24th IEEE International Conference on Software Maintenance
ICSM 2008
★ Beijing, China — September 28 to October 4, 2008
http://www.icsm2008.org/

Traceability for Maintenance of Secure Software Systems

[1]Yijun Yu
[1]Jan Jurjens
[2]John Mylopoulos

The Open University
UNIVERSITY OF TORONTO

Requirements for Traceability in …

- Requirements traceability is the ability to follow requirements in the development lifecycle (Gotel et al, RE'94)
- For secure systems, right decisions must depend on *high quality* traceability links:
  - precision ~ 100%
  - recall ~ 100%
- Otherwise, vulnerability may result from malicious adversaries
  - Especially important for self-protection

SSL handshake protocol

- SSL is widely used in SSH, HTTPS, etc.
- It is one of mandatory internet secure protocols (Antoniol ICSM'08)
- The handshake protocol has been implemented before its design were verified (Jurjens, ASE'06)
- Therefore one cannot enforce traceability through model-driven development

Man-in-the-Middle Attack

$$C \xrightarrow{N_i::K_C::Sign_{K_C^{-1}}(C::K_C)} A \xrightarrow{N_i::K_A::Sign_{K_A^{-1}}(C::K_A)} S$$

$$A \xleftarrow{\{Sign_{K_S^{-1}}(K_j::N_i)\}_{K_A}::Sign_{K_{CA}^{-1}}(S::K_S)} S$$

$$C \xleftarrow{\{Sign_{K_S^{-1}}(K_j::N_i)\}_{K_C}::Sign_{K_{CA}^{-1}}(S::K_S)} A$$

$$C \xrightarrow{\{s\}_{K_j}} A \xrightarrow{\{s\}_{K_j}} S$$

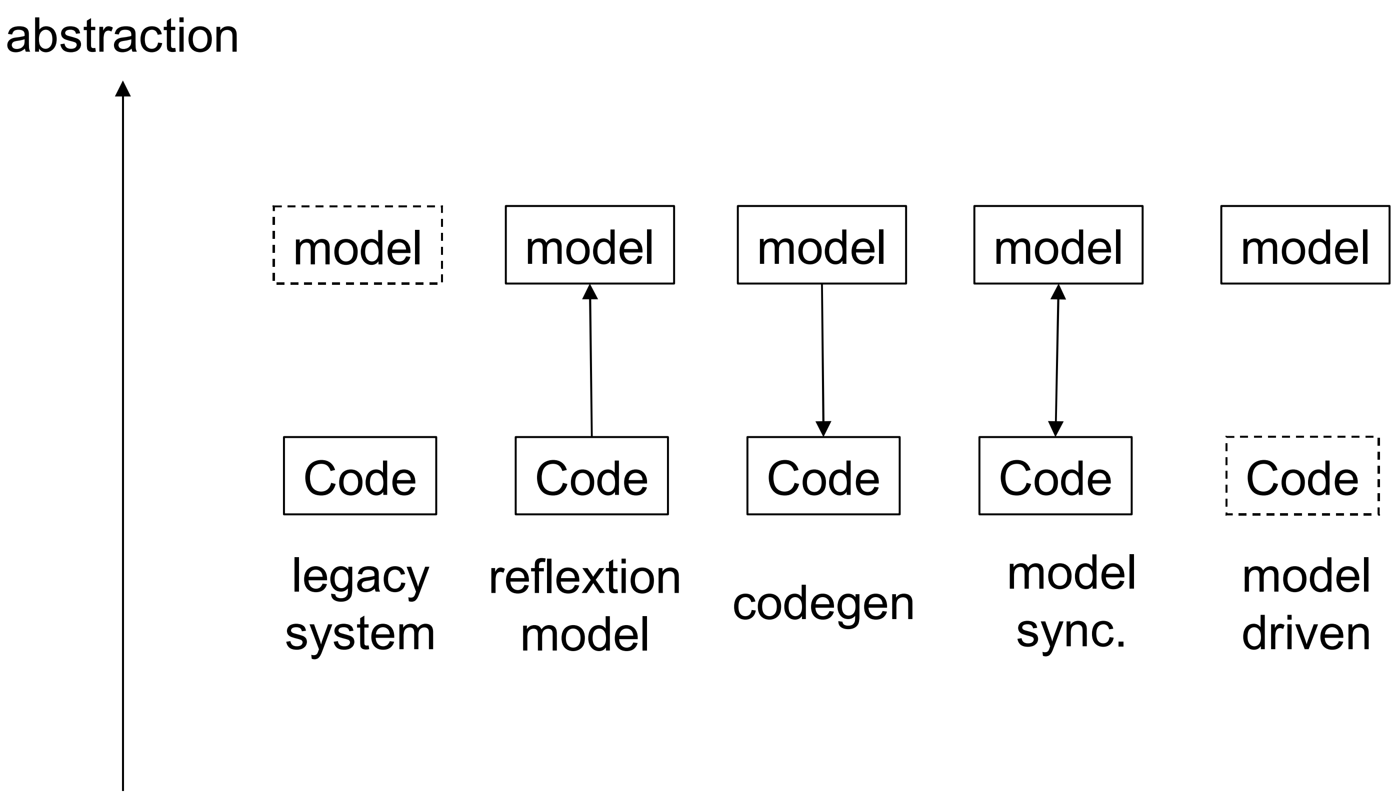
Research questions related to high-quality traceability links?

