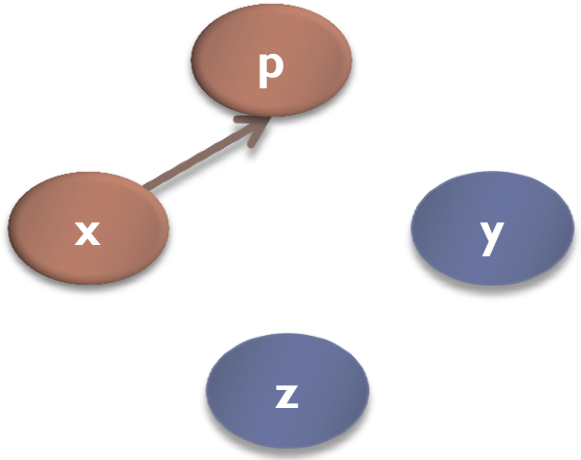
Data Dependency Graph Generation



Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20;
int z;
int setup(int a, int b){
   a=lib2.pow(a,4);
    x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
   return y*y-2*y;
}
```

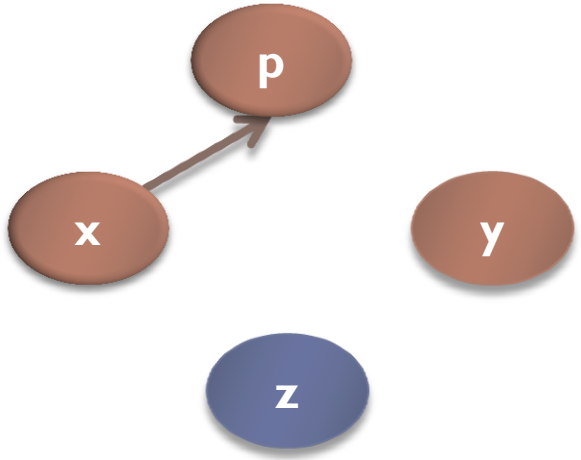
Data Dependency Resolution



Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20;
int z;
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
   return y*y-2*y;
}
```

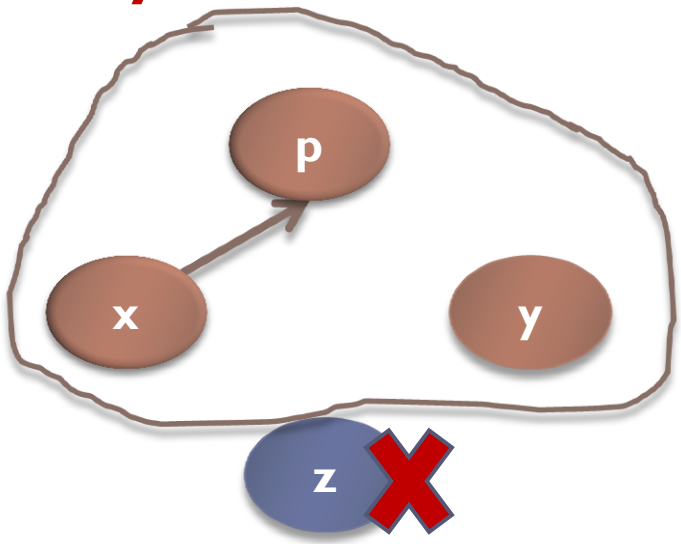
Data Dependency Resolution



Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20;
int z;
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
    return y*y-2*y;
}
```

Data Dependency Resolution

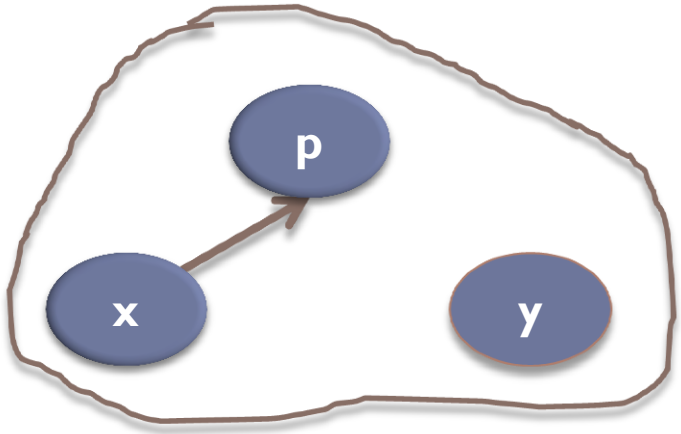


Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20;
int z;
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
    return y*y-2*y;
}
```

```
int testDummy(){
   -------
   -------
   -------
   dummy1();
}

void dummy1(){
  -------
  -------
 pow(2,x);
}
```

