Sichere Evolution für verteilte Software-Systeme

Jan Jürjens

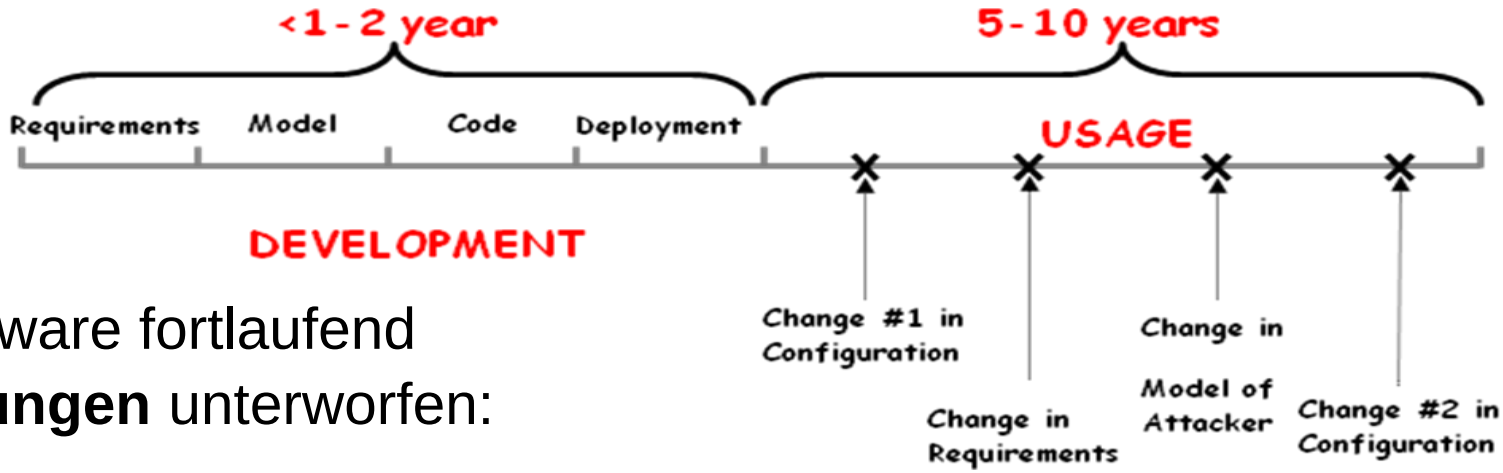
TU Dortmund

http://jan.jurjens.de

Was ist Softwareevolution ?

Software oft „**langlebiger**“ als geplant (vgl. Jahr-2000-Fehler).

„Nutzt“ sich nicht ab.



Aber: Software fortlaufend **Änderungen** unterworfen:

- Änderungen der **Anforderungen**
- Änderungen der **Softwareumgebung**
- **Fehlerkorrektur**

➔ „**Softwareevolution**“

Was ist dabei die Herausforderung ?

Nach Änderung Software neu testen (**Regressionstest**):
Alle Tests wiederholen (auch wenn nur Umgebung geändert).
Hohe Kosten:

- **Jahr-2000**-Problem (zweistellige Jahreszahlen):
  Weltweit insges. ca. 600 Mrd US$ Kosten.

➔ Oft **unzureichende Regressionstests**:

- Selbstzerstörung der **Ariane 5** (1996):
  Ariane 4-Software ohne Test in Ariane 5 wiederverwendet. 1 Mrd US$ Schaden.

Insbesondere **nachträglicher** Einbau von **Sicherheitsmaßnahmen** oft problematisch…

Welche Softwareengineering-Techniken können helfen?

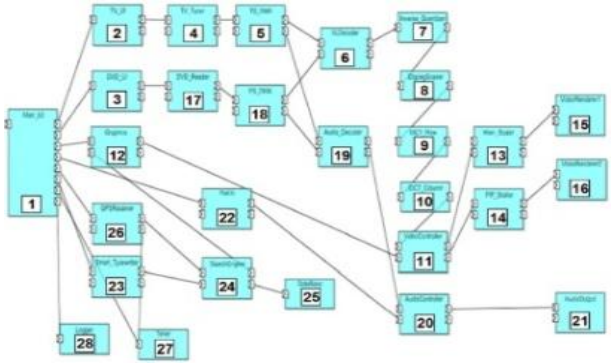
Hilfreich: Designtechniken und Architekturprinzipien, die Management von Evolution unterstützen.

Designtechnik: **Verfeinerung von Spezifikationen.**

➔ Evolution zwischen Verfeinerungen einer Spezifikation.

Architekturprinzip Modularisierung

➔ Auswirkungen auf Systemteile beschränken.



Problem: Verfeinerung & Modularisierung bewahren **nicht alle** Anforderungen (z.B. Sicherheitsanforderungen) !

Was tun ?

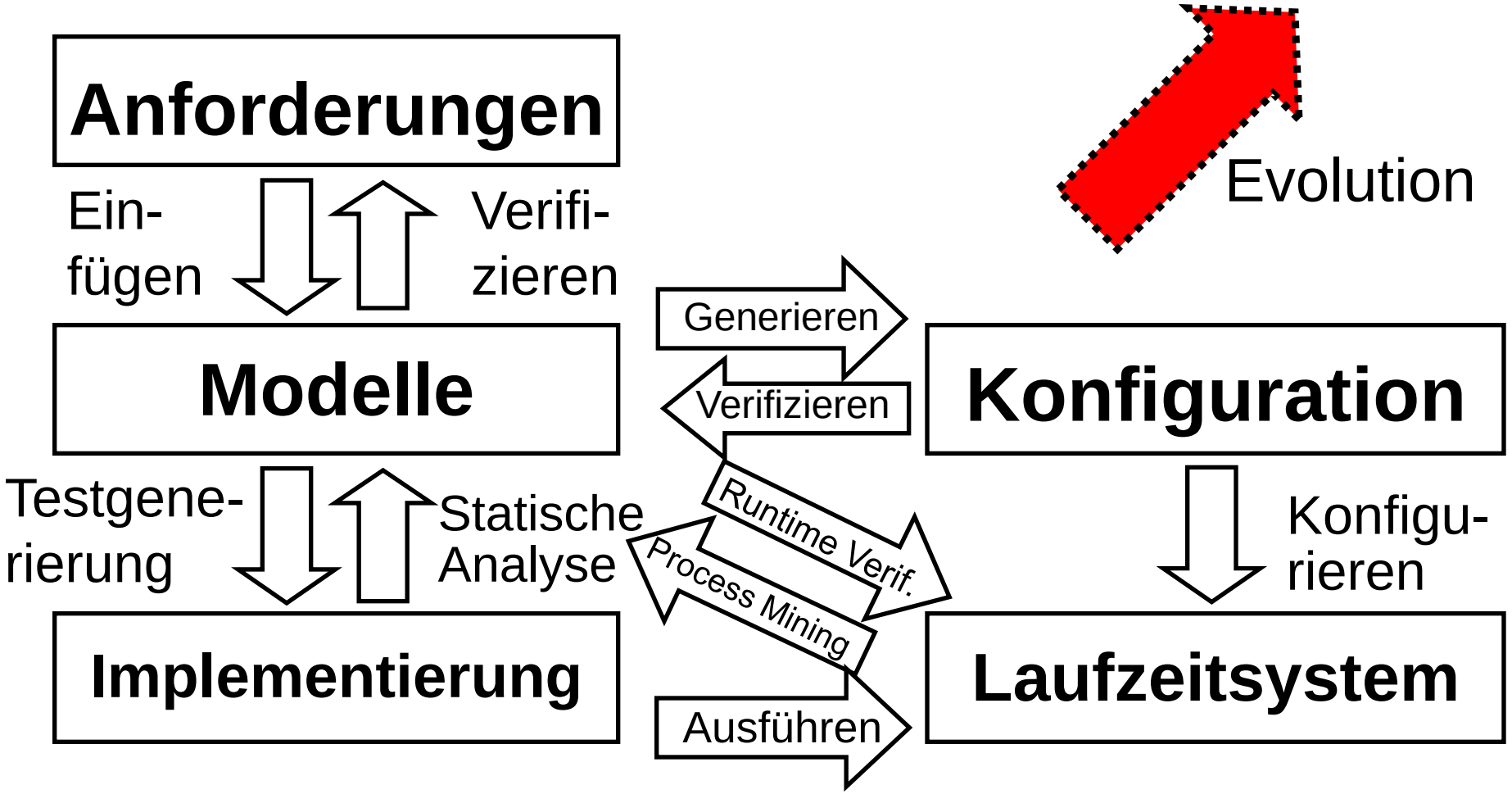
Ziel: QS-Ansatz, der **Evolution** unterstützt:

