An Abstraction Mechanism for Aspect-Oriented Programming based on Test Cases

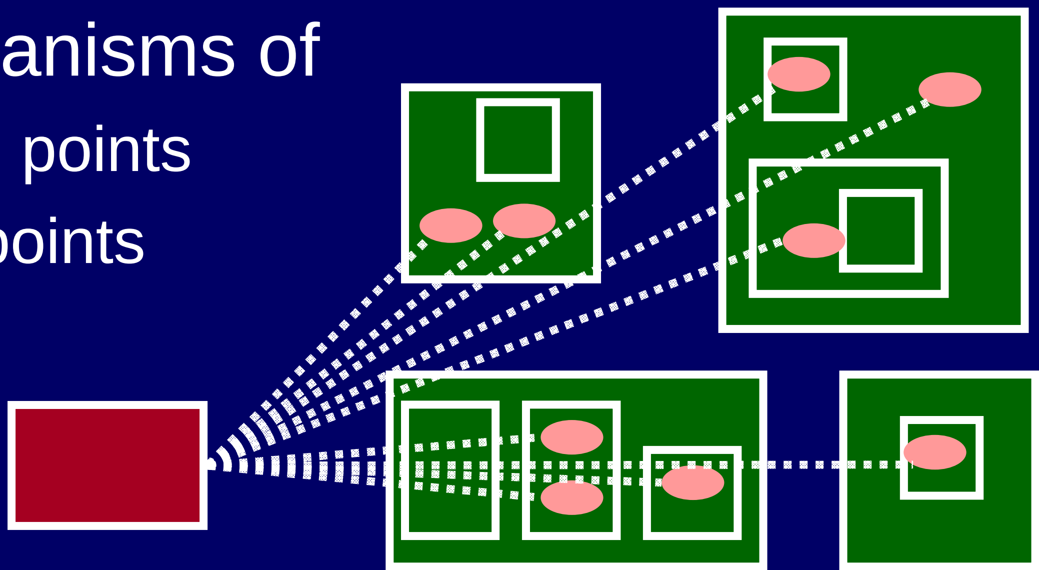
Hidehiko Masuhara (University of Tokyo)

joint work with Kouhei Sakurai

Kouhei Sakurai and Hidehiko Masuhara, "Test-Based Pointcuts for Robust and Fine-Grained Join Point Specification", In *AOSD'08*, pp.96-107, March-April 2008.

Quick introduction to AOP

- for modularizing crosscutting concerns (CCC)
  - e.g., security, logging, GUI, exception handling, …
  - don't fit hierarchical modularization
  - distribution is "difficult or likely to change"
- provides mechanisms of
  - identifying join points
  - affecting join points



State of the art of AOP

- Industrial strength implementations
  - AspectJ, Spring AOP, JBoss AOP, etc.
- Proven to be useful
  - in applying hibernating on application servers
  - in remodularizing existing code base, e.g., a commercial application server, a distributed JVM
  - in enforcing coding standards, e.g., first failure data capture (FFDC) [Colyer04]
  - as an instrumentation tool, e.g., Glassbox inspector
- Many research papers

Criticisms against AO programs

- ***“are difficult to learn"***
- ***"don't work with my favorite tools"***
- ***“are hard to understand how a program runs"***
- ***“are hard to write what I really want to do"***
- ***“are hard to be maintained"***