- For general software maintenance, traceability links are helpful even when not 100% precise (De Lucia et al, TOSEM’07, ASE’08), (Antoniol, et al. ICSM’08)
- What shall we do if high quality traceability links are required in secure software domains?
  - Is it possible to *reuse* the analysis results by using traceability links?
  - Is it possible to *obtain* such traceability links?
  - Is it possible to *maintain* such traceability links for evolving software systems?

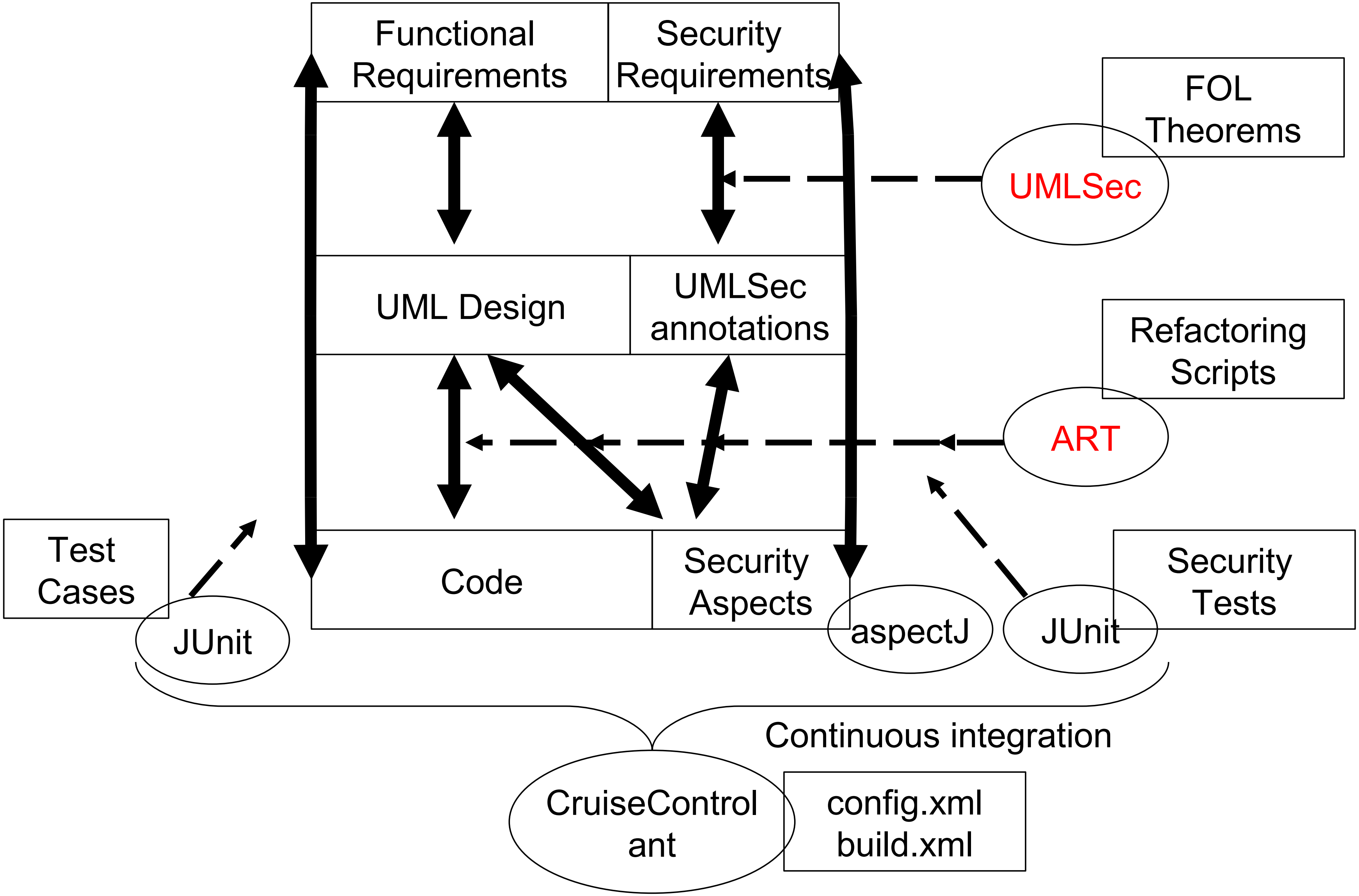
Outline of the talk

- A traceability framework in support of software maintenance lifecycle
  - Security requirements
    - Non-functional requirements
  - Security design
    - UML sec proof obligations
  - Security implementation
    - refactoring for traceability
  - Security vulnerability detection
    - Targeted testing for missing traceability links
  - Security hardening
    - Security aspect
- A case study: Jessie

Models versus Code



Adapted from B. Selic's keynote ASE'07


Functional Requirements
Security Requirements
FOL Theorems
UMLSec
UML Design
UMLSec annotations
Refactoring Scripts
ART
Test Cases
Code
Security Aspects
Security Tests
JUnit
aspectJ
JUnit
Continuous integration
CruiseControl ant
config.xml build.xml

Security requirements

- Functional requirements versus non-functional requirements
  - non-functional $\neq$ non-mandatory
  - satisficing: good-enough solution (H. Simon, Sci. of Artificial 1997)
- Security requirements are NFRs
  - In principle, never fully achievable
  - Practically, it should be fully achieved given the current state-of-the-art

UMLsec

(Jurjens, 2005)

- UML are used for OOD
- Security requirements are encoded as UMLsec stereotypes in representation
- FOL formula are generated from UMLsec
- Automated theorem provers are then used to verify the satisfaction of the formula

Security Analysis in First-order Logic

Approximate adversary knowledge set from above:

Predicate *knows(E)* meaning that adversary may get to know *E* during the execution of the system.

E.g. secrecy requirement:
For any secret *s*, check whether can derive *knows(s)* from model-generated formulas using automatic theorem prover.