- Aufwändige QS-Resultate soweit möglich **wiederverwenden**.

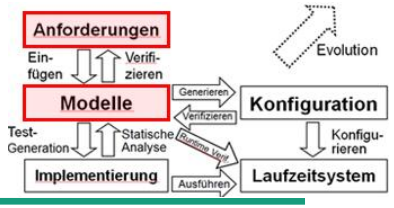
➔ Unter welchen **Bedingungen Anforderungen bewahrt** ?

- Erneute Überprüfung nur, wenn Anforderungen **nicht** bewahrt.
- Nur Software-Teile erneut überprüfen, für die **notwendig**.
- Verschiedene **Evolutions-Alternativen** automatisch auf Auswirkungen auf Anforderungen analysieren.
- Im Voraus: **Architektur** wählen, die bei Evolution Bewahrung kritischer Anforderungen optimal unterstützt.

Modellbasierte Qualitätssicherung als Grundlage



Konkret: Modellbasierte Sicherheit mit UMLsec



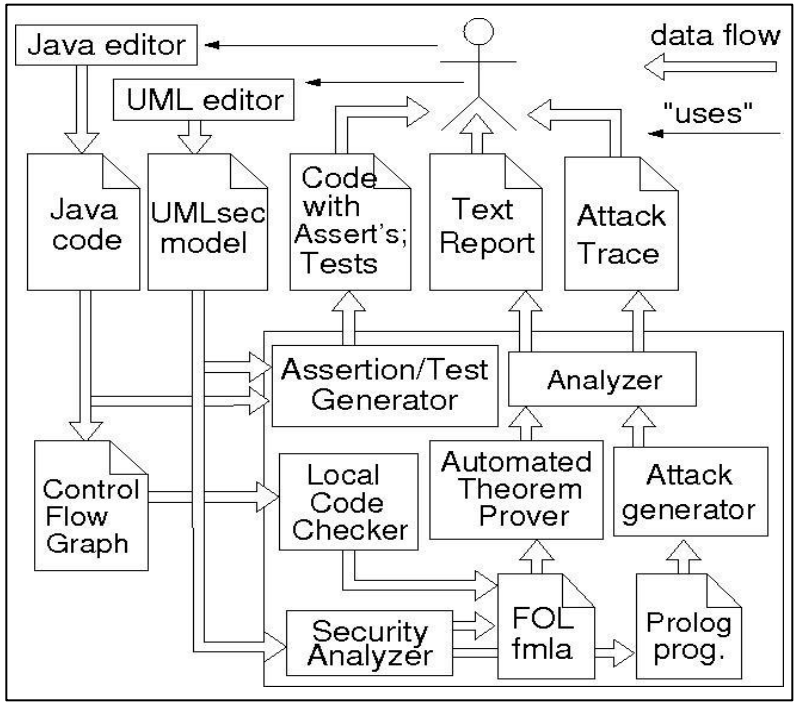
UMLsec: UML-Erweiterung für **Dokumentation** und **automatische Überprüfung** von Sicherheitsanforderungen.

[Jürjens: Secure systems development with UML. Springer 2005. Chines. Übers. 2009]

Anforderungen als **Annotationen** in UML-Modell.

Automatische Überprüfung der Modelle und Implementierungen gegen Anforderungen.

Werkzeugplattform CARiSMA (open-source, integriert in Eclipse).



[http://carisma.umlsec.de]

[Jürjens: Automated security hardening for evolving UML models. ICSE'11]

Modellbasierte Sicherheit in der Praxis: Beispiel-Anwendung

UMLsec-basierte Sicherheitsanalyse bei O_2 (Germany).

62 Anforderungen extrahiert:

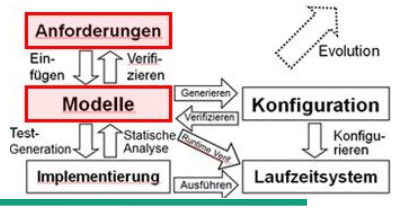
- 21 Geschäftsprozess-relevante Anforderungen
- 15 Anforderungen zu Sicherheit der Netzwerkdienste, Einsatz von Firewalls / Antivirensoftware
- 10 Datensicherheits-Anforderungen
- 3 Anforderungen zu Rollen-basierter Zugangskontrolle

Zu einer davon gleich mehr.

Nr.	Sicherheitsanforderungen	<<Secure Links>>, Secure Links with XML	<<Fair Exchange>>	<<Provable>>	SecrecyIntegrity	TPTP-Datei z
1.9	Authentifizierung des Benutzers (Mitarbeiters) gegenüber dem Endgerät durch Chipkarten			X		
1.10	Verschlüsselung der auf den mobilen Endgeräten befindlichen Daten	X				
1.14	Keine zum Fernzugang parallele Verbindungen in andere Netze - Vermeidung der Kopplung mit unsicheren Netzen durch Umgehung der Firewall					
1.26	Starke Verschlüsselung der Verbindung zwischen Endgerät und Fernzugangs-Server	X				
1.37	Bei O_2 übliche Virenschutzprogramme auf den Endgeräten					
1.38	Aktualisierung des Virenschutzes über Fernzugang					

[Jürjens, Schreck, Bartmann. Model-based security analysis for mobile communications. ICSE '08]

