# Model-based Security Engineering for Evolving Systems

Jan Jürjens

Open University (GB)

Microsoft Research (Cambridge)

Robinson College (Univ. Cambridge)

http://www.jurjens.de/jan

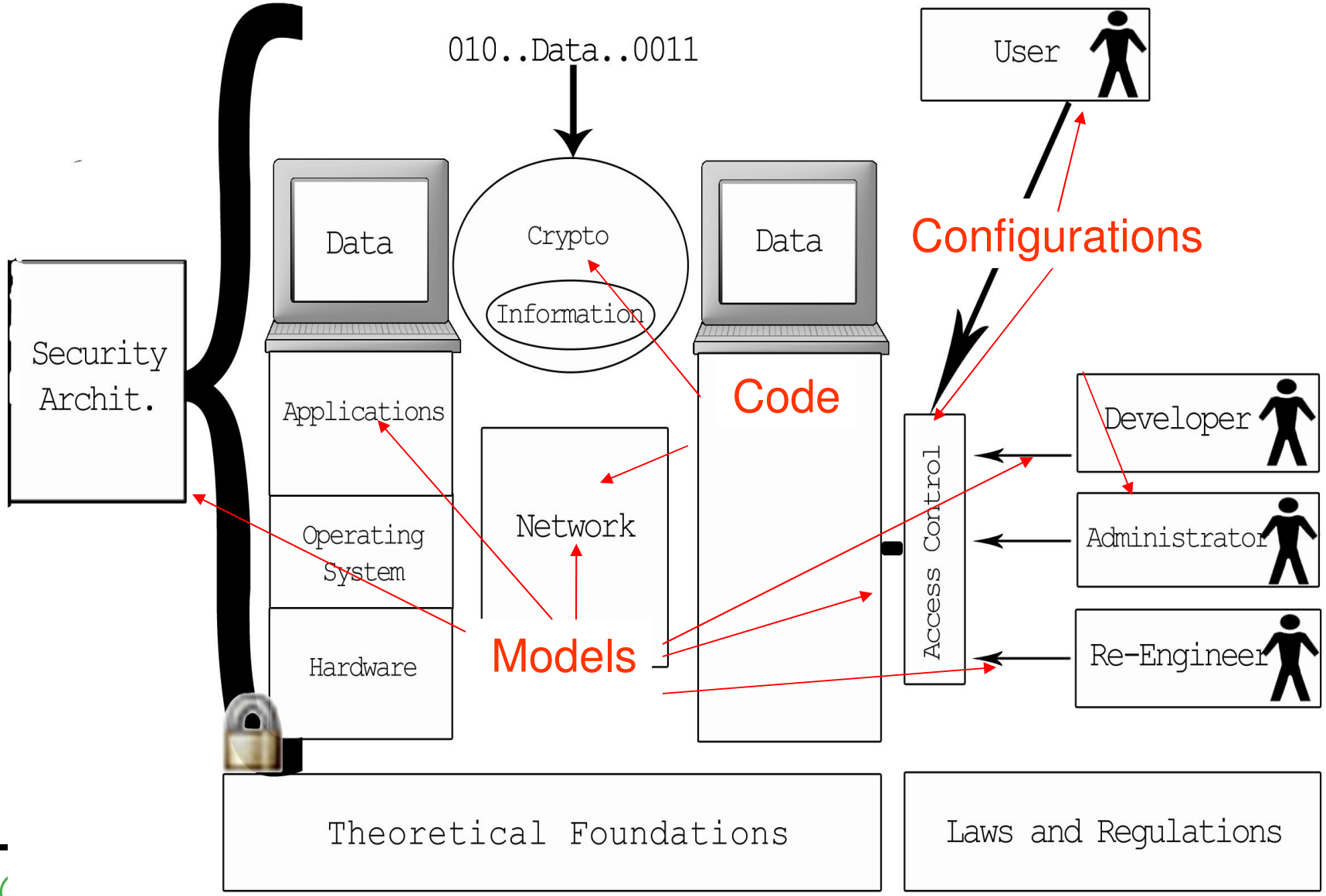
# Security: Some Problems

„Blind“ use of security mechanisms:

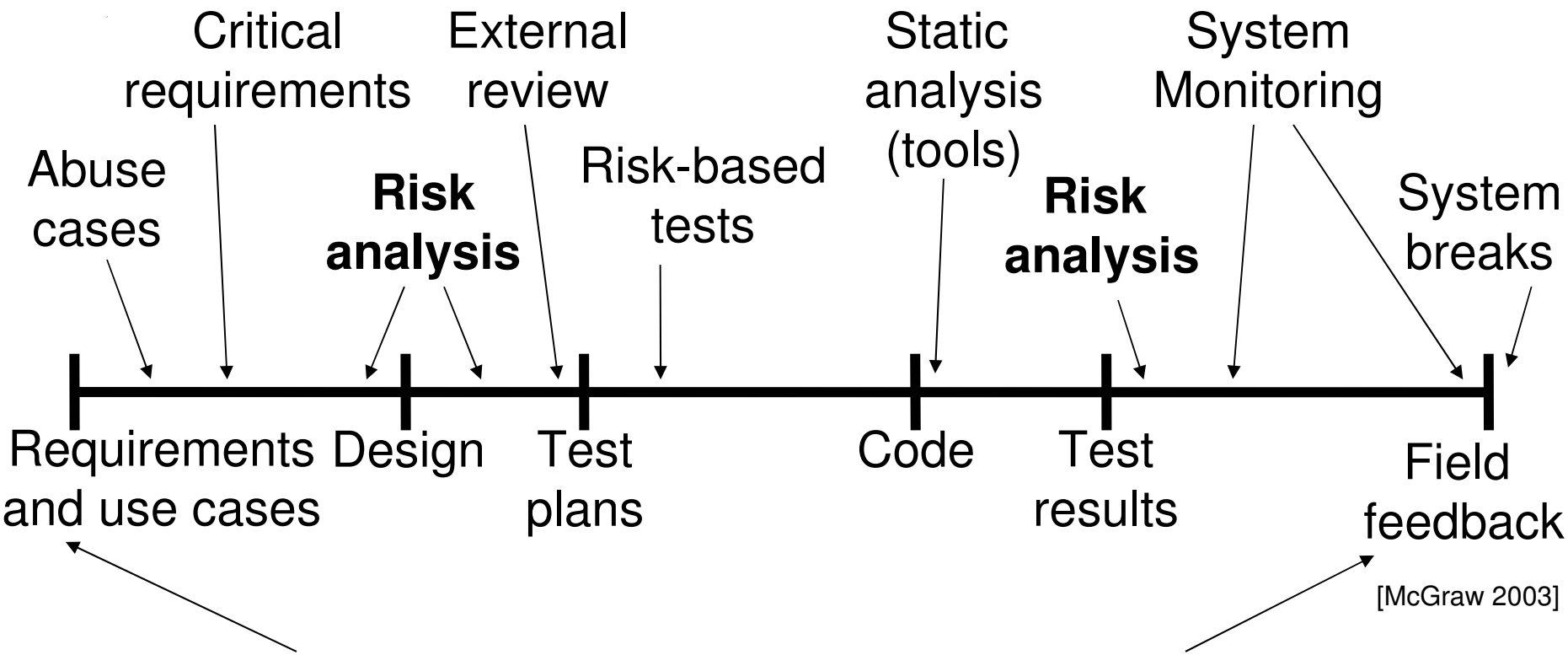
- Security usually compromised by circumventing (rather than breaking) them.
- Assumptions on system context, physical environment.
- Attacker may use unintended/unnoticed functionality

„Those who think that their problem can be solved by simply applying cryptography don`t understand cryptography and don`t understand their problem“ (R. Needham).

# Architectural Layers



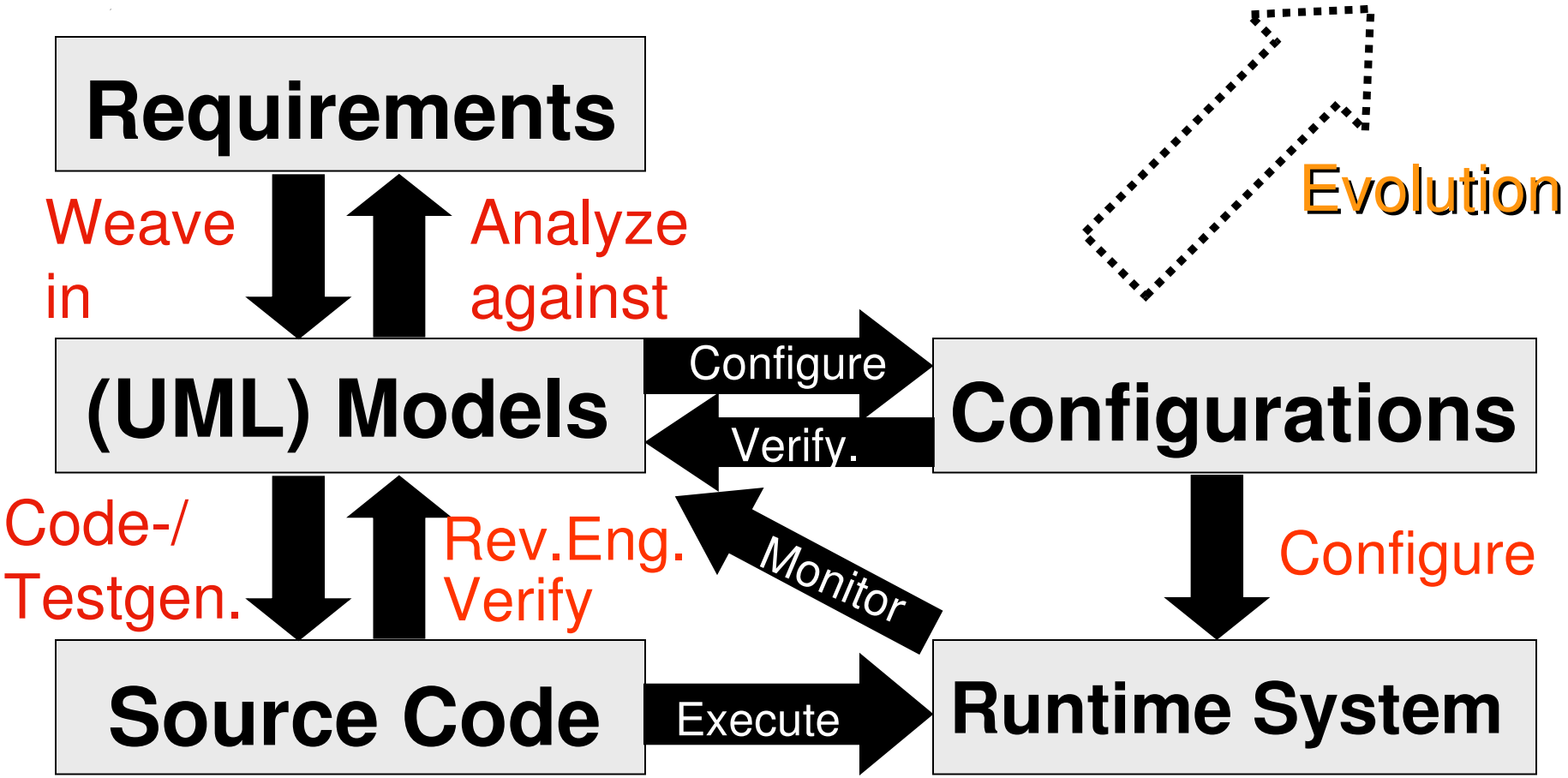
# Critical System Lifecycle




[McGraw 2003]