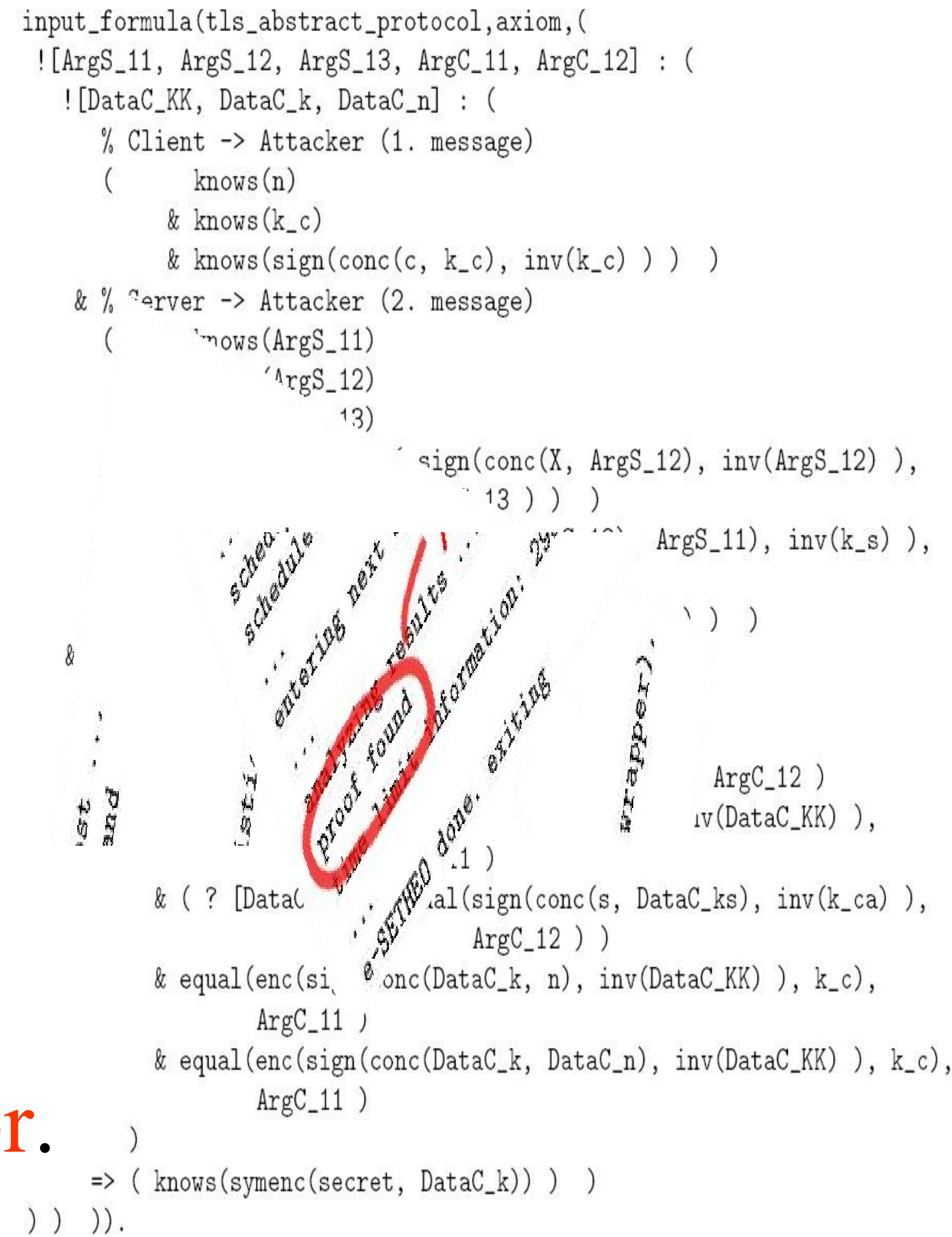
Analysis

Check whether can derive *knows(s)* e.g. using e-Setheo.

