Software Engineering für kritische Systeme

Jan Jürjens

technische universität dortmund

Fraunhofer ISST

http://jan.jurjens.de

Wer bin ich ?

Ab 09/2009:

- Professor für Angewandte Informatik (insb. Software Engineering) an der TU Dortmund
- Wissenschaftskoordinator „Enterprise Engineering“ am ISST
- Leiter der Fraunhofer-Attract-Projektgruppe „Architectures for Auditable Business Process Execution (Apex)“ (2009-2014)
- Wissenschaftlicher Direktor des EU-Projektes „Security Engineering for Lifelong Evolvable Systems (Secure Change)“
- Ehrenhalber: Senior Member am Robinson College (Univ. Cambridge)

Vorher u.a.:

- Royal Society Industrial Fellow bei Microsoft Research Cambridge (formale Verifikation von Implementierungen kryptographischer Protokolle in C)
- Research Fellow am Robinson College (Univ. Cambridge)
- Postdoc an der TU München (modellbasierte Entwicklung sicherer Systeme)
- Promotion zu „Principles for Secure Systems Design“ (Univ. Oxford)
- Forschungsaufenthalte am LFCS (Univ. Edinburgh) und Bell Labs (Palo Alto) (komplexitätstheoretische vs. symbolische Verifikation von kryptographischen Protokollen)
- Studium Mathematik (Nebenfach Informatik) an Univ. Bremen und Univ. Cambridge (Diplomarbeit in kategorieller Modelltheorie)

Vertrauenswürdige IT-Systeme

IT-Systeme durchziehen heute fast alle Aspekte menschlichen Lebens.

Gleichzeitig werden IT-Systeme offener und daher leichter angreifbar.

Den gestiegenen Anforderungen an ihre Vertrauenswürdigkeit werden die Systeme aber oft nicht gerecht.

Sichere Software und ihre Tücken

Klassische Schwachstelle bei Benutzeranmeldung in einer alten Unix-Version:

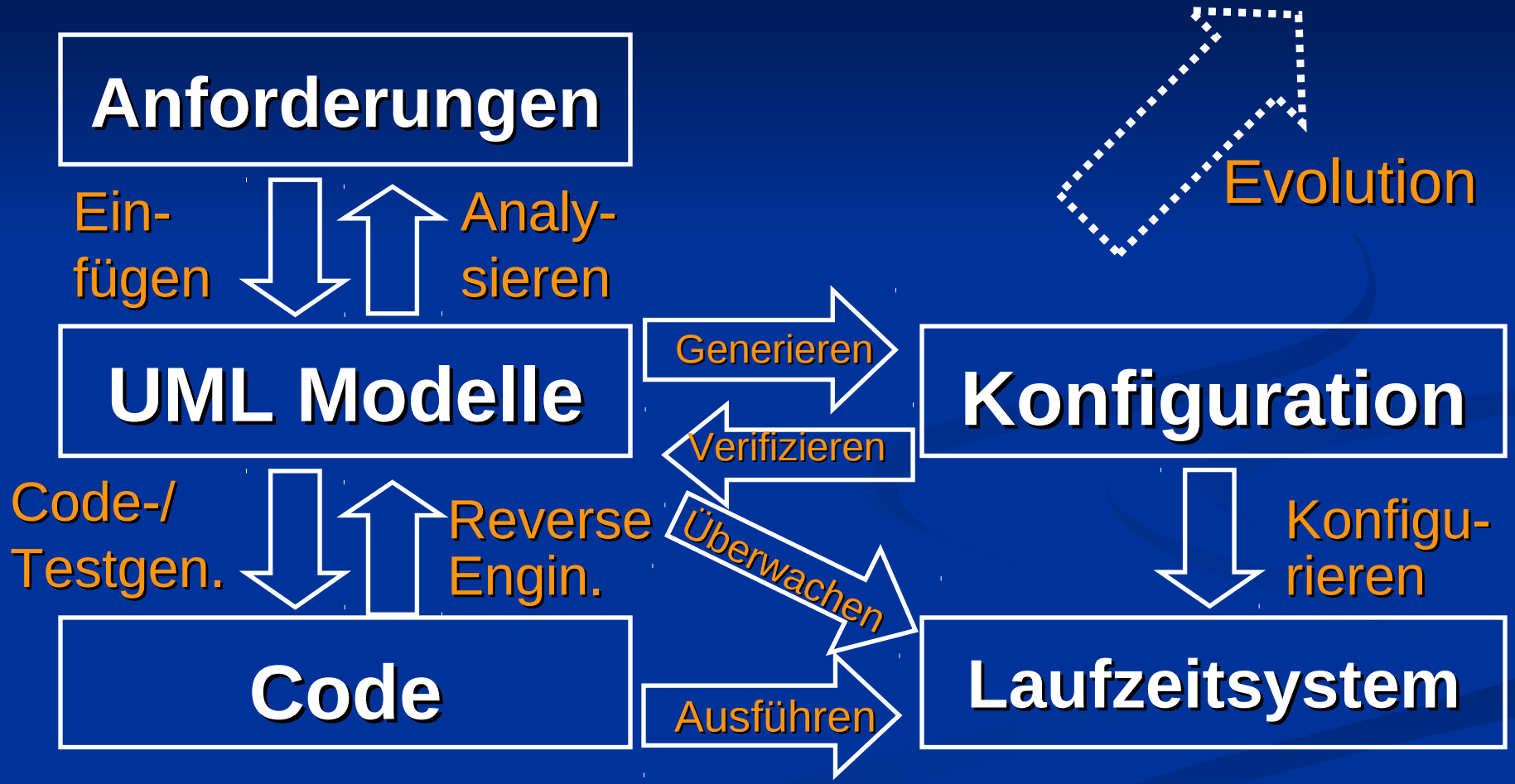
“wrong password”-Fehlermeldung beim ersten falschen Zeichen im Passwort.

Durch Messung der Antwortzeit kann der Passwort-Suchraum von x^n auf x*n reduziert werden

(x = Alphabetgröße, n = Passwortlänge. Beweis: Übungsaufgabe :-)),

womit das Passwort unsicher wird .