Model-based Security Engineering

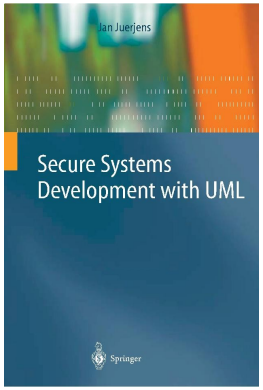
# Model-based Security Engineering



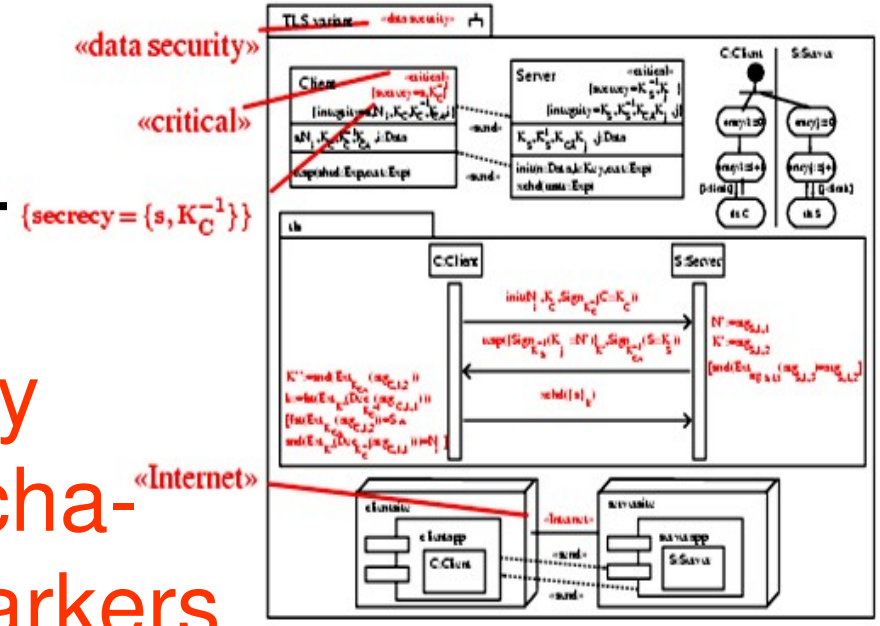
# Model-based Security with UMLsec

Extension of the Unified Modeling Language (UML) for secure systems development.

- evaluate UML models for security
- encapsulate established rules of prudent secure engineering
- make available to developers not specialized in secure systems
- consider security requirements from early design phases, in system context
- can use in certification



# UMLsec




Insert recurring security requirements, adversary scenarios, security mecha-nisms as predefined markers.

Use associated logical constraints to verify specifications using model checkers and ATPs based on formal semantics.

Ensures that UML specification enforces the relevant security requirements wrt Dolev-Yao type adversaries. [FASE01,UML02,FOSAD05,ICSE05]

# What Does UMLsec Cover ?

**Security requirements**: <<secrecy>>,…

**Threat scenarios:** Use $\text{Threats}_{\text{adv}}$(ster).

**Security concepts:** For example <<smart card>>.

**Security mechanisms:** E.g. <<guarded access>>.

**Security primitives:** Encryption built in.

**Physical security:** Given in deployment diagrams.

**Security management:** Use activity diagrams.

**Technology specific:** Java, CORBA security.

# Refinement, Composability, Aspects, Services

Need to refine models down to code.

Common formalizations of security properties not preserved by refinement.

Bad: re-verify after each step (incl code).

Theorem: Our notion of model refinement preserves security requirements. [FME01]

Similar: Established composability for certain security requirements under suitable assumptions. [Concur01]

Also: Demonstrated how to apply security using aspect-oriented weaving / service orientation. [ICSOC 04, Models 05]

# Layered Security Architectures

System layer on top uses security services below.

client authenticity

confidentiality, integrity, server authenticity

?

=

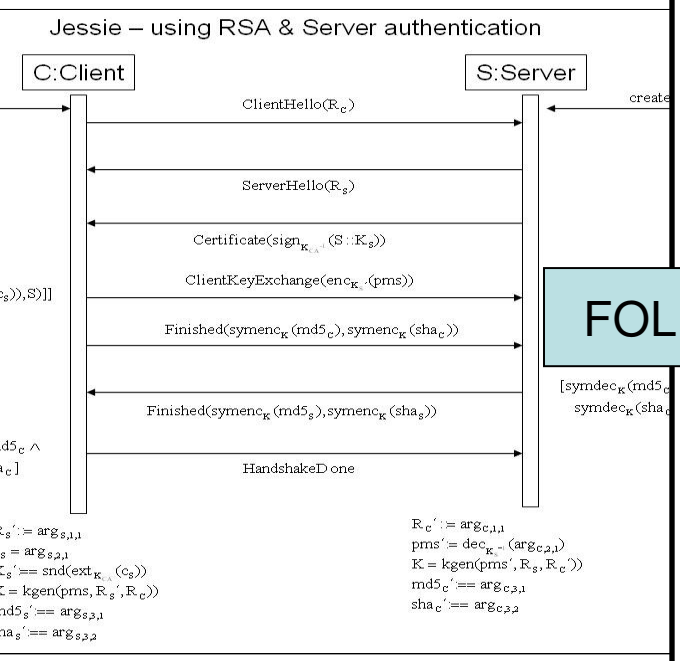
confidentiality, … + client authenticity

Security properties additive ? [Safecomp03]

Theorem: Yes, under suitable conditions.