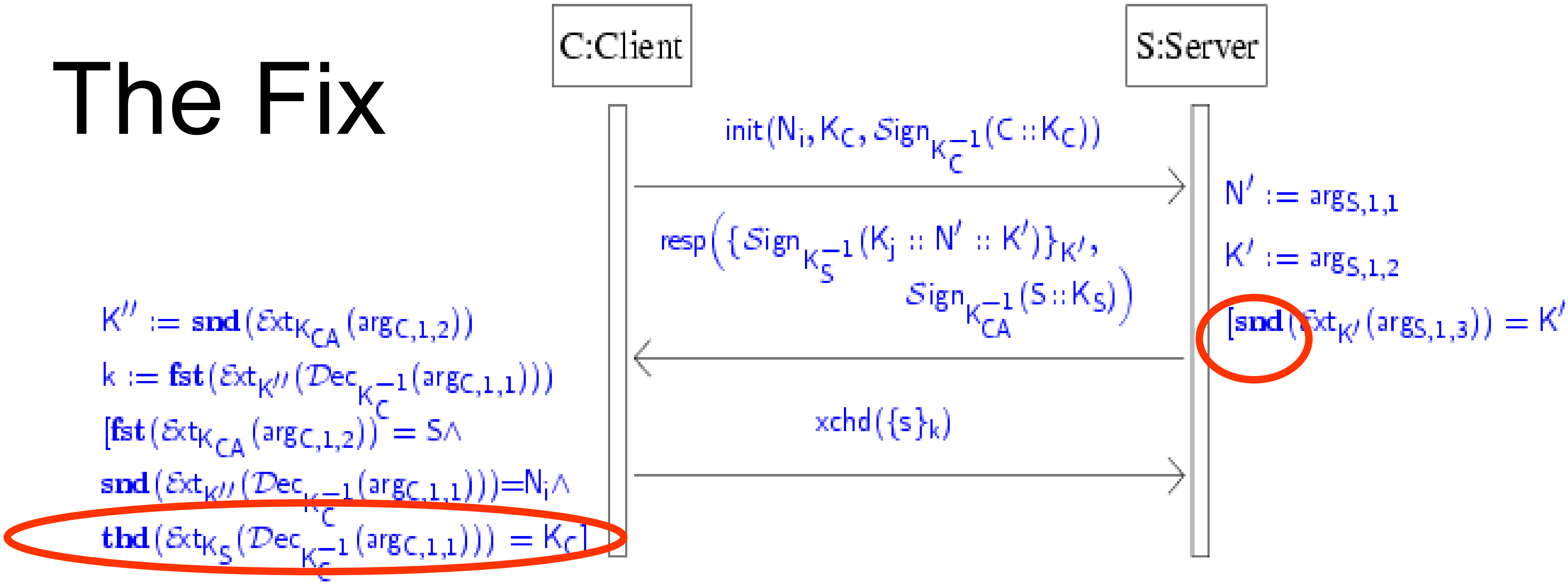
Surprise: Yes !

➔ Protocol does not preserve secrecy of *s*.

Why ? Use Prolog-based attack generator.



The Fix



e-Setheo: Proof that *knows(s)* not derivable.

Note completeness of FOL (but also undecidability).