Modellbasierte Entwicklung



Software Engineering für kritische Systeme

Ziel: Ansatz für Entwicklung und Einsatz kritischer Systeme.

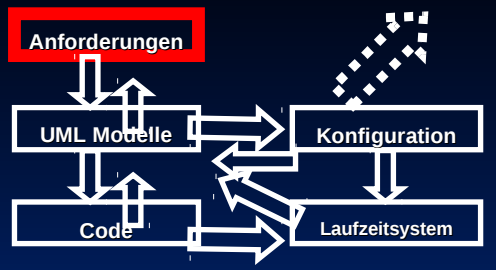
Fokus in diesem Vortrag auf Security. Ist aber nur eines von vielen wichtigen Qualitätsattributen, Ideen lassen sich auch auf andere übertragen (z.B. Fehlertoleranz, Performanz).

[UML '04 / Journ. Systems & Software '09]

Anwendung auf Altsysteme (ggf. nicht modell-basiert entwickelt) durch Verwendung von Reverse-Engineering, insbesondere zur Extraktion von Domänenwissen.

[CSMR 07, CSMR 08, IPCP 08]

Security Requirements Engineering



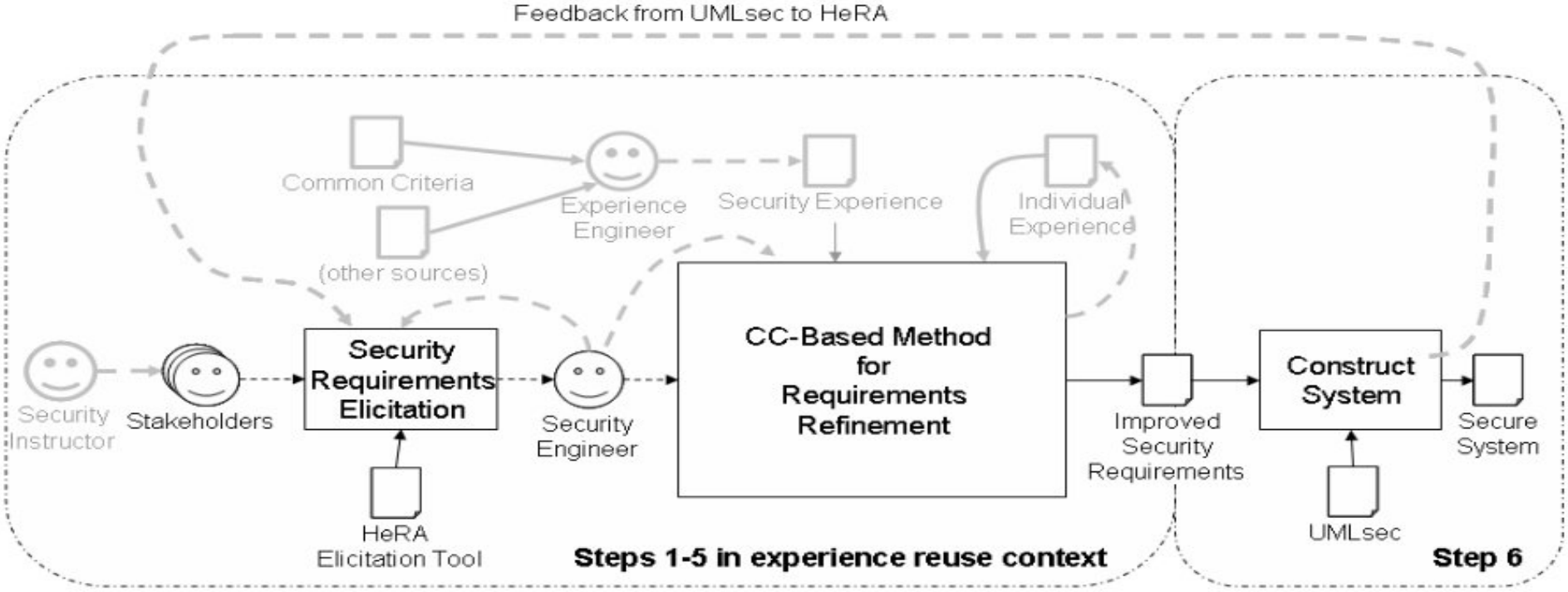
Ziele:

- Sicherheitsanforderungen innerhalb der Anforderungsanalyse ermitteln

Idee: “Requirements Mining” in Security Standards (z.B. Common Criteria) bzw. im gegebenen Spezifikationsdokument.

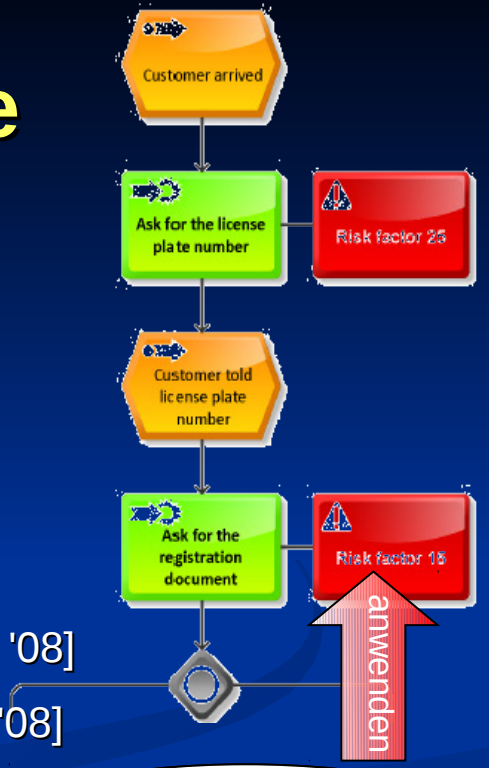
Validierung: IPTV Standard des Europ. Telecom. Standards Inst. (ETSI)