Modellbasierte Sicherheit in der Praxis: Beispiel-Anforderung

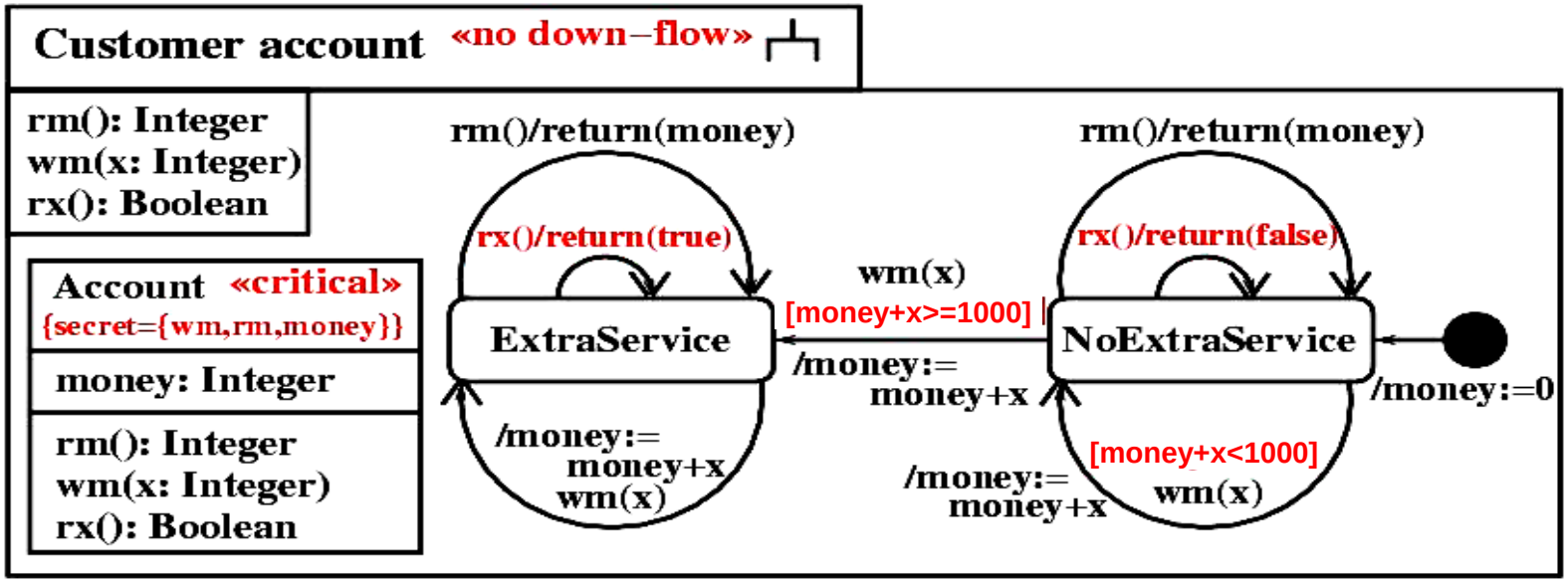


Beispiel „sicherer Informationsfluss“:

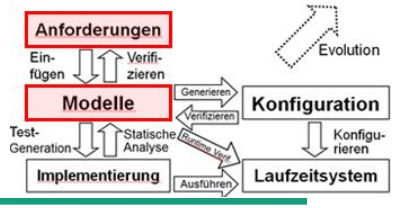
Kein Informationsfluss von vertraulichem zu öffentlichem Wert.

Analyse: Wenn zwei Systemzustände sich nur in Werten der vertraulichen Attribute unterscheiden, darf öffentlich beobachtbares Verhalten sich nicht unterscheiden.

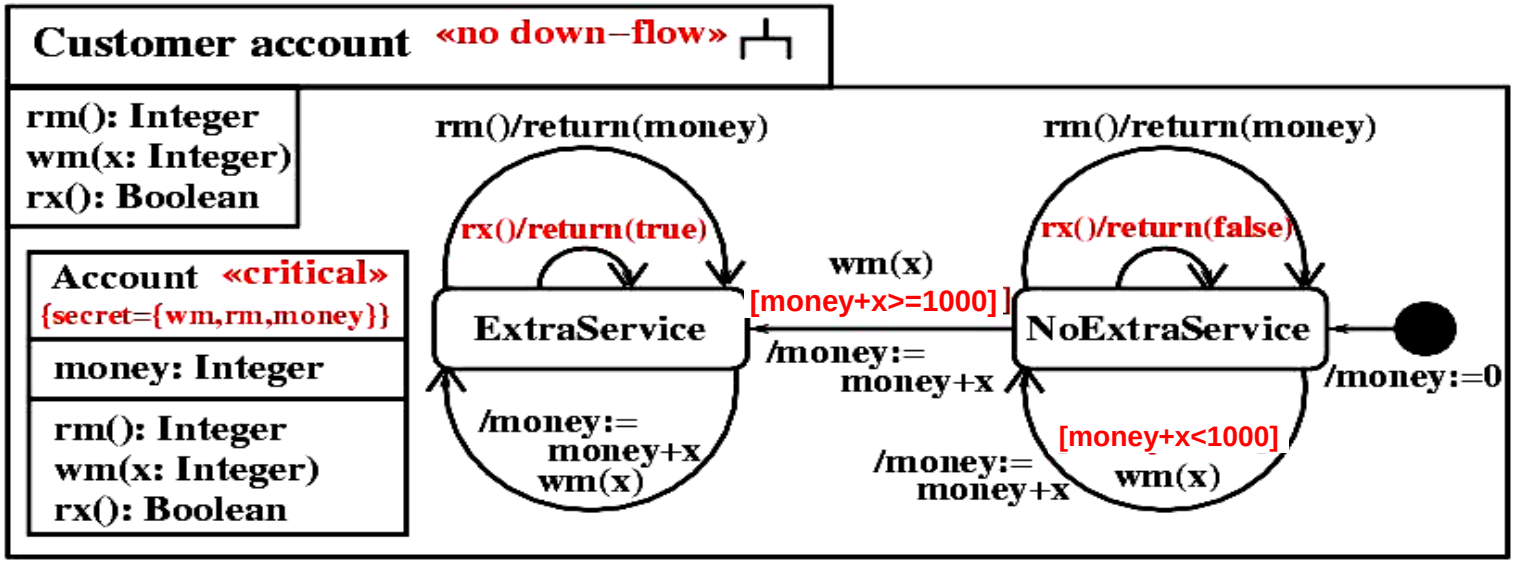
Beispiel: Web-basiertes Kundenkonto. Sicher ?



Modellbasierte Sicherheit in der Praxis: Beispiel „sicherer Informationsfluss“



Unsicher: vertrauliches Attribut *money* beeinflusst Rückgabewert der öffentlichen Methode *rx()*.

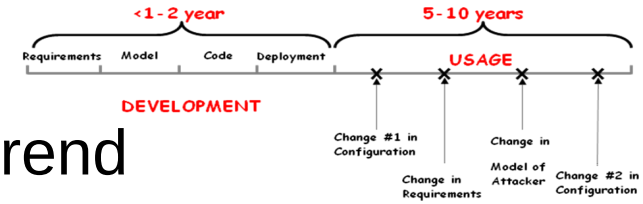


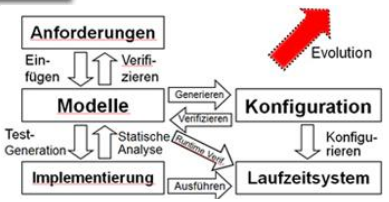
Wie kann man diese Anforderung **automatisch** und in Gegenwart von **Evolution verifizieren** ?

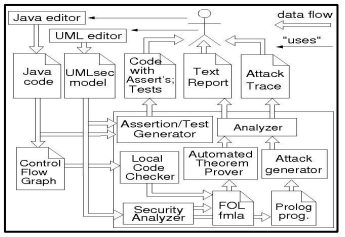
Einführung: Zusammenfassung

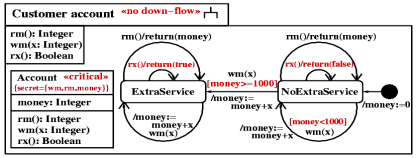
Worum ging es in dieser Einführung ?

- **Software-Evolution**: Änderungen während Softwarelebenszyklus
- Schwierig: **Bewahrung kritischer Anforderungen** (Beispiel: Ariane 5)
- **Modellbasierte Entwicklung**:
  - Nachverfolgbarkeit der Anforderungen durch Softwarelebenszyklus
  - Werkzeuggestützte Überprüfung der Anforderungen
- Beispiel sicherer Informationsfluss: Bei **Evolution automatisch verifizieren** ?