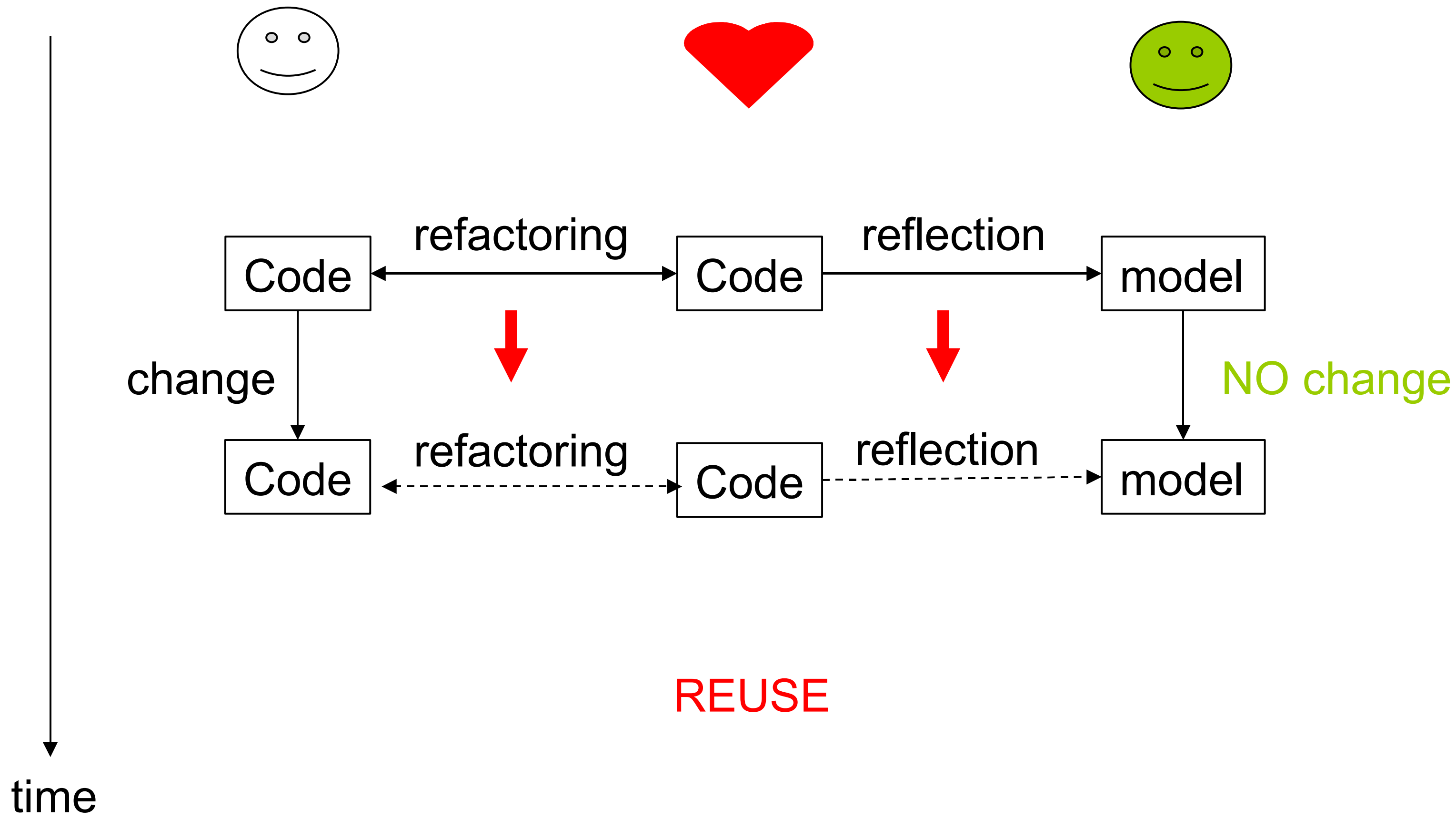
Implementation correctness

- After verifying UML models, are we secure?
  - Code may wrongly implement the model
  - Correct code may evolve into wrong implementations
- Then we need to verify the changed code against the design model. But that’s costly !
  - We provide refactoring-based tool support
  - Reuse the analysis on changed code base
- Regressive model-based security engineering
  - Continuous integration

Why refactoring?

- By definition (Fowler, Refactoring 1999):
  – Refactoring preserves program behaviours
  – Refactoring changes internal structure, and this may lead to better abstraction
- We need to bridge model with code, in two steps
  – We make code more abstract
  – We trace abstract code to the design, which reduce the number of traceability links

Reusable traceability links



Refactoring for Traceability


```
/* $workspace/abc/src/abc.java */
public class abc {
  public static void main2(String args[]) {
    System.out.println("Hello");
    System.out.println("Hello");
  }
}
---- Step 1. rename.type ----
/* $workspace/abc/src/hello.java */
public class hello { … }
---- Step 2. extract.method ----
public static void main2(String args[]) {
   print_hello();
   print_hello();
}
public static void print_hello() {
   System.out.println("Hello");
}
---- Step 3. extract.temp ----
String string = "Hello";
System.out.println(string);
---- Step 4. promote.temp ----
public static String message="Hello";
public static void print_hello() {
    System.out.println(message);
}
---- Step 5. rename.method ----
public class hello {
  public static String message="Hello";
  public static void main(String args[]) {
   print_hello();
   print_hello();
  }
  public static void print_hello() {
   System.out.println(string);
  }
}
```