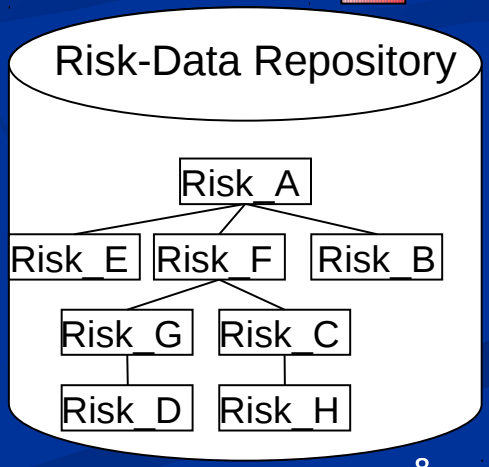
[CAISE '06, Requirem. Engin. Jour. '10, Journ. Softw. & Systems Modeling '10]



Governance, Risk & Compliance auf Geschäftsprozessen

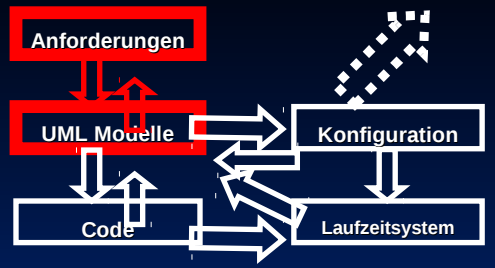
- Idee: Automatische Analyse von Geschäftsprozessmodellen auf operationale Risiken, z.B.:
  - gegenüber Benutzerberechtigungen zur Laufzeit, sowie der Benutzer-berechtigungen gegenüber der Sicherheitspolitik, [FASE '08]
  - automatische Risiko-Identifikation und -Bewertung [ICSE '08]
- Laufendes Projekt (Fraunhofer Attract): Architekturen für auditierbare Geschäftsprozessausführung (Apex).





8

Modellierung mit UMLsec

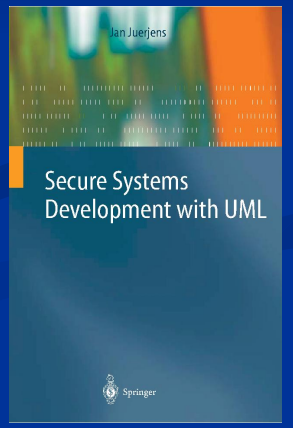


Ziel:

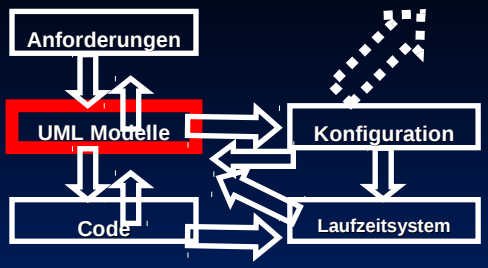
- Dokumentation und automatische Analyse von sicherheits-relevanten Informationen (z.B. Sicherheits-Eigenschaften und -Anforderungen) als Teil der Systemspezifikation.

Idee:

- UML für System-Modellierung. [FASE 01, UML 02]
- Sicherheitsinform. als Markierungen (Stereotypen) einfügen. Definiere dazu UML-Erweiterung UMLsec.
- Formale Semantik mit stromverarbeitenden Funktionen als Grundlage für Verifikation.



[Jour. Logic & Algebr. Program. '08]



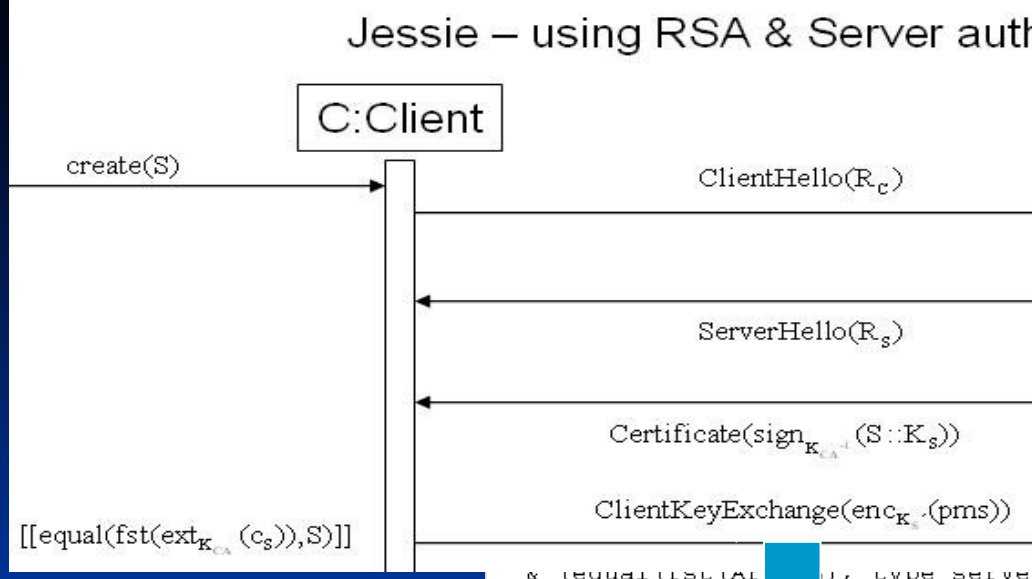
Modell-Analyse

Ziel:

- Automatische Analyse von Modellen und darin spezifizierten Sicherheits-Eigenschaften des Systems gegenüber spezifizierten Sicherheits-Anforderungen.

Idee: [ICSE 05, ICSE 06]

Automatische Generierung der logischen Formeln für die Sicherheitsanalyse in 1.-Stufe-Logik (bzw. LTL, ...) auf Basis der formalen Semantik. Übergabe an automatischen Theorembeweiser (bzw. Modelchecker / ...).