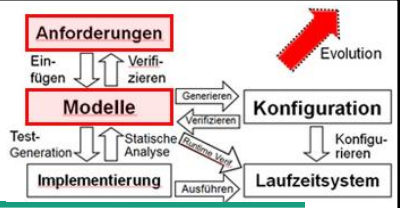






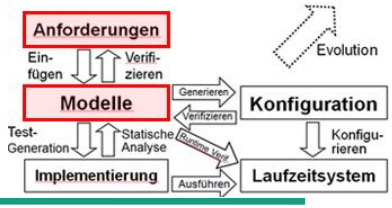


Sichere Evolution für verteilte Software-Systeme



- Einführung
  - Evolution, Modellbasierte Qualitätssicherung
- Sichere Evolution für verteilte Software-Systeme
  - Evolution auf Entwurfs-Ebene
  - Evolution auf Implementierungs-Ebene
- Validierung

Sichere Evolution auf Modellebene: Analyse durch Formalisierung



Ziel: Automatische Überprüfung von Anforderungen.

Beispiel: UML-Statechart.

Formalisierung der Modellausführung. Gegeben: Statechart-Transition *t=(source,msg,cond[msg],action[msg],target)* und erhaltene Nachricht *m*.
Formalisierung der Ausführung der Transition:

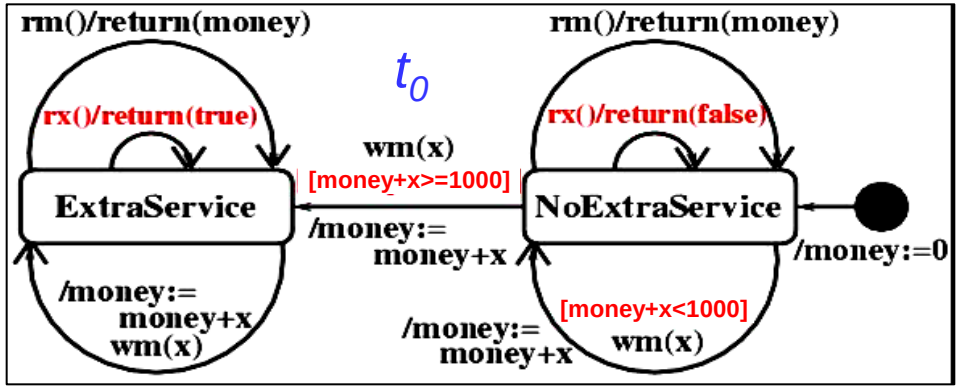
$$Exec(t,m) = [\ state_{current}=source \wedge m=msg \wedge cond[m]=true \Rightarrow action[m] \wedge state_{current.t(m)}=target\ ].$$

(wobei $state_{current}$ aktueller Zustand; $state_{current.t(m)}$ Zustand nach Ausführung von *t* mit Nachricht *m*).

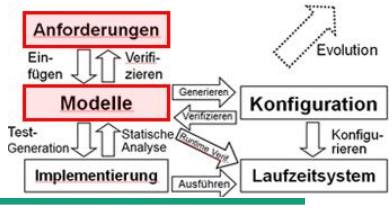
[Ochoa, Jürjens, Cuellar. Non-interference on UML State-charts. TOOLS Europe '12]

Beispiel: Transition t_0:

$$Exec(t_0,m)= [\ state_{current}=NoExtraService \wedge m=wm(x) \wedge money_{current}+x>=1000 \Rightarrow money_{current.t_0(m)}=money_{current}+x \wedge state_{current.t_0(m)}=ExtraService\ ].$$



Beispiel sicherer Informationsfluss: Automatische Verifikation



Angenommen Systemzustände $state_{current}$, $state'_{current}$ unterscheiden sich nur in vertraulichen Attributen. Dann darf auch zukünftig öffentlich beobachtbares Verhalten sich nicht unterscheiden:

$$state_{current} \approx_{pub} state'_{current} \Rightarrow state_{current.t(m)} \approx_{pub} state'_{current.t(m)}$$

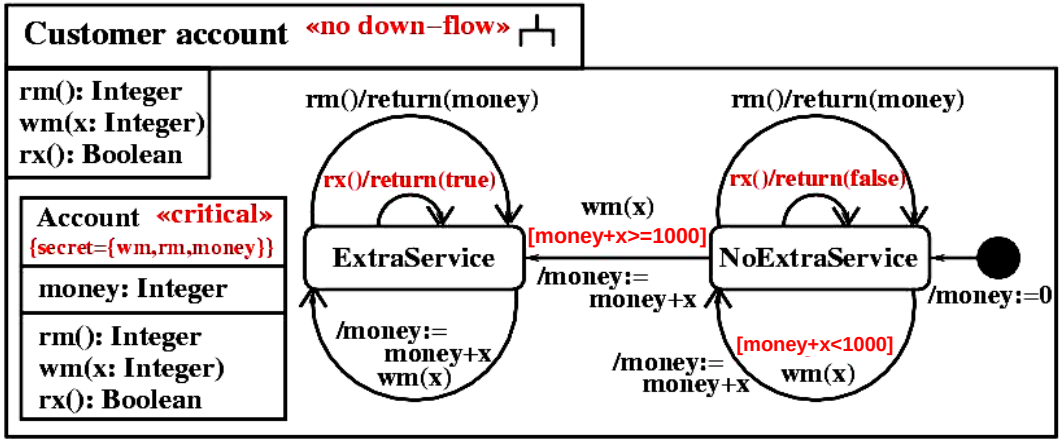
($state_{current} \approx_{pub} state'_{current}$: gleiches öffentlich beobachtbares Verhalten; $state_{current.t(m)}$: nächster Zustand)

Beispiel: Unsicher: vertrauliches Attribut *money* beeinflusst Rückgabewert der öffentlichen Methode *rx()*.

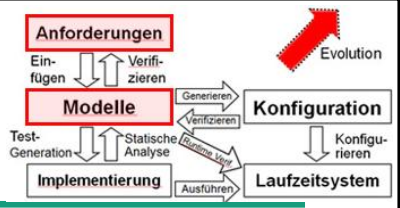
$ExtraService \approx_{pub} NoExtraService$

aber nicht:

$ExtraService.rx() \approx_{pub} NoExtraService.rx()$



Evolution vs. Design- / Architekturprinzipien: Resultate



Unter welchen Voraussetzungen bewahren Verfeinerung und Modularisierung Anforderungen ?

Verfeinerung: Verfeinerungsansatz entwickelt, der Sicherheitsanforderungen **bewahrt**.

[Schmidt, Jürjens: Connecting Security Requirements Analysis and Secure Design Using Patterns and UMLsec. CAiSE'11]

Modularisierung: z.B.:

- Schichtung von **Architekturebenen**
- **Komponenten**-orientierte Architekturen

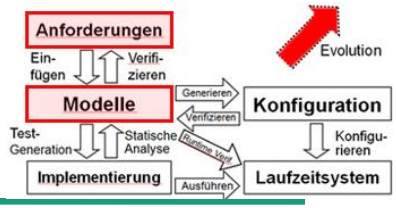
[Hatebur, Heisel, Jürjens, Schmidt: Systematic Development of UMLsec Design Models Based on Security Requirements. FASE'11]
[Ruhroth, Jürjens: Supporting Security Assurance in the Context of Evolution: Modular Modeling and Analysis with UMLsec. HASE'12]

[Ochoa, Jürjens, Warzecha: A Sound Decision Procedure for the Compositionality of Secrecy. ESSoS'12]

Bedingungen für Bewahrung von Anforderungen identifiziert.

Im Folgenden am Beispiel von Sicherheitsanforderungen.