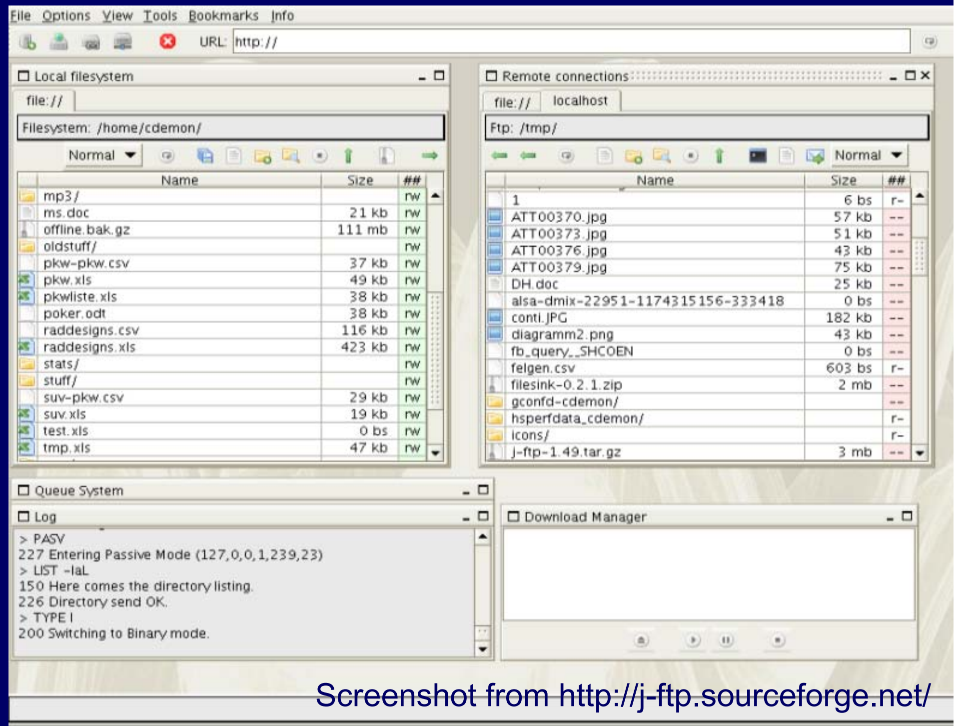
precision

fragility

- Typical reactions to new paradigms
- Do you understand OO program by following control flow?
- Challenges tackled in this talk

Example: a GUI concern in JFtp

- JFtp: an FTP client with GUI
- Updating concern: refresh the file list when it changes
  - ▸ scattered: login, chdir, put, mkdir, etc.



Screenshot from http://j-ftp.sourceforge.net/

Updating aspect

```
pointcut serverAction():
  call(doLogin(..))||call(doChdir(..))||…;
after returning(): serverAction() {
    Window.update(); }
```

- Updating concern: refresh the file list when it changes
  - scattered: login, chdir, put, mkdir, etc.

Problem of precision (1)

- Difficult to precisely specify join points

```
pointcut serverAction():
  call(doLogin(..))||call(doChdir(..))||…;
after returning(): serverAction() {
    Window.update(); }
```

- Example request:
  "update only when the list is changed"
  - i.e., don't update if failed to login
  - but do update if failed by a network failure
- Implementation (next slide):
  - check server response during doLogin

Problem of precision (2)

- "Update only when the list is changed"
- Implementation
  - allocate a flag
  - clear before doLogin
  - set when sendPass failed
  - update when not failed

```
pointcut serverAction():
  call(doLogin(..))||call(doChdir(..))||…;
after returning(): serverAction()
          && if(!failed)
{ Window.update(); }
boolean failed;
before(): serverAction() { failed=false; }
after returning(boolean success):
     call(sendPass(..)) && if(!success) {
  failed = true; }
```

… **not easy & requires detailed knowledge**

Problem of fragility

- "Fragile pointcut" problem [Stroezer05,Kellens06]
  - Aspects are easily broken when a program is changed without being aware of aspects

rename to "login(…)"

```
doLogin(…) {
  …sendPass(…)…
}
```

returns an integer


```
aspect {
  update after doLogin
  set flag if sendPass
    returned false }
```

Problem summary

- Difficult to be precise
  - ► need to keep track of state
  - ► need to know implementation detail
- Fragile to program changes
  - ► because aspects specify by using names and types
- More precise, more fragile