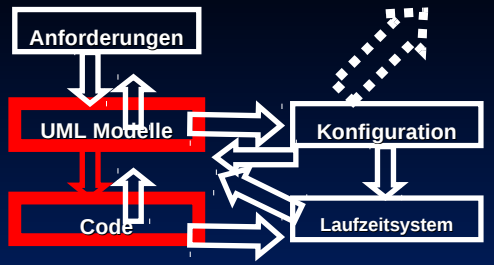



```
  & (equal(snd(ex      d(snd(ArgC_3)
  & (equal(snd(ext     nd(ArgC_2), k_
)
=>(
  ((knows(ArgC_4_1)
    & equal(ArgC_4_1, type_serverh
    =>(
  ( ( true & equal(ClientKeyExchan
)
...
%------------------------

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

```
analyzing results ...
model found/total failure
time limit information: 19 t
(leaving wrapper).
task myUML_PID1491 on atbroy
(model found by strategy 300
deleting temporary files.
e-SETHEO done. exiting
```

Modell-basiertes Testen



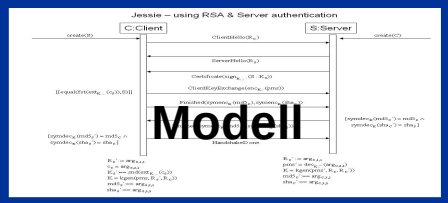
Probleme mit Konformanz-Testen:

- Komplette Test-Abdeckung i.A. nicht praktikabel.
- Findet i.A. nur Angriffe, die auf Modellebene sichtbar sind.

Idee: Mutations-Testen. [ASE 01, ICFEM 02]

- Schwerpunkt auf kritische Testfälle
- Findet auch Schwachstellen, die auf Modellebene nicht sichtbar sind.

Validierung: Common Electronic Purse Specifications.
Mehrere Schwachstellen aufgedeckt.



Verifikation

Programm

Testfall-generierung

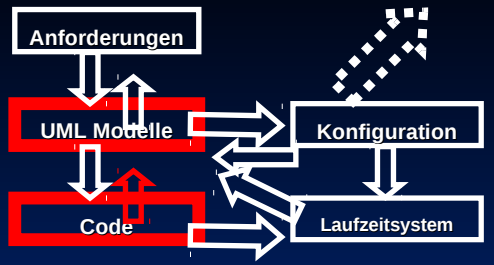
Testfälle

Konform ?

Ausführung

Programmverhalten

Statische Programmanalyse

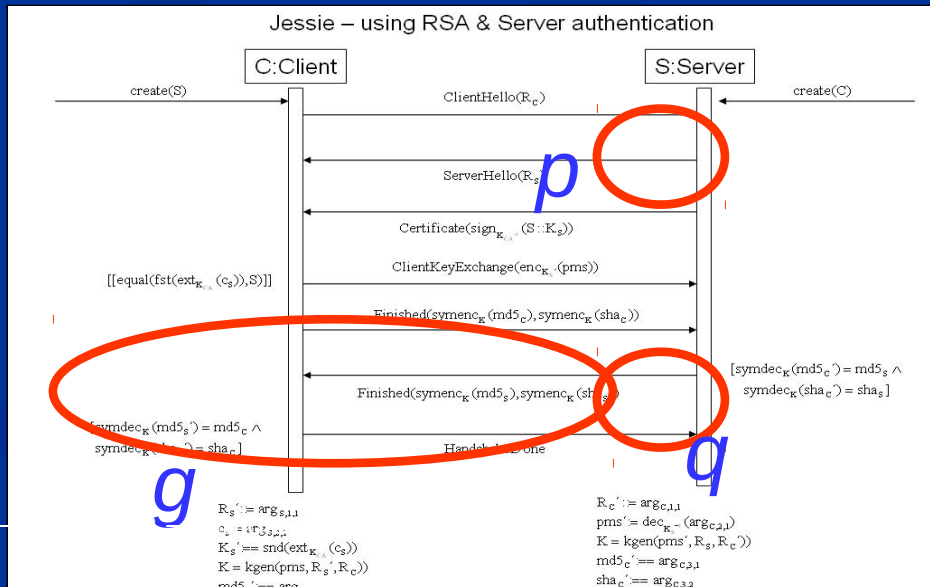


Problem: Korrekte Verwendung von Kryptographie inhärent schwierig zu testen: Ausreichende Testabdeckung vergleichbar mit brute-force Angriff. [ICSM 05, ASE 05, ASE 06]

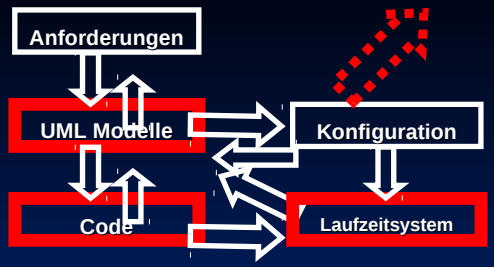
Idee: Automatische, formale statische Programmanalyse des korrekten Aufrufs von kryptograph. Funktionen (mit ATP für FOL).

Validierung: Java Secure Sockets Extension (JSSE).

Laufendes Projekt Csec:
Analyse von C Code.



Und noch ein Problem...



Allgemeines Problem: Verifizierte Implementierung im Systemkontext sicher?