Difference from use of refactoring in software maintenance

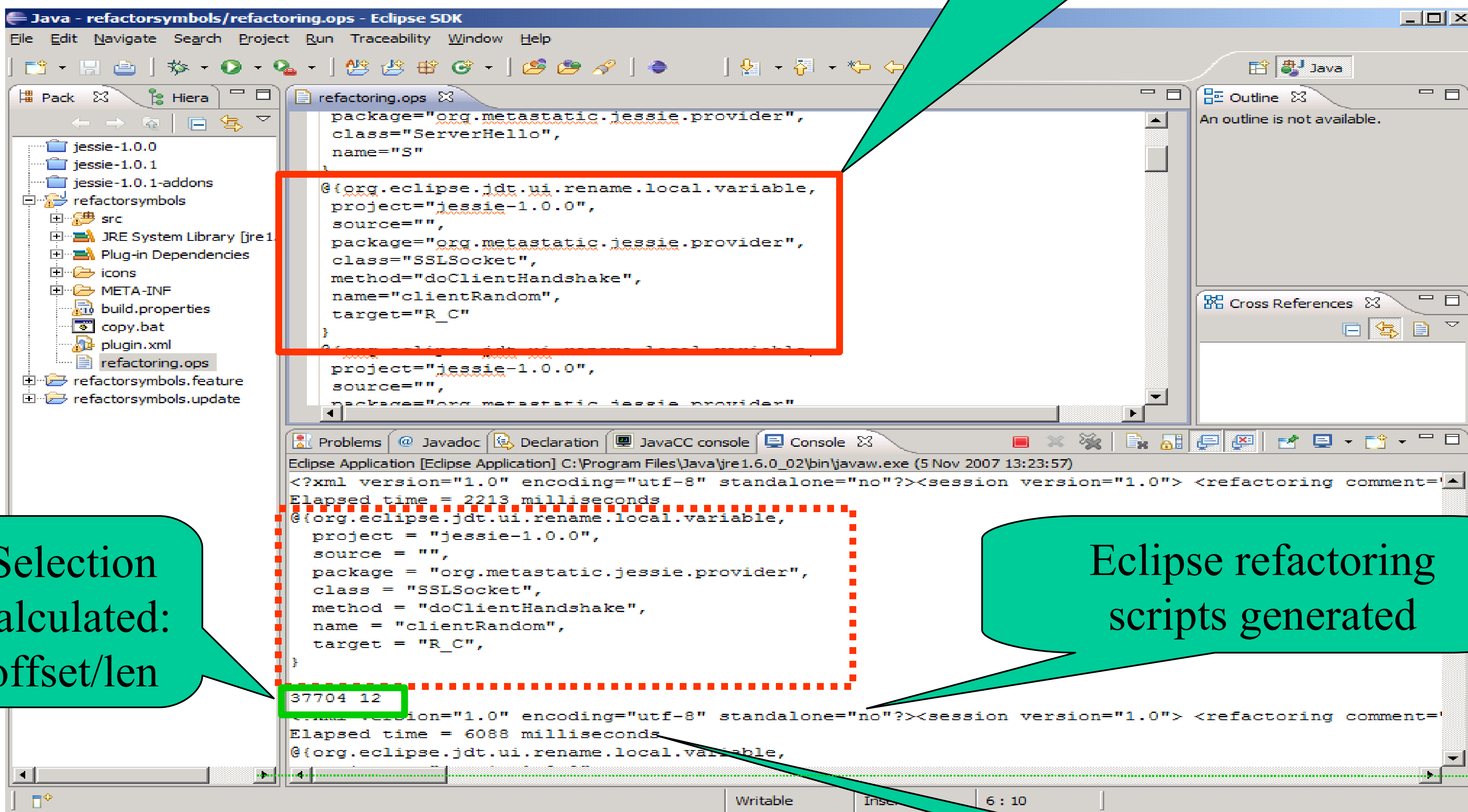
- However, we do not intend to modify the source code in the repository
  - Ownership: programmers are familiar with their own code, analysts may not have permission to commit changes
  - Refactoring is solely for analysis purposes, one design perspective at a time
- Two remaining problems
  - How to apply refactoring to changing code bases?
  - How to reuse the analysis results for implementations of similar design?