Our idea: example-based aspect

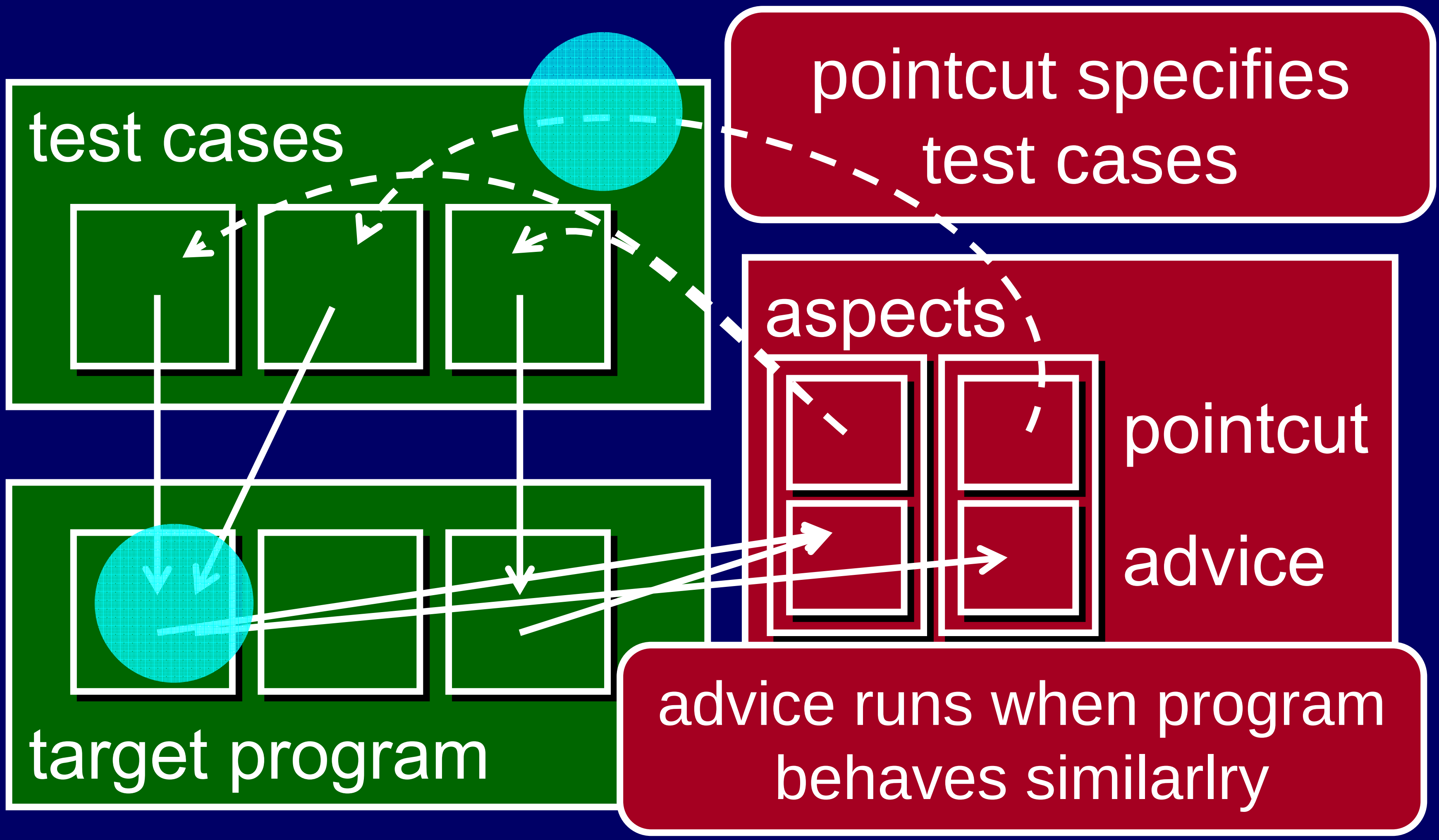
- Instead of specifying detailed behavior of the target, specify join points ***by using examples***
  - only dependent on external interfaces
  - can be more precise

after ≈⟨*goodNet.doLogin("john", "asdf876")*⟩ {
    Window.update(); }
after ≈⟨*badNet.doLogin("john", "asdf876")*⟩ {
    Window.update(); }
after ≈⟨*goodNet.doLogin("john", "0000")*⟩ { ; }

Our proposal: test-based pointcuts

- Mechanism
  - Aspect programmer specifies ***unit test cases*** in a pointcut
  - Compiler/runtime selects join points ***with similar execution history***
- More precise:
  - distinguishing different control flow
- Less fragile:
  - directly dependent on only public interface
  - automated testing for compatibility check

Test-based pointcuts: overview



Specifying test cases

- Assumptions/requirements:
  - One execution per test case
  - Shared fixture variables for test parameters (no immediate values)
  - Explicitly declared init./exec./verif. phases
- Syntax: test(***pct***)
  e.g., test(get(Fixtures.invalidUser))
- Semantics: “any join point that behaves similar to one of the test cases matching ***pct***”

Specifying test cases: example

```
testLoginSuccess() {
    Server s = ...;
    testBody();
    r = s.doLogin(F.validUser, F.validPass);
    testCheck();
    assertTrue(r);
}
```

phase separator

F
Str validUser
Str vaildPass
Str invalidUser
Str invalidPass

fixtures

after(): **test(get(F.validUser))**
&& test(get(F.validPass)) {
Window.update(); }
after(): **test(get(F.invalidUser))**
&& test(get(F.invalidPass)) ...

test pointcuts can identify crosscutting test cases

Similarly check: what are similar?