- Berücksichtigt statisches Systemmodell alle relevanten Aspekte ?
- Sind die Annahmen über die Systemumgebung korrekt ?
- Sind für statische Verifikation notwendige Abstraktionen zulässig ?

➔ Run-time verification.

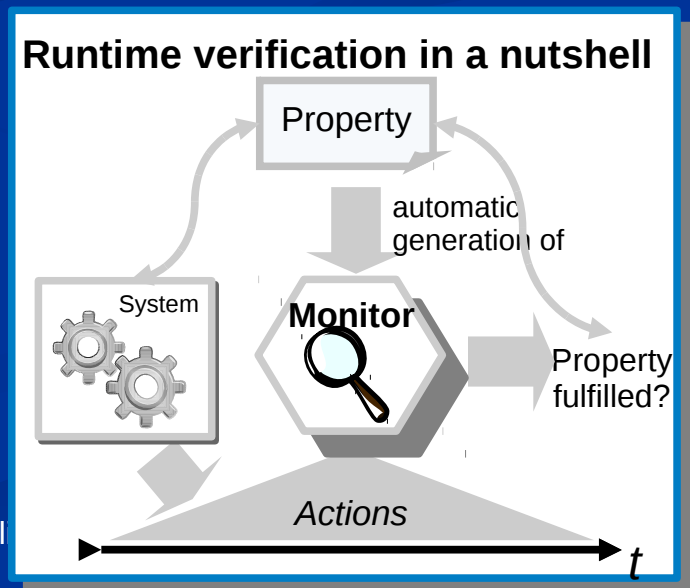
Klassischer Ansatz: Fred Schneider's Security Automata (nur Safety).

Neuer Ansatz mit 3-wertiger Semantik für LTL: Auch Nicht-Safety.

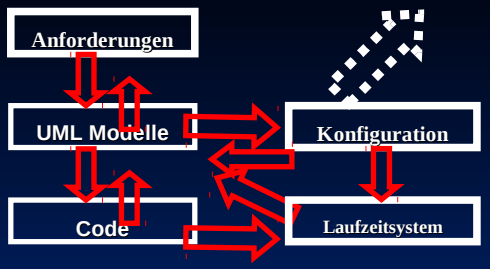
Validierung mit verschiedenen Versionen der Java Secure Sockets Extension.

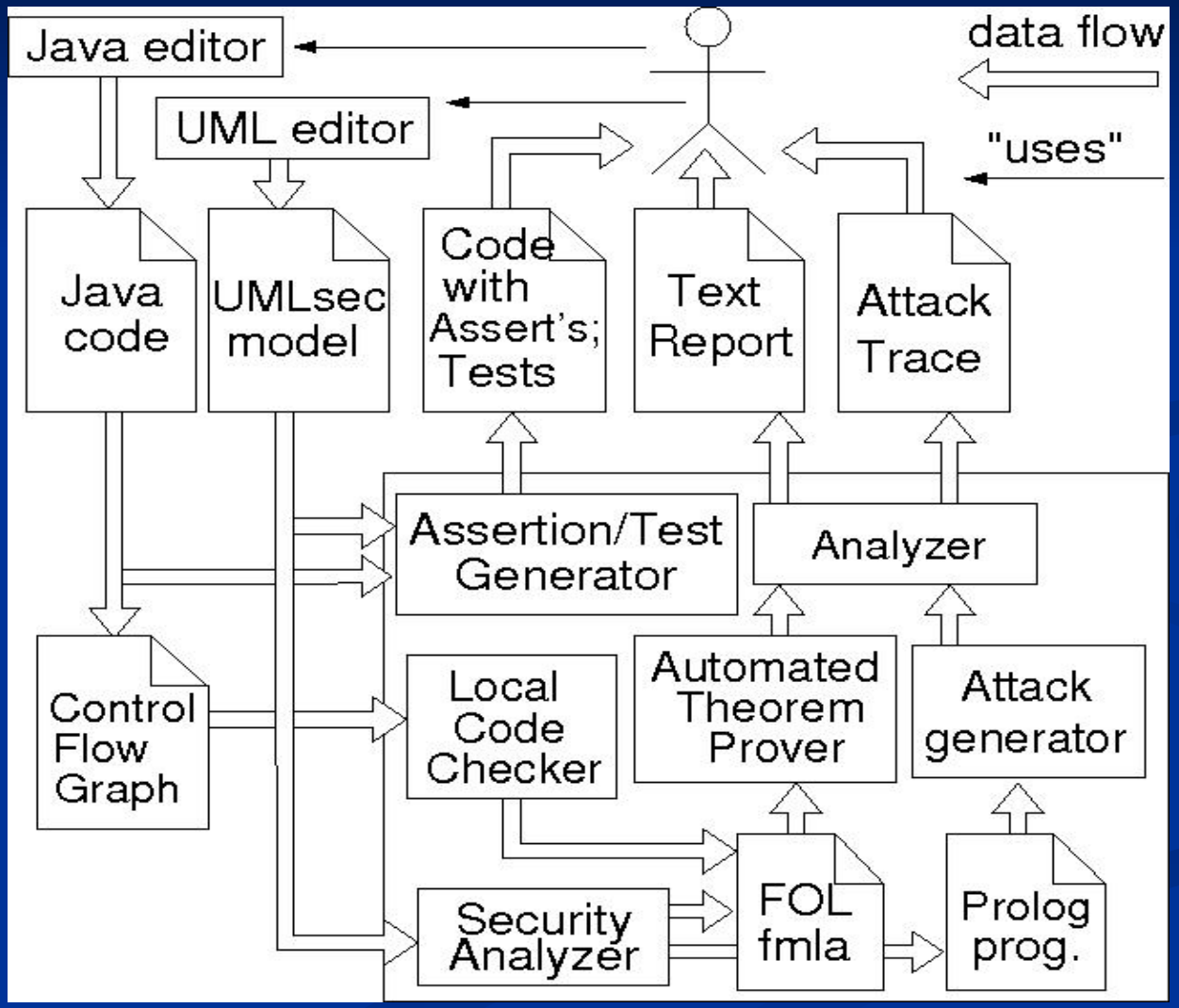
[Diss. A. Bauer]

[Jour. Computers & Security '10, Computer Journal '10]



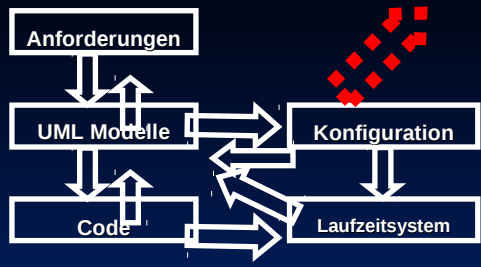
Werkzeugunterstützung





[UML 04, FASE 05, Jour. Softw. Tools & Techn. Transf. (STTT) 07]

Das vergessene Ende des System-Lebenszyklus



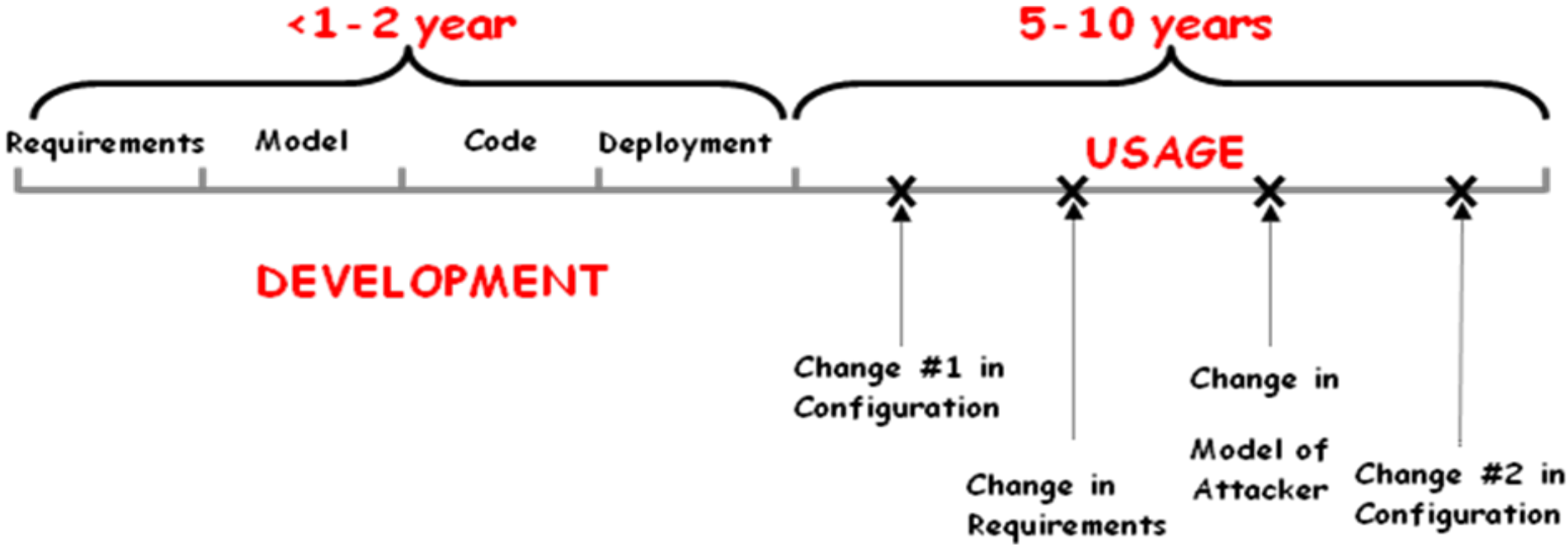
Herausforderungen:

- Lebenszeit oft länger als geplant (vgl. Jahr-2000-Fehler)
- Systeme evolvieren während Lebenszeit
- Evolution in der Praxis schwer zu beherrschen [vgl. Bank Bsp.].

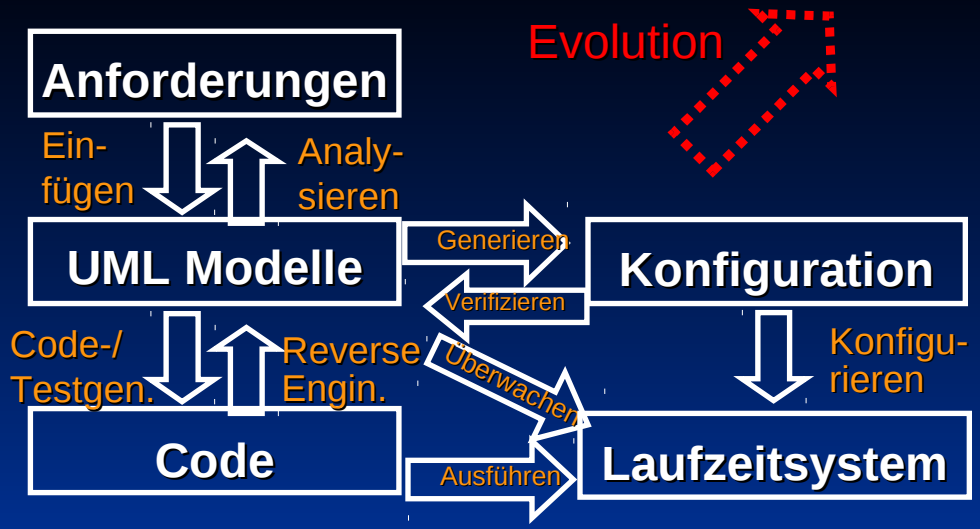
Problem: Werden kritische Anforderungen (z.B. Sicherheit) bewahrt ?

secure CHANGE

http://securechange.eu



Herausforderung Evolution



Jedes Artefakt in obiger Abbildung evolviert.

Zur Kostenreduktion:
Verifikationsresultate soweit möglich wiederverwenden.

=> Unter welchen Bedingungen bewahrt Evolution Sicherheit ?

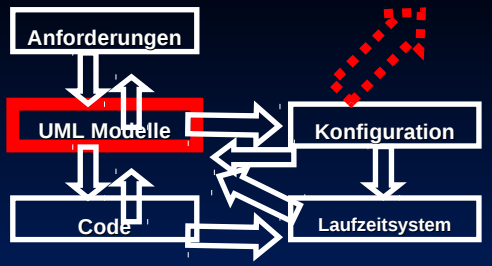
Noch besser: mögliche zukünftige Evolution auf Auswirkungen auf Sicherheit untersuchen, um bereits zur Design-Zeit die Architektur zu wählen, die Evolution und Sicherheit am besten unterstützt.