Data Dependency Graph

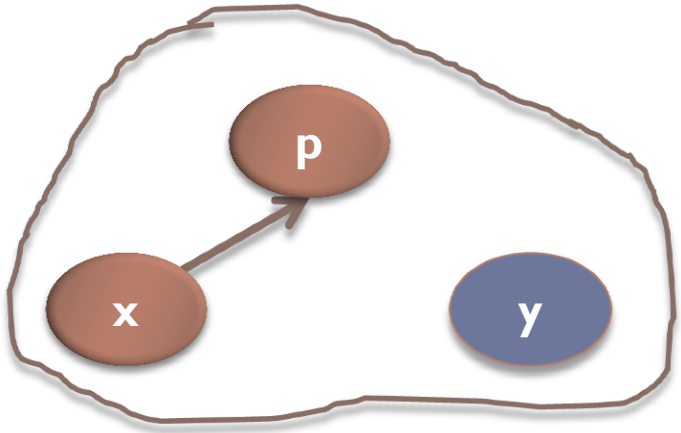


Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20;
int z;
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
    return y*y-2*y;
}
```

```
int testDummy(){
   -------
   -------
   -------
   dummy1();
}

void dummy1(){
  -------
  -------
 pow(2,x);
}
```

Data Dependency Graph



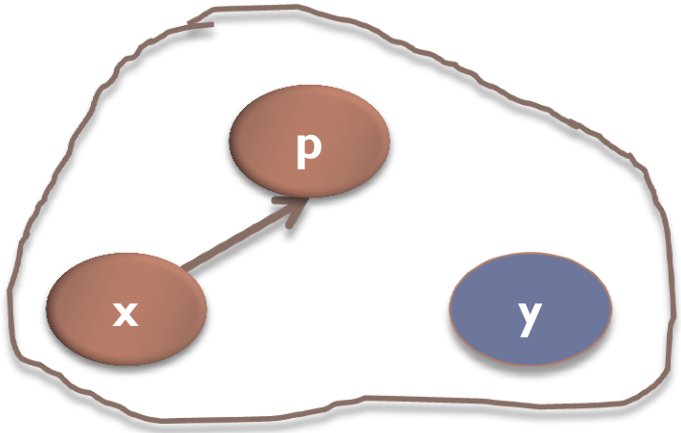
Used setup field={ x, p}

Dead Field Detector (DFD)

```
import java.lib2
import java.lib3
int p=200;
int x=p-3;
int y=20;
int z;
int setup(int a, int b){
   a=lib2.pow(a,4);
   x=x+a+b;
   return m2(x);
}
int m2(int a){
   return m3()-2*a;
}
int m3(){
    return y*y-2*y;
}
```