"any join point that behaves ***similar*** to one of the test cases matching the pointcut"

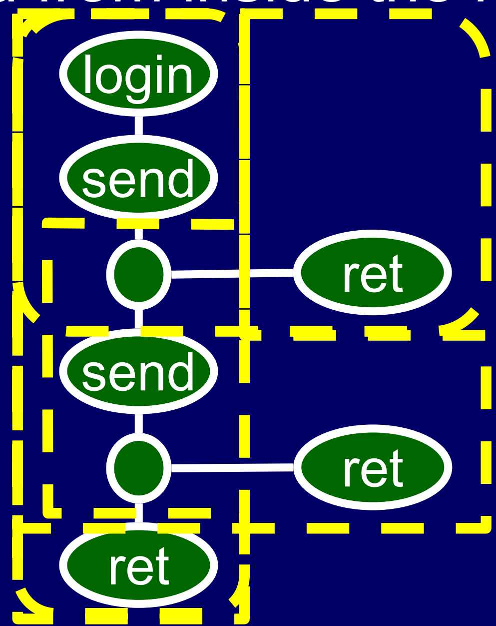
- Our definition: two executions are similar when they reached the same set of nodes in a CFG
- Many possibilities for improvement (will discuss)

```
boolean login(user,pass) {
  send("USER", user);
  if (!check("OK")) return false;
  send("PASS", pass);
  if (!check("OK")) return false;
  return true; }
```

Similarly check: our approach

- Def. Two execution histories are similar when the sets of visited CFG nodes are the same
  - bounded by particular method execution
  - including methods called from inside the method

```
boolean login(user,pass) {
  send("USER", user);
  if (!check("OK")) return false;
  send("PASS", pass);
  if (!check("OK")) return false;
  return true; }
```



Implementation

- Prototype compiler is implemented
  - ▸ 2.5KLoC extension to abc
- 2-Phase compilation
  1. run all test cases with profiling aspects
  2. run instrumented target program
     - create a flag set at entry
     - flag at each shadow
     - test the flag set at exit

Phase 1: test execution

- Instrument target program (w/o aspects) and test cases
- Run each test case to know
  - if it matches any of the test pointcuts
  - name of tested method
  - set of visited CFG nodes

```
boolean login(user,pass) {
  send("USER", user);
  if (!check("OK")) return false;
  send("PASS", pass);
  if (!check("OK")) return false;
  return true; }
```

```
testLoginSuccess() {
   Server s = ...;
   testBody();
   r = s.doLogin(
   F.validUser, F.validPass);
   testCheck();
   assertTrue(r);
}
```

Phase 2: aspect weaving with instrumentation

For each test case selected by a test pointcut,

- Instrument tested method to
  - allocate a set of flags
  - set a flag at each branch
- Insert advice invocation with a condition

```
boolean login(user,pass) {
  send("USER", user);
  if (!check("OK"))
  { r=false; goto L;}
  send("PASS", pass);
  if (!check("OK"))
  { r=false; goto L;}
  r=true;
  L:<exec after advice>
  return r; }
```

s=0;