=> Evolution als Modellierungskonzept in UMLsec.

Trade-off: Flexibilität der Evolution vs. Bewahrung von Sicherheit.

[NB: Analoges Problem: Software-Produktlinien.]

Evolution vs. Software-Architektur



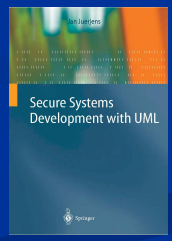
Idee: Allgemeine Prinzipen auf Einhaltung von Sicherheit untersuchen:

Architekturprinzipien:

- Schichtung von Architekturebenen [Safecomp'03]
- Modularisierung / Komposition von Architektur-Teilen [Concur'00]
- Dienstorientierte Architekturen [ICSOC'04]
- Aspektorientierte Architekturen [Models'05]

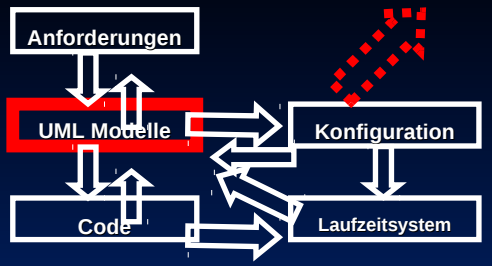
Entwicklungstechniken:

- Verfeinerung von Spezifikationen [FME'01]
- Refactoring
- Einschränkung der Schnittstelle
- Evolution bis auf definiertes Ein-/Ausgabeverhalten



Jeweils: Theorem, das Vorbedingung für Bewahrung von Sicherheit liefert.

Evolution vs. Verfeinerung



Idee: Spezifikation des Versionsraumes durch Unterspezifikation.

Ziel: Vermeidung erneuter Verifikation jeder instanziierten Version.

Frage: Unter welchen Voraussetzungen bewahrt Verfeinerung (d.h.: Konkretisierung einer Spezifikation) Sicherheitseigenschaften ?

Bei verhaltensbewahrender Verfeinerung würde man Bewahrung von Sicherheitseigenschaften erwarten.

„Verfeinerungsparadox“: Im Allgemeinen nicht der Fall.

Beobachtung: Problematisch: Vermischung von Nichtdeterminismus zur Unterspezifikation bzw. als Sicherheitsmechanismus.

Idee: Trennung auf Modellierungsebene.

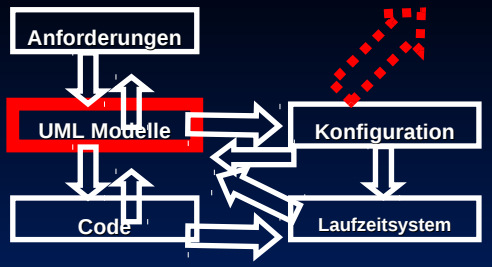
[FME 01]

Theorem: Verfeinerung bewahrt dann Sicherheitsanforderungen.

Theorem

- If P preserves secrecy of m and $P \rightsquigarrow Q$ then Q preserves secrecy of m.

Evolution vs. Modularisierung



[ASE '06]