Sicherheit vs. Verfeinerung: Problem

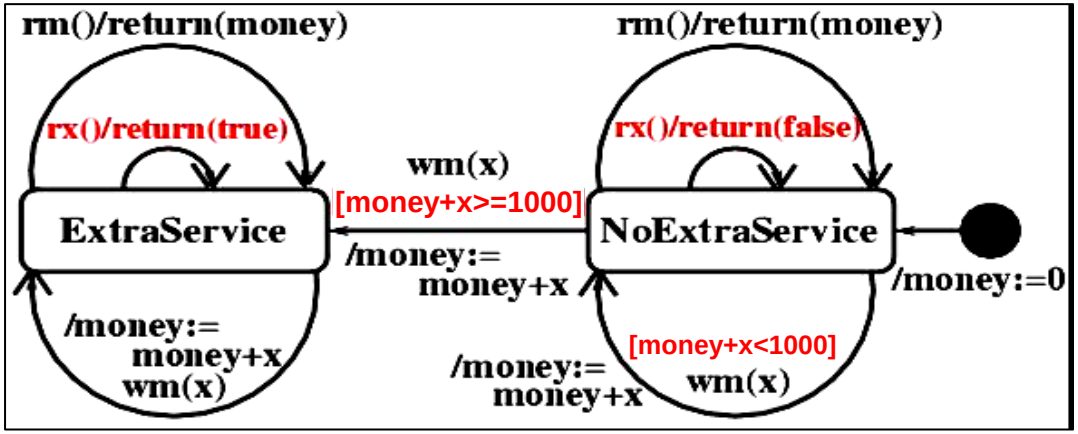


Verhaltensbewahrende Verfeinerung: würde Bewahrung von Sicherheitsanforderungen erwarten.

„**Verfeinerungsparadox**“: Im Allgemeinen jedoch nicht ! [Roscoe‘96].

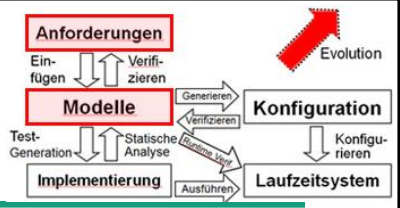
Obiges Beispiel: Transitionen *rx()/return(true)* (bzw. *false*)

Verfeinerung der „sicheren“ Transition *rx()/return(random_bool)*.



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

Sicherheit vs. Verfeinerung: Lösung



Lösung: Unser Spezifikationsansatz trennt beide Arten von Nicht-Determinismus.

Resultat: Verfeinerung bewahrt wichtige Sicherheitseigenschaften.

Beispiel:

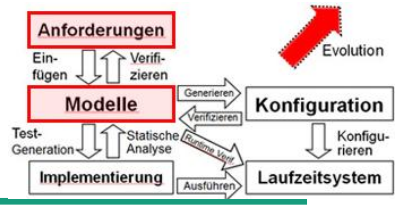
Definition Q refines P ($P \rightsquigarrow Q$) if for each $\vec{s} \in \mathbf{Stream}_{I_P}$ have $[\![P]\!](\vec{s}) \supseteq [\![Q]\!](\vec{s})$.

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

Nachweis: mit formaler Semantik.

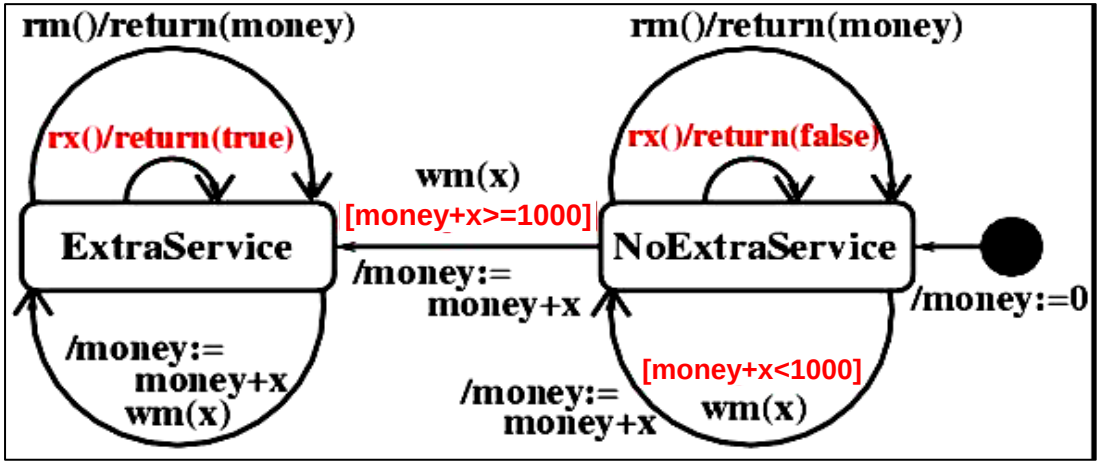
Obiges Beispiel: mit unserem Ansatz **keine** Verfeinerung.

Sicherheits vs. Modularisierung: Problem



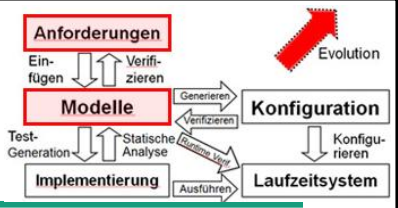
Problem: Sicherheitseigenschaften i.A. nicht kompositional.

Obiges Beispiel: Zustände *ExtraService* und *NoExtraService* jeweils „sicher“ (nur ein Rückgabewert von *rx*); Komposition in Statechart nicht.



Frage: Unter welcher Bedingung bewahrt Komposition Sicherheit ?

Sicherheits vs. Modularisierung: Lösung



Lösungsansatz: Sicherheitsanforderung als „Rely-guarantee“[1]-Bedingung definieren.

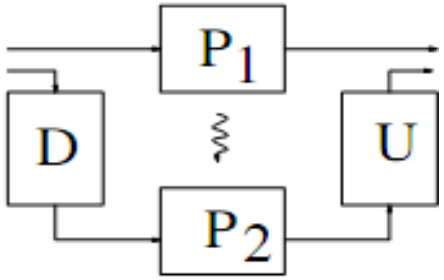
Resultat: Dafür Bedingungen an Kompositionalität.

Beispiel-Resultat:

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.

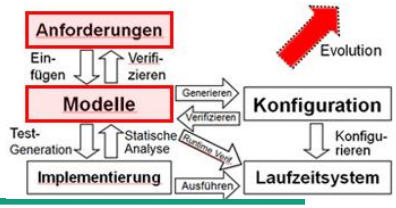
Nachweis: mit formaler Semantik.

Obiges Beispiel: Rely-guarantee-Formalisierung zeigt: Sichere Komposition nicht möglich.



[1] C.B. Jones 1981

Differenz-basierte Verifikation auf Design-Ebene



Differenz-basierte Verifikation:

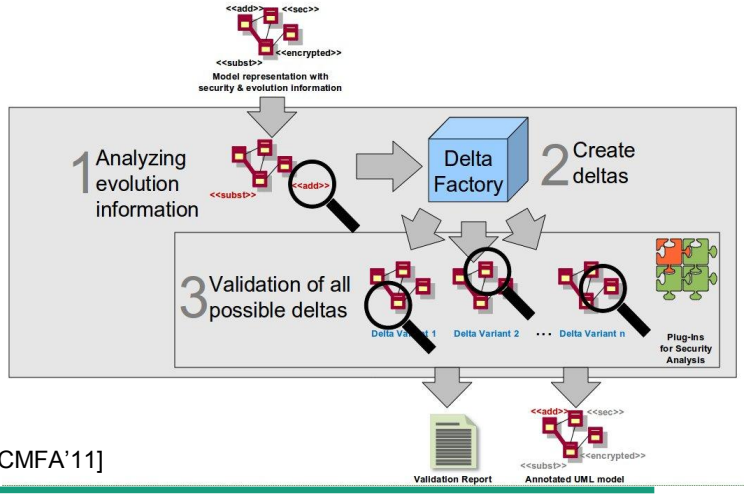
- Erstmalige Analyse: Registrieren, welche **Modellelemente relevant**.
- Teilresultate in Modell **speichern** („proof-carrying models“).