s|=1b

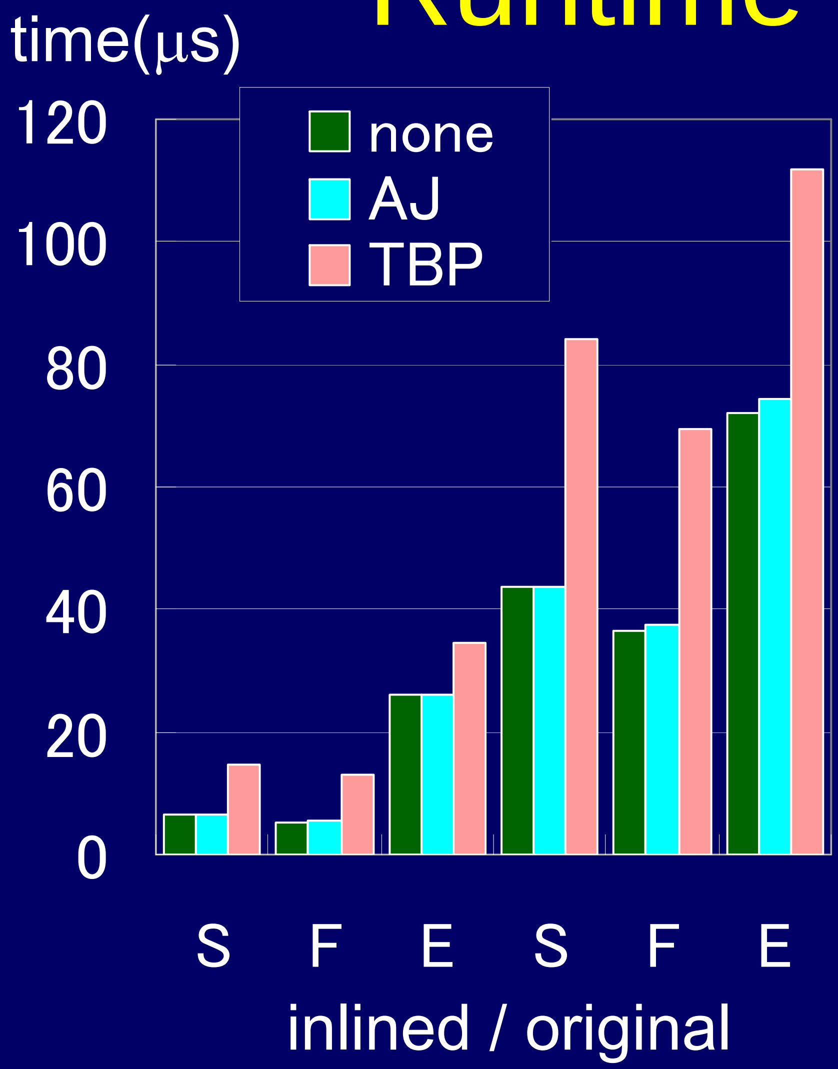
s|=10b

if(s==1010b)

Evaluation

- Runtime performance
  - How slow it is?
- Evolution test
  - Do test-based pointcut work after evolution?