# Model Verification

$$\forall arg_1, \ldots, arg_n \;.\; (knows(arg_1) \wedge \ldots \wedge knows(arg_n) \wedge cond(arg_1, \ldots, arg_n) \Rightarrow knows(exp(arg_1, \ldots, arg_n)))$$




FOL

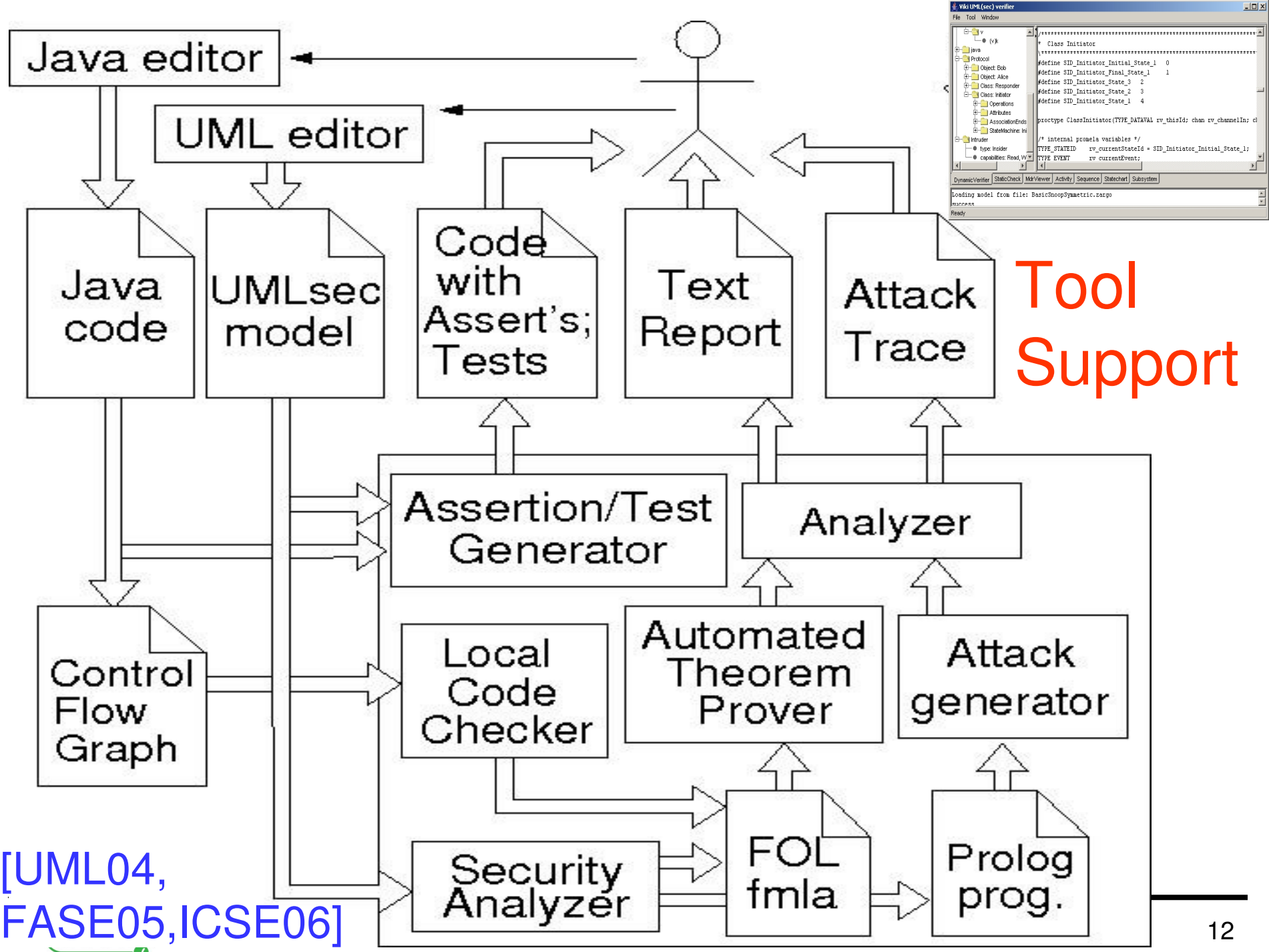
```
...
((
  knows(ArgC_3)
  & (equal(fst(ArgC_3), type_serverkeyexchange))
  & (equal(snd(ext(snd(snd(ArgC_3)), k_ca)), skey))
  & (equal(snd(ext(snd(ArgC_2), k_ca)), fst(snd(ArgC_3))))
)
=>(
  ((knows(ArgC_4_1)
    & equal(ArgC_4_1, type_serverhellodone))
    =>(
  ( ( true & equal(ClientKeyExchange, enc(premasterkey, skey))
)
...
%------------------------ Conjecture --

input_formula(attack,conjecture,(
   knows(mastersecret) )).
```

ATP

```
analyzing results ...
model found/total failure
time limit information: 19 total / 18 strategy
(leaving wrapper).
task myUML_PID1491 on atbroy1 has status SUCCESS
(model found by strategy 300) consuming 1 seconds
deleting temporary files.
e-SETHEO done. exiting
```

[FASE05,ICSE05, ICSE06]

secure CHANGE


Java editor
UML editor
Java code
UMLsec model
Code with Assert's; Tests
Text Report
Attack Trace
Tool Support
Assertion/Test Generator
Analyzer
Control Flow Graph
Local Code Checker
Automated Theorem Prover
Attack generator
Security Analyzer
FOL fmla
Prolog prog.
[UML04, FASE05,ICSE06]

# Security Analysis: Model or Code ?

Model:

+ earlier (less expensive to fix flaws)
+ more abstract ➔ more efficient
- more abstract ➔ may miss attacks
- programmers may introduce security flaws
- even code generators, if not formally verified

Code:

+ „the real thing“ (which is executed)

➔ Do both where feasible !