Theorem 1 *Assume that the program p' evolved from the program p where p and p' are related as in the following cases*
p = either p' or p'': This implies $p \succcurlyeq p'$ and $p \succcurlyeq p''$.
*p = if $E = E'$ then p' else p'': For any expression $X \in$ **Exp** such that p preserves the secrecy of X: p' preserves the secrecy of X assuming $E = E'$ and p'' preserves the secrecy of X assuming $E \neq E'$.*

...

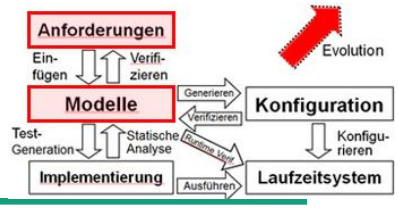
- **Differenz**: altes zu neues Modell berechnen (z.B. mit SiDiff [Kelter]).
- Obige Resultate für **sicherheitsbewahrende Änderungen** verwenden.
- Nur die **Modellteile reverifizieren**, die
  1) in der initialen Analyse relevant,
  2) **geändert** wurden, sodass
  3) Änderung o.g. Bedingungen **nicht erfüllt**.

➔ Erheblich weniger Aufwand als komplette Reverifikation.



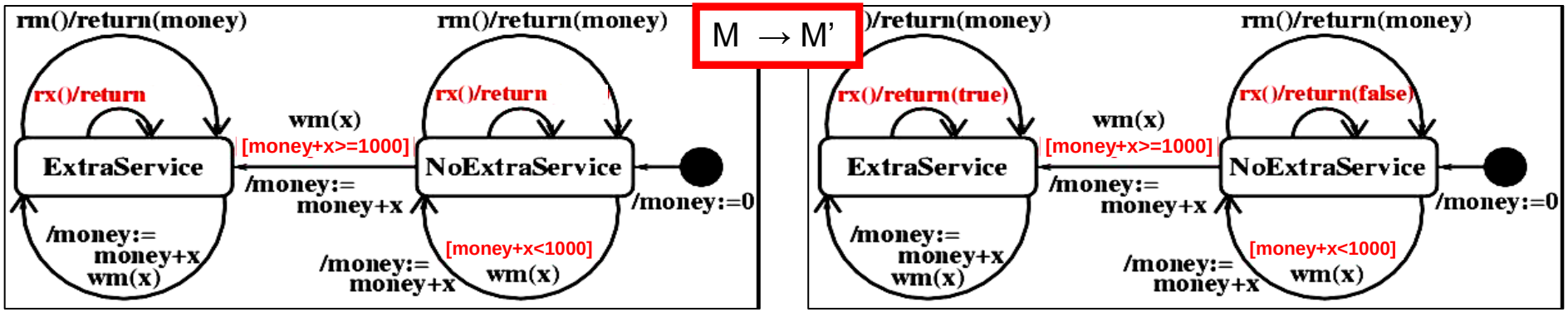
[Jürjens, Marchal, Ochoa, Schmidt: Incremental Security Verification for Evolving UMLsec models. ECMFA'11]

Evolutions-basierte Verifikation: Beispiel



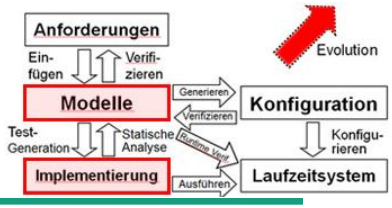
Beispiel: Heuristik zu sicherem Informationsfluss bei Evolution M → M‘: Nur Systemzustände *s*, *s‘* betrachten, für die:

- $s \approx_{pub} s'$ in M‘ aber nicht in M, oder
- $s.t(m) \approx_{pub} s'.t(m)$ in M aber nicht in M‘.



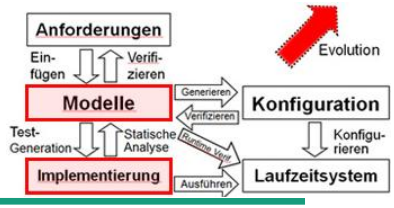
Beispiel: $wm(0).rx() \approx_{pub} wm(1000).rx()$ in M aber nicht in M‘. Impliziert, dass M‘ **nicht sicheren Informationsfluss erfüllt** (vertrauliche Argumente *0* und *1000* unterscheidbar).

Sichere Evolution für verteilte Software-Systeme



- Einführung:
  - Evolution, Modellbasierte Qualitätssicherung
- Sichere Evolution für verteilte Software-Systeme
  - Evolution auf Entwurfs-Ebene
  - Evolution auf Implementierungs-Ebene
- Validierung

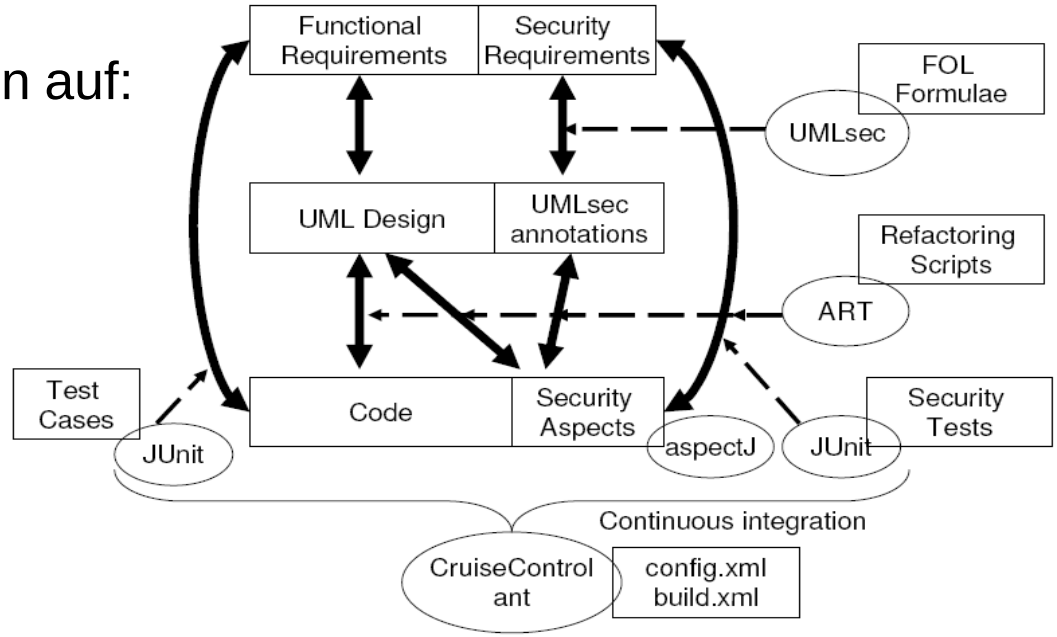
Verbindung von Modell und Implementierung bei Evolution



Ziel: Nachverfolgbarkeit von **Anforderungen** vs. **Implementierung** bei **Evolution**.

Lösung: Änderungen reduzieren auf:

- Hinzufügen / Entfernen von Systemteilen.
- Grundlegende Refactoring-Operationen.