Our current solutions and results

- Automated refactoring tool
  http://computing-research.open.ac.uk/repos/art/trunk
  - Context generalisation: parameterise the refactoring scripts to allow for ranges of changes
  - Context specialisation: obtain concrete refactoring scripts for IDE, e.g., Eclipse
- Security aspects
  - Fix requires a change in behaviour, which is beyond refactoring
- Continuous integration
  - Converge changes in design, code, and security analysis

Refactoring scripts

Change-proof specification of refactoring operations



Selection calculated: offset/len

Eclipse refactoring scripts generated

Executed

Jessie


SSLSocket.java
{
msg = Handshake.read(din, certType);
if (DEBUG_HANDSHAKE_LAYER) debug.println(msg);
if (msg.getType() != Handshake.Type.CERTIFICATE)
{
throwUnexpectedMessage();
}
Certificate serverCertificate = (Certificate) msg.
checkCertificate(suite, peerCerts);
serverKey = peerCerts[0].getPublicKey();
serverKex = serverKey;
}
hello.java
SSLSocket.java
if (msg.getType() != Handshake.Type.CERTIFICATE
{
throwUnexpectedMessage();
}
Certificate serverCertificate = (Certificate) m
X509Certificate[] peerCerts = serverCertificate
try
{
session.trustManager.checkServerTrusted(pee
sui
if (suite.getSignature() == "RSA" &&
!(peerCerts[0].getPublicKey() instanceo
throw new InvalidKeyException("improper p
if (suite.getKeyExchange() == "DH" &&
!(peerCerts[0].getPublicKey() instanceo
throw new InvalidKeyException("improper p
if (suite.getKeyExchange() == "DHE")
{
if (suite.getSignature() == "RSA" &&
!(peerCerts[0].getPublicKey() insta
throw new InvalidKeyException("improp
if (suite.getSignature() == "DSS" &&
!(peerCerts[0].getPublicKey() insta
@{org.eclipse.jdt.ui.extract.method,
project="jessie-1.0.0",
source="",
package="org.metastatic.jessie.provider",
class="SSLSocket",
method="doClientHandshake",
begin=" try
{
session.trustManager.check
",
end=" serverKey = peerCerts[0]
serverKex = serverKey;
",

Traceability becomes reusable

- JESSIE and JSSE are two open-source implementations of SSL in Java
- JESSIE 1.0.1 fully replays traceability refactoring operations on 1.0.0
- Most refactoring operations are reusable in JSSE 1.6

Messages in sequence	op.	diff	Time (ms)
S1: $C \rightarrow S : (P_{\mathrm{ver}}, R_C, S_{\mathrm{id}}, \mathrm{Ciph}[\,], \mathrm{Comp}[\,])$	7	31	13,891
S2. $S \rightarrow C : (P_{\mathrm{ver}}, R_S, S_{\mathrm{id}}, \mathrm{Ciph}[\,], \mathrm{Comp}[\,])$	5	20	9,437
S3. $S \rightarrow C$: Certificate[X509Cert$_s$]	2	2	1,474
S4. C : Veri(X509Cert$_s$)	2	2	3,854
...	...	...	...
Total of 7 messages and 3 checks	27	86	40,303

Messages	JESSIE 1.0.1		JESSIE 1.0.0		JSSE 1.6	
	op.	diff	op.	diff	op.	diff
S1	7	31	7	33	5	23
S2	5	20	5	21	4	16
S3	2	2	2	2	2	2
S4	2	2	2	2	2	2
...	...	...	...	...	...	...
Total	27	86	27	89	21	68

Vulnerability detected

- According to the UMLsec protocol, a certificate with (issuedDate, expiredDate) should be checked whenever a message is received
- 3 call sites of getCertificate were found in the refactored code
- Only 2 of them call the checkCertificate function
- The missing call site can be listed by a *pointcut* expression in aspectJ: getCertificate()
- Test cases are then constructed to reveal the vulnerability
- A fix of the vulnerability was done by an aspect, with an *after()* advice to call checkCertificate

Conclusion and future work

- Security engineering requires high-quality traceability links
- A refactoring-based traceability link maintenance technique was proposed
- Effectiveness was evaluated on an important security protocol

- Future work: scalability, full automation, more case studies … collaborations!