# Model vs. Implementation

„meaning“

„meaning“

**compare meaning!**

Backtrace assignments

Defined during model creation

Sent and received data

Elements of connections

Find

Has

**Implement -ation**

.java

Equal?

Jessie – using RSA & Server authentication

C:Client

S:Server

create(S)

ClientHello($R_c$)

ServerHello($R_s$)

Certificate(sign$_{K_{CA}^{-1}}$(S::$K_s$))

ClientKeyExchange(enc$_{K_{CA}}$(pms))

[[equal(fst(ext$_{K_{CA}}$($c_s$)),S)]]

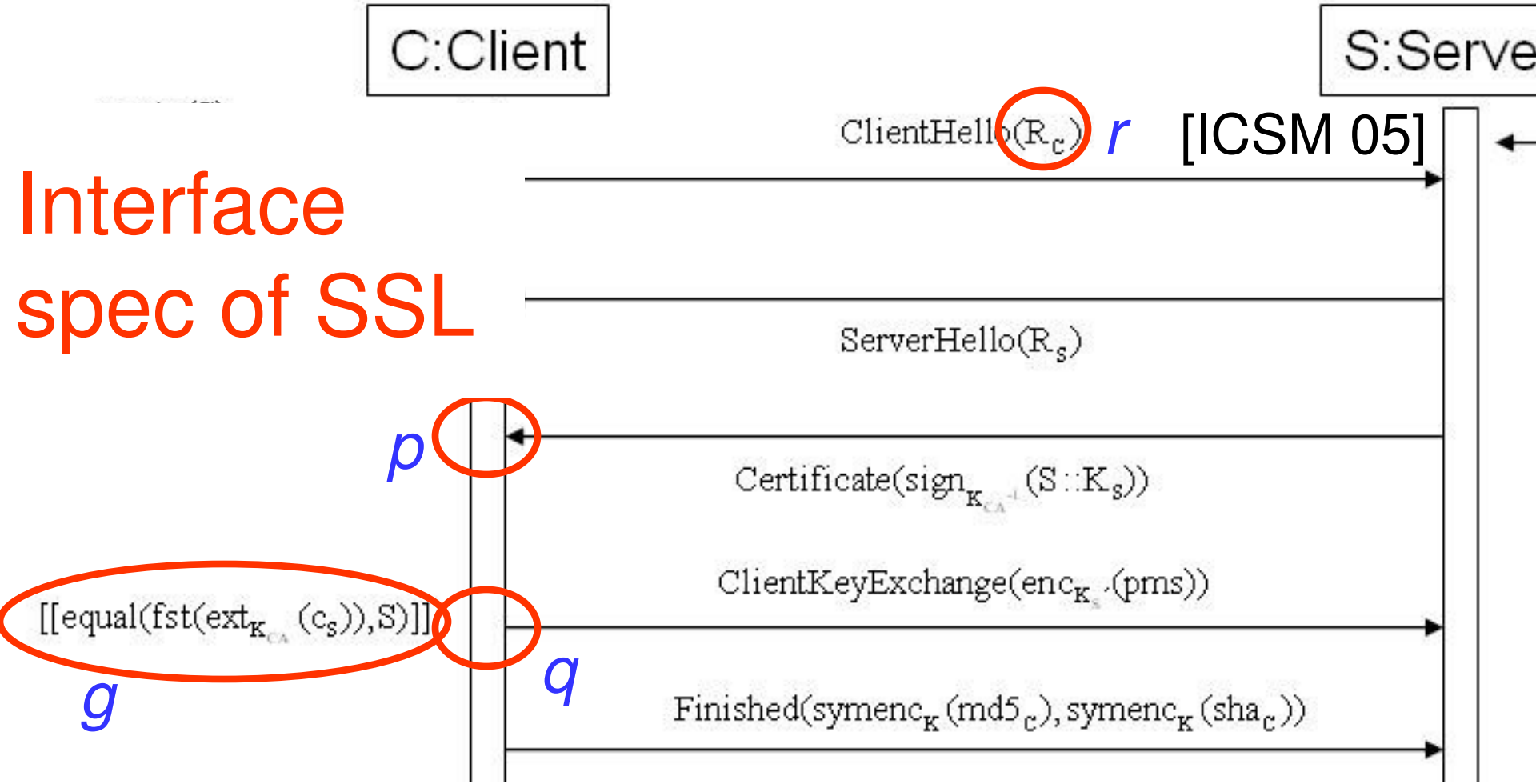
Finished(symenc$_K$(md5$_c$), symenc$_K$(sha$_c$))

Finished(symenc$_K$(md5$_s$), symenc$_K$(sha$_s$))

[symdec$_K$(md5$_s$') = md5$_c$ ∧ symdec$_K$(sha$_s$') = sha$_c$]

**Abstract model**

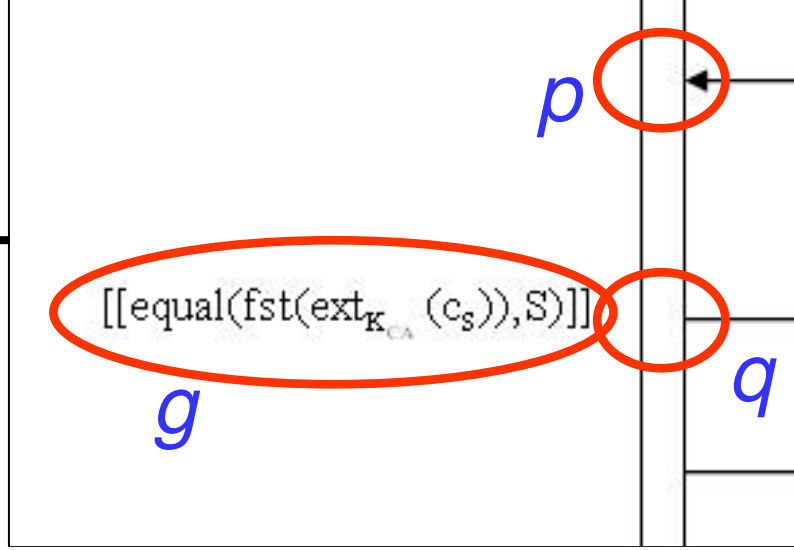
# Interface spec of SSL




- Identify program points:
  value (*r*), receive (*p*), guard (*g*), send (*q*)

II) Check guards enforced

# Checking Guards




Guard *g* enforced by code?