Runtime performance

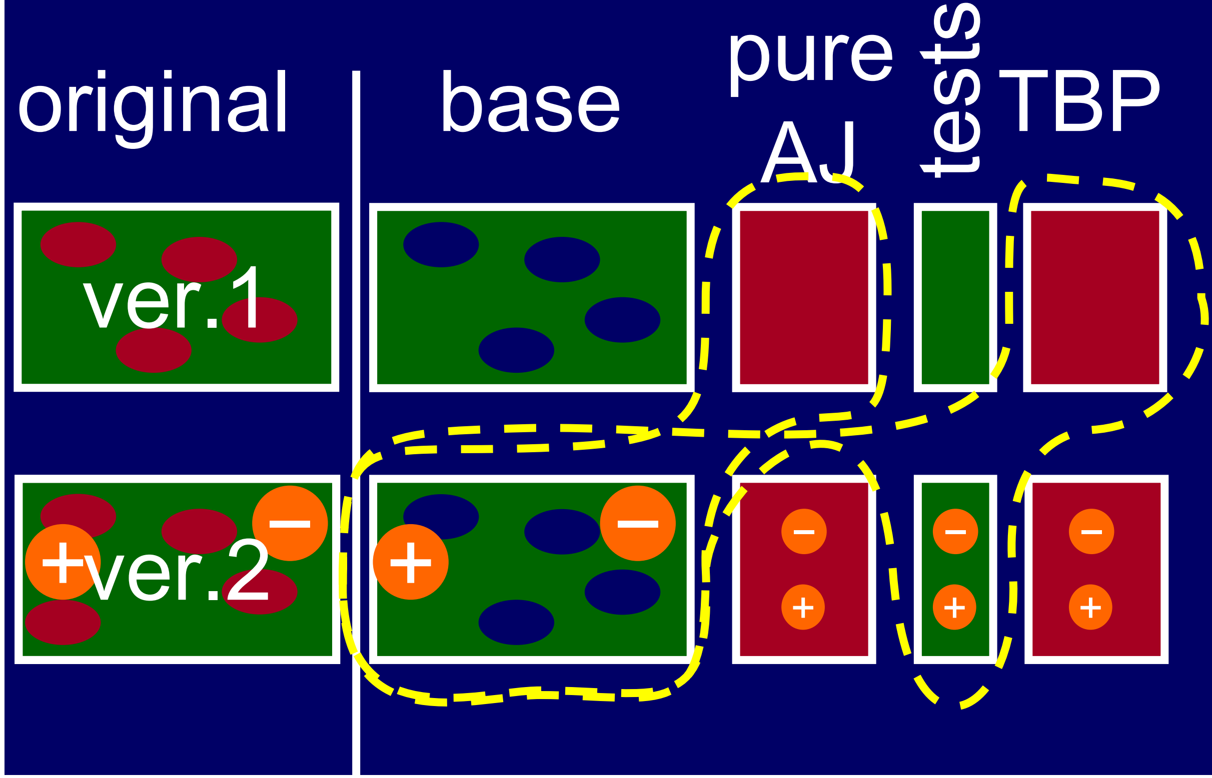


- Compared execution time of doLogin in JFtp
  - inlined version (shown)
  - original version
- TBP is 1.3-2.4 times slower than AspectJ version
  - comparison of kernel method

Intel Core Duo 2.16 GHz with 2 MB L2 cache memory, 2 GB RAM, a HotSpot JVM version 1.5.0 07 on Mac OS X 10.4.10.

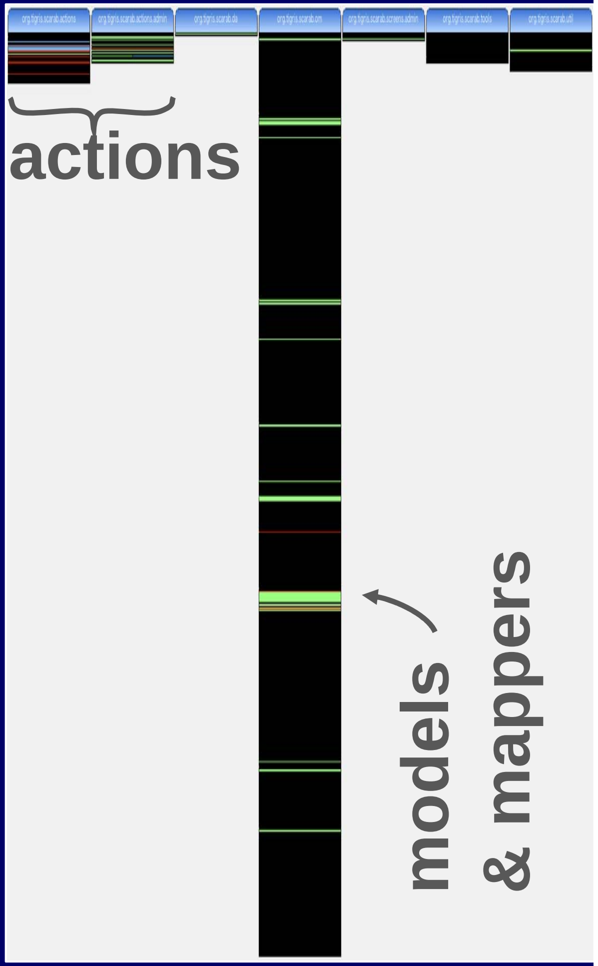
Evolution test

- We retroactively changed an open source software
- Then counted and compared the number of changes to get aspects right



Evolution Test: Target
Scarab Issue Tracking System

- 100KLoC, 530 files in Java
- Refactored 3 CCCs
  - ▸ Notification (19)
  - ▸ Security (13)
  - ▸ Caching (84)
- Tested with 4 major versions



Evolution test: Results in AspectJ

- # of changed pointcuts in AspectJ classified by the causes: MoVed decl, Signature Change, ReMoval or ADded decl.