➔ **Automatische Nachverfolgbarkeit** der Änderungen zwischen Modell und Implementierung mit **Refaktoring Scripts** (Eclipse).

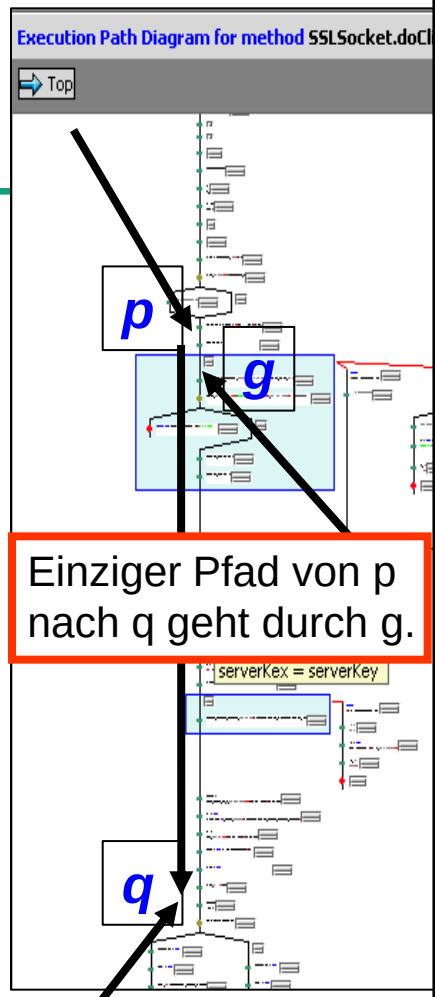
[Bauer, Jürjens, Yu: Run-Time Security Traceability for Evolving Systems. Computer Journal 2011]

Evolutions-basierte Verifikation auf Implementierungs-Ebene

Evolutions-basierte Verifikation der Implementierung:

=> **Automatische statische Analyse der Implementierung** gegenüber Modell (z.B. Bedingungen in Sequenz-diagramm korrekt in Implementierung umgesetzt).

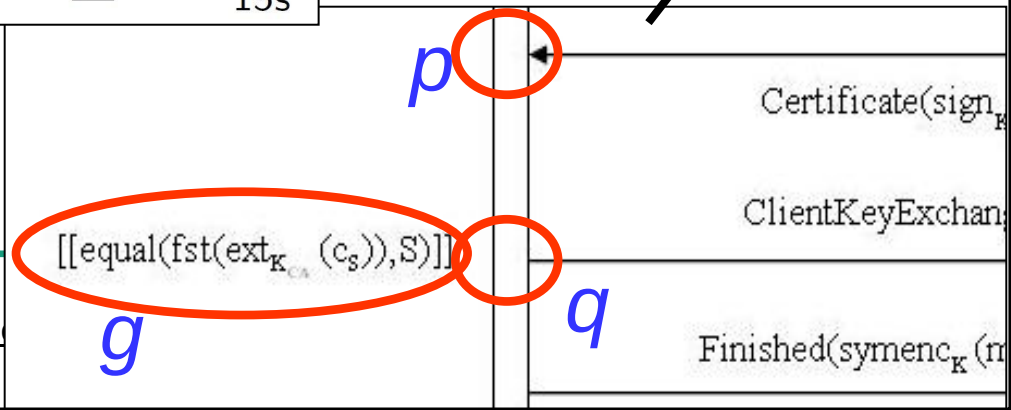
Projekt Csec mit Microsoft Research Cambridge: Werkzeug erfolgreich eingesetzt, mehrere Schwachstellen gefunden.



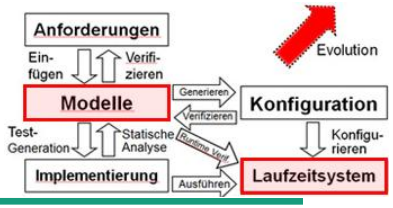
Einziger Pfad von p nach q geht durch g.

	C LOC	IML LOC	outcome	result type	time
simple mac	~ 250	12	verified	symbolic	4s
RPC	~ 600	35	verified	symbolic	5s
NSL	~ 450	40	verified	computat.	5s
CSur	~ 600	20	flaws found	—	5s
Metering	~ 1000	51	flaws found	—	15s

[Aizatulin, Gordon, Jürjens: Extracting and verifying cryptographic models from C protocol code by symbolic execution. CCS'11]
[A., G., J.: Computational Verification of C Protocol Implementations by Symbolic Execution. CCS'12]
[Dupressoir, Gordon, Jürjens, Naumann: Guiding a General-Purpose C Verifier to Prove Cryptographic Protocols. CSF'11]



Evolutions-basierte Laufzeitverifikation



Quellcode nicht verfügbar => Laufzeitüberwachung.

Möglicher Ansatz für Monitoring: Security Automata [F.B. Schneider 2000].

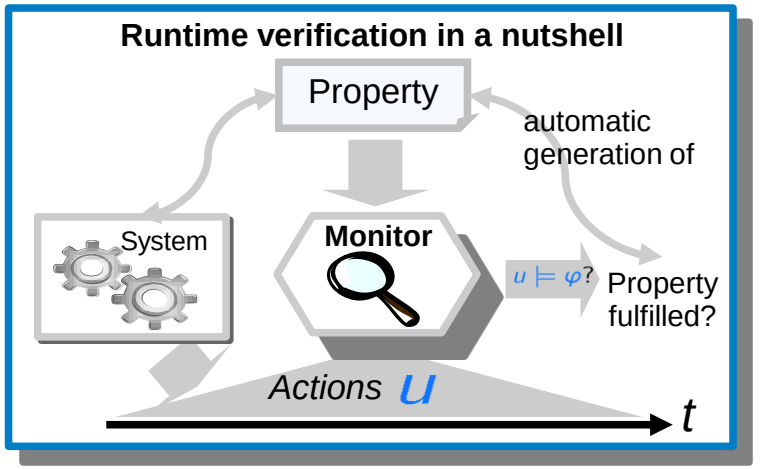
Problem: keine Änderungen; nur **„Safety“-Eigenschaften**.

Neuer Ansatz auf Basis von Runtime-Verification[1] (abgeleitet aus Model-Checking und Testen).

Sicherheitsanforderung in LTL.

Generierung von Monitoren; **automatische Anpassung bei Änderungen**.

Auch **Nicht-Safety-Eigenschaften** (3-wertige LTL-Semantik).

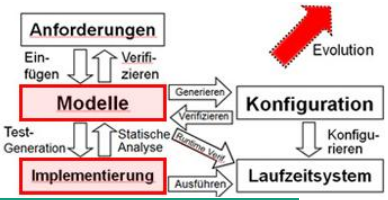


[Bauer, Jürjens. Runtime Verification of Cryptographic Protocols. Computers & Security ‘10]
[Pironti, Jürjens. Formally-Based Black-Box Monitoring of Security Protocols. ESSOS’10]

[1] Havelund, Grosu 2002

Client	Server	No Monitor [s]	Monitor [s]	Overhead [s]	Overhead [%]
GnuTLS	GnuTLS	0.109	0.120	0.011	10.313
OpenSSL	JESSIE	0.158	0.172	0.014	8.986
GnuTLS	JESSIE	0.144	0.148	0.004	2.788

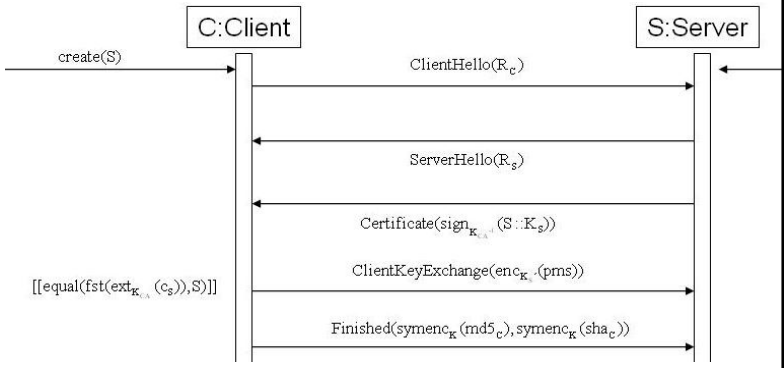
Anwendung: Java Secure Sockets Extension



Anwendung: Verschiedene Versionen der Java-Bibliothek "Java Secure Sockets Extension (JSSE)" und open-source Re-Implementierung (Jessie).