Idee: Modularisierung der Architektur

Ziel: Erneute Verifikation nur für geänderte Komponenten.

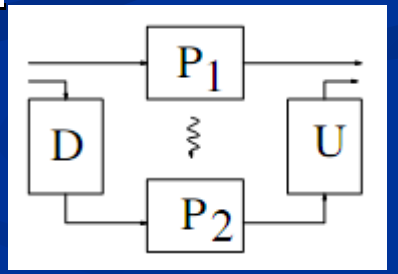
Frage: Unter welchen Voraussetzungen bewahrt Komposition von Komponenten Sicherheitseigenschaften?

Geht nur bei Annahmen an andere Komponente.

Idee: Formalisieren als „Rely-guarantee“-Bedingungen.

=> Komponente kann evolvieren, solange sie die Rely-guarantee Bedingung einhält.

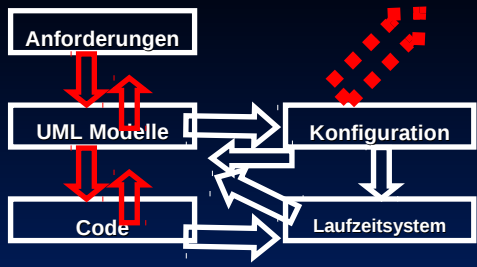
Anwendung: Java Secure Sockets Extension.



Theorem 5. *Let P_1, P_2, D and U be processes with $I_{P_1} = I_D$, $O_D = I_{P_2}$, $O_{P_2} = I_U$ and $O_U = O_{P_1}$ and such that D has a left inverse D' and U a right inverse U'. Let $m \in (\mathbf{Secret} \cup \mathbf{Keys}) \setminus \bigcup_{Q \in \{D', U'\}} (S_Q \cup K_Q)$.*

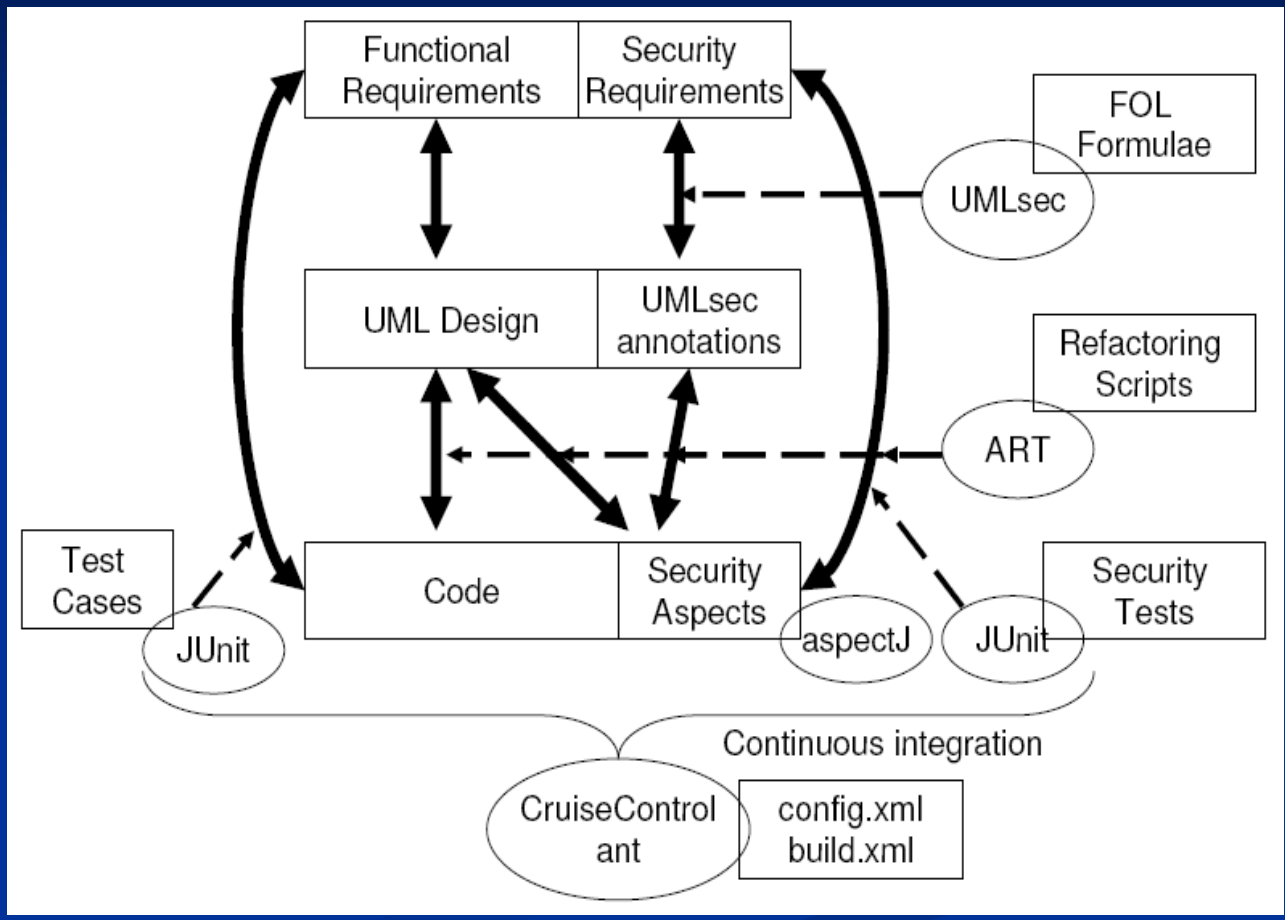
- *If P_1 preserves the secrecy of m and $P_1 \overset{(D,U)}{\rightsquigarrow} P_2$ then P_2 preserves the secrecy of m.*

Sichere Evolution: Werkzeugunterstützung



Nachverfolgbarkeit von Anforderungen vs. Implementierung bei Evolution durch Refactoring bewahren.

[ASE'07, ICSM'08, ASE'08, Computer Journal '10]



Anwendungsbeispiel: Internes Informationssystem bei BMW

MetaSearch Engine: Personalisierte Suche im Firmen-Intranet (passwort-geschützt).

[ICSE 07]

Einige Dokumente sehr sicherheitskritisch.

Über 1.000 potentielle Benutzer, 280.000 Dokumente, 20.000 Anfragen pro Tag.

Nahtlos in unternehmensweite Sicherheitsarchitektur integriert. Bietet Sicherheitsdienste für Anwendungen (Benutzerauthentisierung, rollenbasierte Zugangskontrolle, globales Single-Sign-On), Ansatzpunkte für weitere Sicherheitsdienste.

Erfolgreich mit UMLsec analysiert.