	b16 → b19						b19 → b20						b20 → b21					
			RM		AD				RM		AD				RM		AD	
	MV	SC	DECL	JP	DECL	JP	MV	SC	DECL	JP	DECL	JP	MV	SC	DECL	JP	DECL	JP
notif.		1				1									1		1	
security											1			1	1	1	4	1
cache	5		1		6			1			1	1	2	1				

Evolution test: Results with Test-Based Pointcuts

- No change
- Change by new fixture var.
- Change in concern
- Unusable TBP

Confirmed cases when TBP can automatically follow changes

	b16 → b19						b19 → b20						b20 → b21					
			RM		AD				RM		AD				RM		AD	
	MV	SC	DECL	JP	DECL	JP	MV	SC	DECL	JP	DECL	JP	MV	SC	DECL	JP	DECL	JP
notif.		1				1									1		1	
security											1			1	1	1	4	1
cache	5		1		6			1			1	1	2	1				

Discussion: distinguishing execution history

- Pure AspectJ: need to keep track of behavior
- Extensions: Tracecut[Walker04], stateful aspects[Douence04], Tracematch[Allan05]
  - with state machine / regular expression
  - yet dependent on the details
- Test-based pointcuts offer indirect means of specifying execution history

Discussion: pointcut maintenance

- Tool supports: complementary
  - ► IDE to mark advised locations / AO-aware refactoring / pointcut delta[Stoerzer05]
- Model-based pointcut[Kellens06]
  - ► specify join points with names in the model
  - ► map the model names to implementation

  —someone has to maintain the map?
- Test-based pointcut: someone has to maintain test cases

Discussion: similarity

Alternatives:

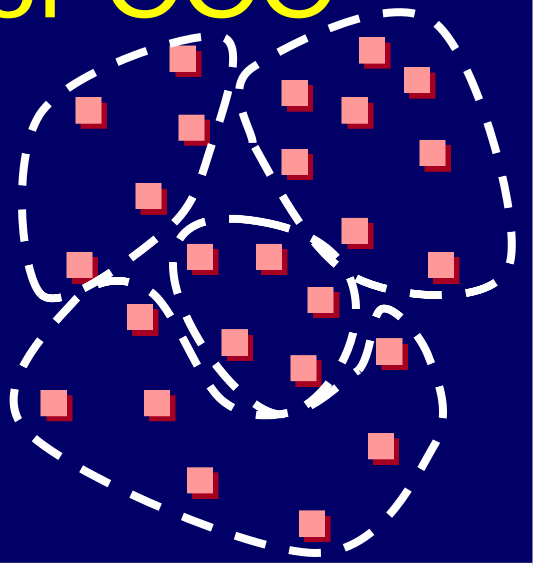
- Sequence equality
- Set equality over CFG nodes
- Identification of key braches

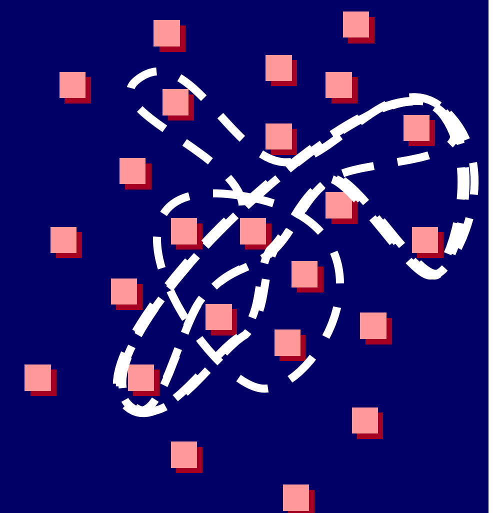
Issues:

- Efficient impl.
- Coverage
- Sensitivity to # of iterations & execution order
- Values
- Future execution

Abstraction mechanism for CCC

- A traditional module abstracts changes inside the module
- An aspect abstracts changes "where to affect"
  - existing pointcuts specify ***names*** of affected elements
  - TBP specify ***examples*** of affected behaviors
    — more abstracted?





Conclusion

- Proposed test-based pointcut
  - ▸ can distinguish different execution histories
  - ▸ relies on unit test cases:
    - automated testing
    - only dependent on public interface
- An implementation based on sets of CFG nodes
- Not too slow (overhead factor 1.3-2.4)
- Usable, sometimes more robust