- Generate runtime check for *g* at *q* from diagram: simple + effective, but performance penalty.
- Testing against checks (symbolic crypto for inequalities). [ICFEM02]
- Automated formal local verification: conditionals between *p* and *q* logically imply *g* (using ATP for FOL). [ASE06]

Execution Path Diagram for method SSLSocket.doClientHandshake()

msg = Handshake.read(din, certType);

p

try

catch

Navigator

only possible way without throwing exception

g

serverKex = se

session.trustManager.checkServerTrusted (peerCerts,suite.getAuthType());

q

msg = new Handshake(Handshake.Type.CLIENT_
msg.write (dout, version);

p

[[equal(fst(ext$_{K_{CA}}$ (c$_S$)),S)]]

q

g

Class Sequence Flow

secure CHANGE

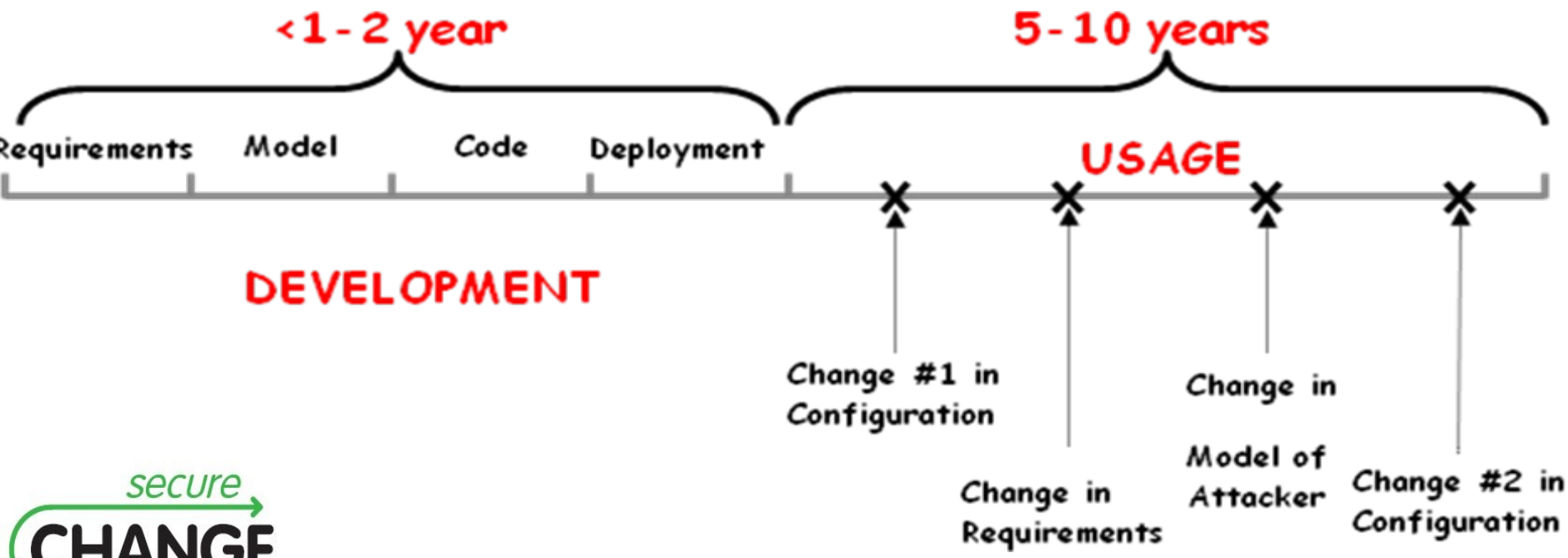
# System Evolution

Systems increasingly live longer than planned

Example: Year-2000-bug

Problem: Quality assurance for complex evolving systems

=> EU Project “Secure Change”



# Maintaining Traceability under Evolution

Code evolution can potentially multiply testing effort.

Need to re-verify code parts that changed, re-do integration testing etc.

Particular problem when testing for sophisticated properties (such as security) since requires particular effort.

➔ Want to automatically trace code evolution to model level so can automatically reuse earlier tests.

[Bauer, Jurjens, Yu 09]

# Model-Code Co-Evolution

Basic observation: Most system changes can be reduced to two kinds:

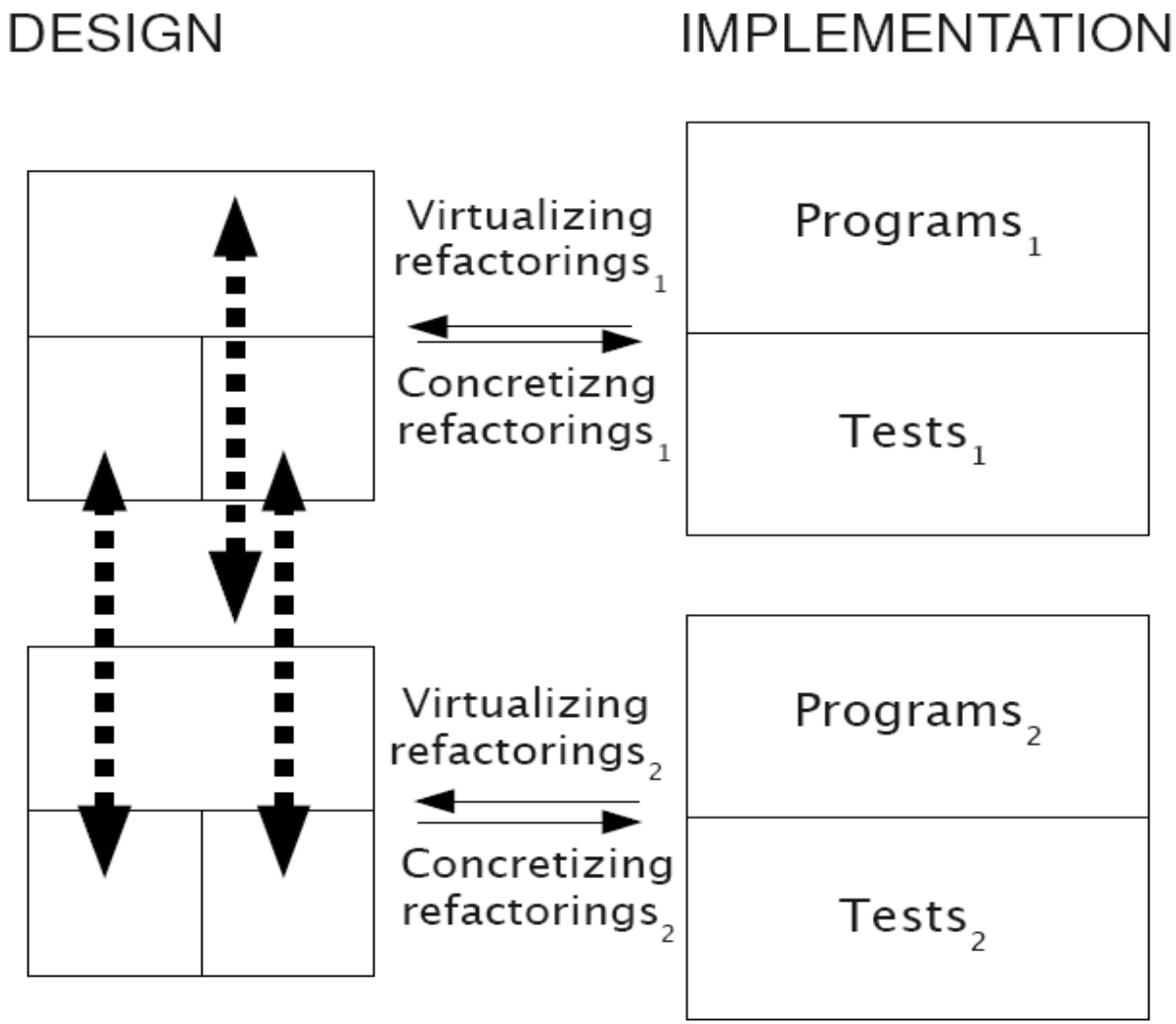
- Adding / removing parts of the system.
- Basic refactoring operations to hold system parts together despite changes.

When adding / removing code parts we need to assume that the corresponding models are also added / removed.

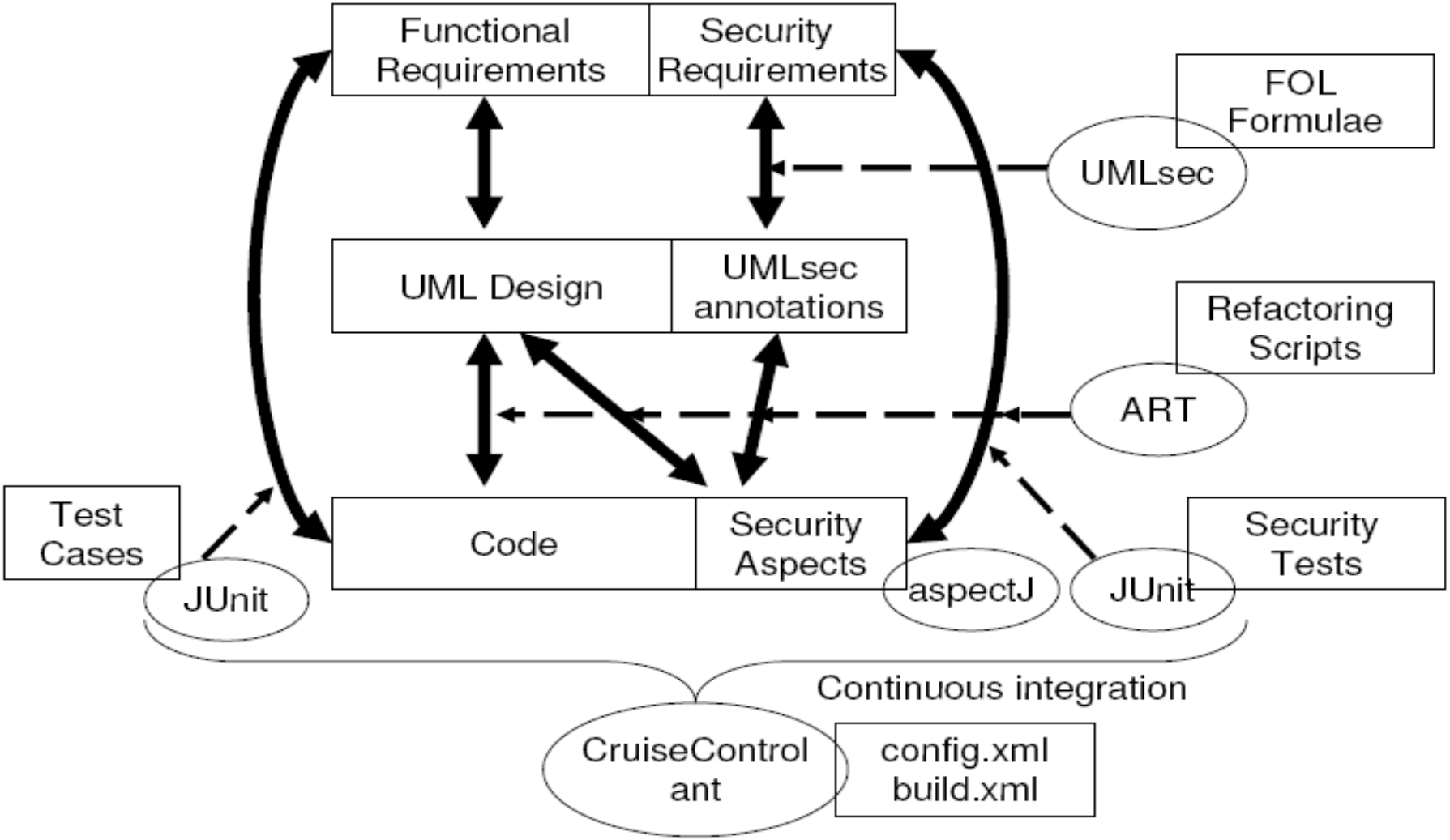
For evolution by refactoring can achieve automated model-code traceability e.g. using Eclipse Refactoring Language Toolkit (LTK) / XML based refactoring scripts.