BMW Group

Weitere Anwendungen

- Gesundheitskarte: Architektur untersucht, Schwachstellen aufgedeckt [Jour. Meth. Inform. Medicine '08]
- Mobile Architekturen bei O_2 (Germany) [ICSE '08]
- Digitaler Formularschrank bei HypoVereinsbank [SAFECOMP '03]
- Common Electronic Purse Specifications [ASE '01]
  (Globaler Standard für elektronische Geldbörsen):
  mehrere Schwachstellen aufgedeckt
- Biometrische Authentisierungssysteme: mehrere Schwachstellen aufgedeckt [ACSAC '05, Models '09]
- Gesundheitsinformationssysteme [Caise '09]

Einige empirische Resultate

Lohnen sich automatische / modellbasierte QS-Ansätze im Vergleich mit klassischen Techniken (z.B. Testen) ?

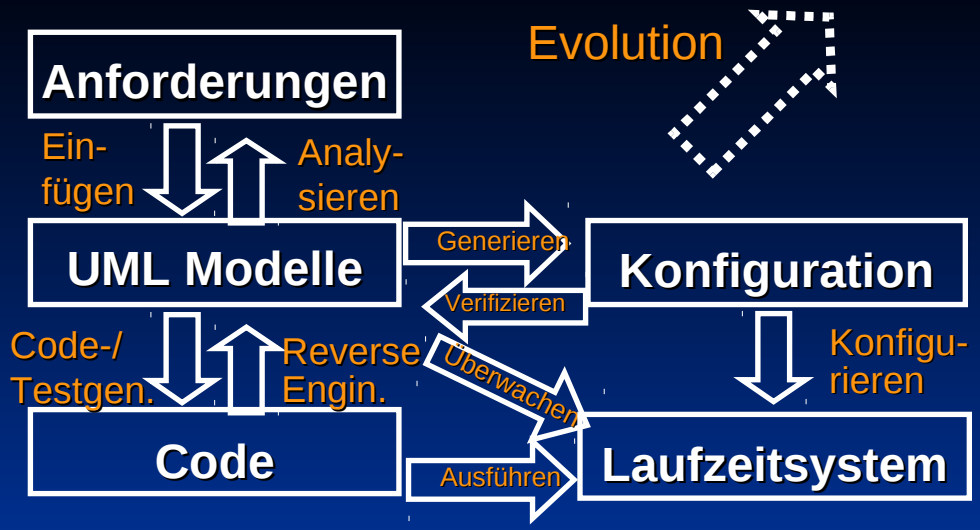
Statische Analyse vs. Code Review: industrielle Software (bei O2 (Germany)) auf Fehler untersucht. Ergebnisse: [Testcom '05]

- Statische Analyse-Werkzeuge finden nur bestimmte Fehlerklassen, aber besonders verlässlich.
- Wichtigstes Ziel: “false positive”-Rate verringern.

Modelchecking vs. Simulation / Testen: Türsteuergerät (Kooperation mit BMW). Typische Fehlerklassen: [Models '08]

- Simulation / Testen findet viele “einfache” Fehler schnell und effizient (z.B. falsche Transitionspriorität: einige Min.).
- Modelchecking findet auch obskure Fehler (z.B. race conditions), aber Zusatz-Aufwand (1-2 Tage für LTL Formel).

Zusammenfassung



- Modell-basierte Entwicklung mit UML
- Automatische Sicherheits-Analyse von Software Artefakten:
  - UML Modelle, Java / C Programme, Konfigurationsdaten
- Erfolgreiche Anwendungen in der Industrie.
- Aktuelle Projekte: Apex, Secure Change, Csec
- Geplante Arbeit: Security & Compliance in Clouds

Mehr Information: http://jan.jurjens.de