```
int testDummy(){
   -------
   -------
   -------
   dummy1();
}

void dummy1(){
  -------
  -------
 pow(2,x);
}
```

Data Dependency Graph



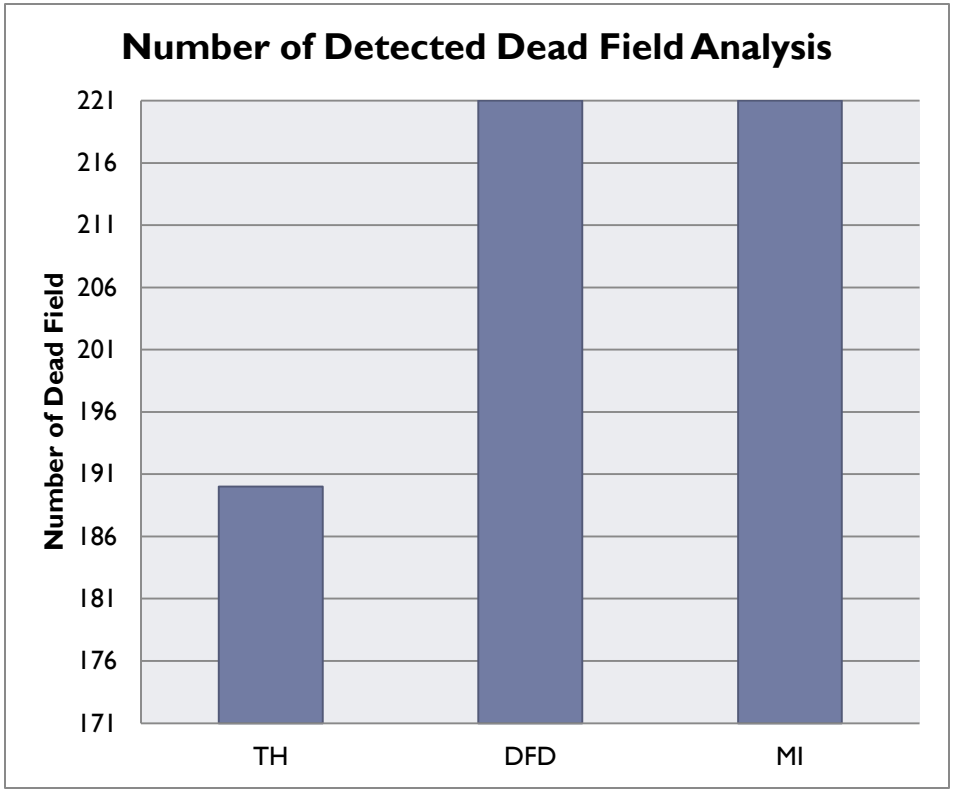
Used setup field={ x, p}

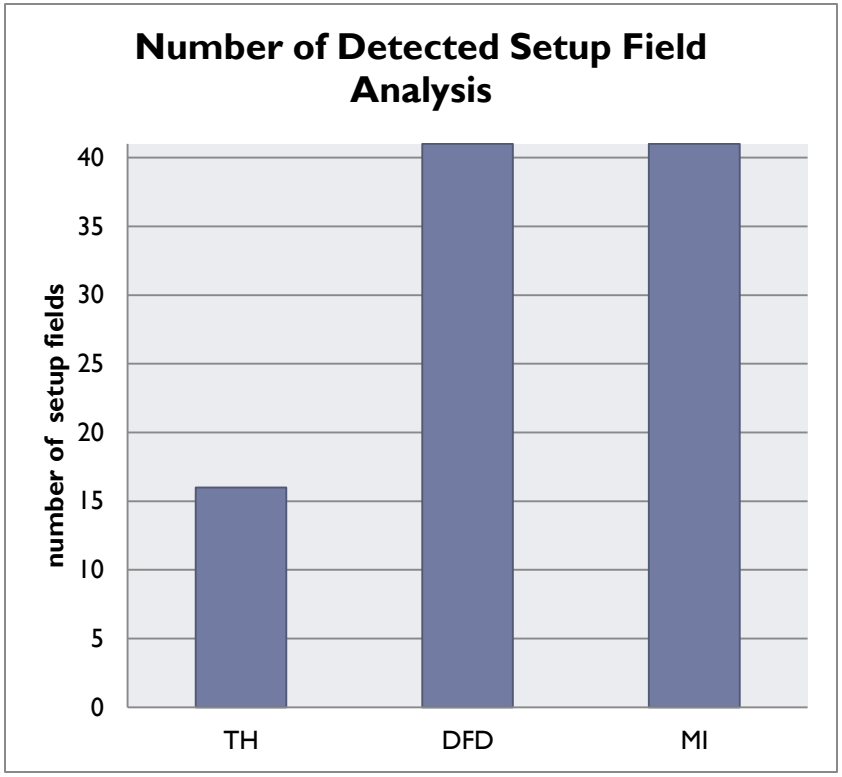
Dead Field = {x, p, y } - { x, p}
= {y}

Experimental Setup

- Implementation Details
  - Java
  - Java Parser
  - Maven
- Experimental Project
  - eGit
  - 130K lines of code, 85 test classes and 10 modules

Result Analysis





TH = TestHound
DFD = Dead Field Detector
MI = Manual Inspection

Conclusion

- DFD seems to identify more setup fields and dead fields correctly than existing approaches
- Automatic identification reduces the time and effort in software maintenance
- In future, experiments will be performed on industrial projects

Thank You

References

- [1] Martin Fowler. Refactoring: improving the design of existing code. Pearson Education India, 1999.
- [2] Arie van Deursen, Leon Moonen, Alex van den Bergh, and Gerard Kok. Refactoring test code. CWI, 2001
- [3] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and David Binkley. An empirical analysis of the distribution of unit test smells and their impact on software maintenance. In Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM), 2012, pages 56–65. IEEE, 2012.
- [4] Manuel Breugelmans and Bart Van Rompaey. Testq: Exploring structural and maintenance characteristics of unit test suites. In WASDeTT-1: 1st International Workshop on Advanced Software Development Tool and Techniques, 2008.
- [5] Stefan Reichhart, Tudor Gˆ ırba, and St ´ ephane Ducasse. Rule-based assessment of test quality. Journal of Object Technology, 6(9):231–251, 2007.
- [6] Michaela Greiler, Arie van Deursen, and Margaret Anne Storey. Automated detection of test fixture strategies and smells. In Proceedings of the Sixth International Conference on Software Testing, Verification and Validation (ICST), 2013 IEEE, pages 322–331. IEEE, 2013.