Maintain model-code synchrony using continuous integration scripts (with CruiseControl / Apache Ant).

# Evolution as Code Refactoring



# Refactoring: Tool Supports



# Java Secure Sockets Extension

Applied our approach to a series of implementations of the Java Secure Sockets Extension library:

- Jessie 1.0.0
- Jessie 1.0.1
- JSSE 1.6

Demonstrated that our model-code co-evolution approach is robust even across major software changes.

# Refactoring JSSE

| Symbols | Program entities | Identif. | Refactoring op. |
|---|---|---|---|
| 1. $C$ | clientHello | C | rename.type |
| 2. $S$ | serverHello | S | rename.type |
| 3. $P_{\text{ver}}$ | session.protocol<br>version | P_ver | extract.temp |
| 4. $R_C$<br>$R_S$ | clientRandom<br>serverRandom | R_C<br>R_S | rename.local.variable<br>rename.local.variable |
| 5. $S_{\text{id}}$ | sessionId<br>sessionId | S_id<br>S_id | rename.field<br>rename.local.variable |
| 6. Ciph[ ] | session.enabledSuites | Ciph | extract.temp |
| 7. Comp[ ] | comp | Comp | extract.temp |
| 8. Veri | Lines 1518–1557 | Veri | extract.method |

# Refactoring: Performance

| Messages in sequence | op. | diff | Time (sec) |
|---|---|---|---|
| S1: $C \rightarrow S : (P_{\text{ver}}, R_C, S_{\text{id}}, \text{Ciph}[\,], \text{Comp}[\,])$ | 7 | 31 | 13.891 |
| S2. $S \rightarrow C : (P_{\text{ver}}, R_S, S_{\text{id}}, \text{Ciph}[\,], \text{Comp}[\,])$ | 5 | 20 | 9.437 |
| S3. $S \rightarrow C : \text{Certificate}[\text{X509Cert}_s]$ | 2 | 2 | 1.474 |
| S4. $C : \text{Veri}(\text{X509Cert}_s)$ | 2 | 2 | 3.854 |
| ... | ... | ... | ... |
| Total of 7 messages and 3 checks | 27 | 86 | 40.303 |

# Applications of MBSE

Analyzed designs / implementations / configurations for

- biometry, smart-card or RFID based identification
- authentication (crypto protocols)
- authorization (user permissions, e.g. SAP systems)

Analyzed security policies, e.g. for privacy regulations.

# Conclusions

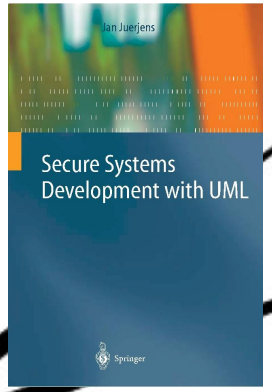
Model-based Security Engineering using UMLsec:

- aims to covers all kinds of security aspects.
- formally based approach
- automated tool support
- industrially used methods
- integrated approach (models, source-code, configuration data)
- ongoing Secure Change project: support system evolution

# Overview

Security Engineering

- Patterns
- Services
- Architectures
- Components
- Aspects
- Layers
- Requirements
- Processes
- Models
  - UMLsec
  - Formal Semantics
  - Analysis Framework
  - Model Checking
  - Autom. Theorem Proving
  - Refinement
- Applications
  - CEPS
  - Biometry
  - Cryptokey
  - Jessie
  - TLS variant
  - Wiesncard
- Code
  - Runtime Checks
  - Model-based Testing
  - Autom. Theorem Proving
- Foundations





Dependable Systems Development

- Fault-tolerance
- Reliability
- Real-time
- Security
  - Engineering
  - Management
    - Risk Assessment
    - Permissions
    - Business Processes
    - Security Investment
    - Firewall Configurations

# Ongoing Work

- Security Verification of Crypto Protocol Implementations in C: Use VCC to verify C code. (with Andy Gordon, MSR Cambridge; RS Industrial Fellowship & 2 PhD projects)
- Modelling for Compliance (EPSRC CASE PhD project with British Telecom)
- Security Engineering for Lifelong Evolvable Systems (EU FP7 Integrated Project)
- RS Joint International Project with TU Munich on Formal Security Analysis of Cryptoprotocol Implementations
- RS Joint International Project with NII Tokyo on Relating Security Requirements and Design
- HIRING NOW: Postdocs / PhD students (at TU Dortmund / Fraunhofer ISST, Germany) !

# Questions?

More information (papers, slides, tool etc.):

http://jurjens.de/jan

J.Jurjens@open.ac.uk