Mehrere Schwachstellen identifiziert.

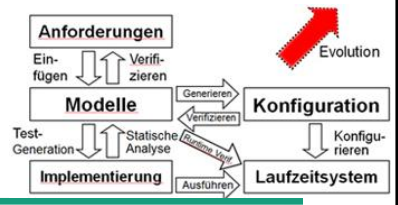
Funktioniert also auch für **größere Systemänderungen** !



Symbols	Program entities	Identif.	Refactoring op.
1. C	clientHello	C	rename.type
2. S	serverHello	S	rename.type
3. P_{ver}	session.protocol version	P_ver	extract.temp
4. R_C R_S	clientRandom serverRandom	R_C R_S	rename.local.variable rename.local.variable
5. S_{id}	sessionId	S_id	rename.field

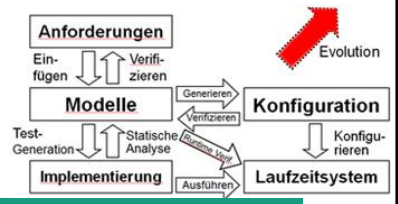
Messages in sequence	op.	diff	Time (sec)
S1: $C \rightarrow S : (P_{\text{ver}}, R_C, S_{\text{id}}, \text{Ciph}[\,], \text{Comp}[\,])$	7	31	13.891
S2. $S \rightarrow C : (P_{\text{ver}}, R_S, S_{\text{id}}, \text{Ciph}[\,], \text{Comp}[\,])$	5	20	9.437
S3. $S \rightarrow C : \text{Certificate}[\text{X509Cert}_s]$	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

Sichere Evolution für verteilte Software-Systeme



- Einführung: 
  - Evolution, Modellbasierte Qualitätssicherung
- Sichere Evolution für verteilte Software-Systeme 
  - Evolution auf Entwurfs-Ebene
  - Evolution auf Implementierungs-Ebene
- Validierung

Technische Validierung



- **Korrektheit**: mittels formaler Semantik
- **Vollständigkeit**: jede Modell-Transformation darstellbar als Folge von Löschen, Ändern und Hinzufügen von Modell-Elementen

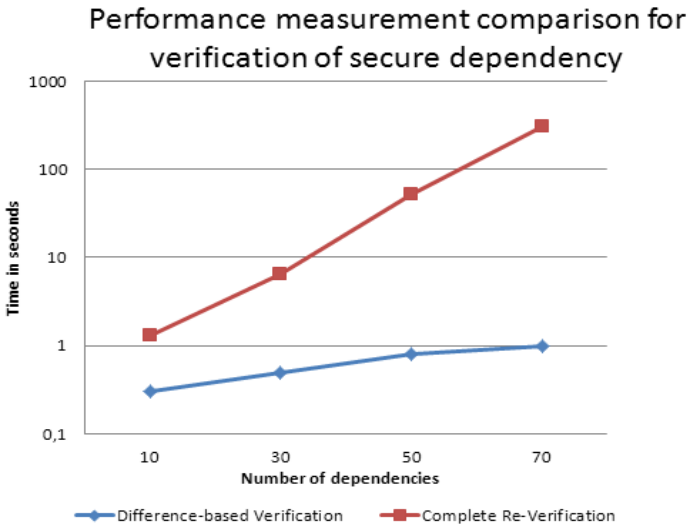
Performanzgewinn am **größten**, wenn **Differenz << Software.** Beispiel-Resultat:

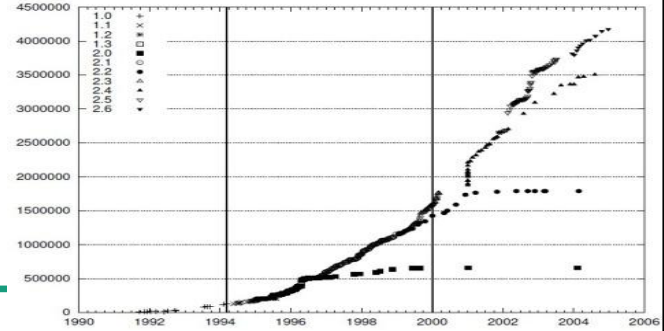
- Evolutions-basierte Verifikation: Laufzeit **linear** in Softwaregröße
- Komplette Re-Verifikation: Laufzeit **exponentiell** in Softwaregröße

(jeweils bei gleichbleibender Differenzgröße)

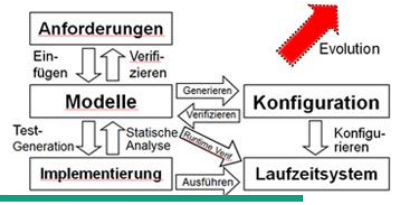
Ist erfüllt:

- bei **Wartung stabiler Software-Versionen**
- bei **änderungsbegleitender QS** (z.B. nightly builds)





Validierung im Praxiskontext



Praktische Anwendungen des UMLsec-Ansatzes (Auswahl):

- Global Platform (Smartcard-Software-Updates, Gemalto)
- Biometrisches Authentisierungssystem
- Gesundheitsinformationssysteme
- ...

[Jürjens et al.: Incremental Security Verification for Evolving UMLsec models. ECMFA'11]
[Lloyd, J. Jürjens, Security Analysis of a Biometric Authentication System using UMLsec and JML. Models'09]
[Mouratidis, Sunyaev, Jürjens: Secure Information Systems Engineering: Experiences and Lessons Learned from Two Health Care Projects. CAiSE'09]

Bei einigen Anwendungen signifikante Schwachstellen identifiziert.

Empirischer Vergleich modellbasierte Sicherheitszertifizierung mit klassischen Ansätze (z.B. Testen):

- Studie anhand Türsteuergerät (Kooperation BMW).
- Modellbasiert: Zusatz-Aufwand, aber findet auch obskure Schwachstellen.

[Jürjens, Trachtenherz, Reiss: Model-based Quality Assurance of Automotive Software. Models'08]

Modellbasierte Qualitätssicherung für sichere Software: Überblick über das Forschungsfeld (Auswahl)

2001: UMLsec: UML profile for security modelling (Jürjens)

2002: Model-based Security with UMLsec (Jürjens): **10 Year Most Influential Paper (Models'12)**
Secure UML: Modelling RBAC with UML (Basin et al.)
Aspect-oriented Security Modelling (France et al.)
Model-based IT security risk assessment (Stølen et al.)
Interactive theorem proving of UML models for security (Haneberg, Reif et al.)

2003: Formal verification for UML models of access control (Koch, Parisi-Presicce)

2004: Actor-centric modeling of user rights with UML (Breu et al.)
Extending OCL for secure database development (Fernández-Medina et al.)

2005: UMLsec Buch (Jürjens)
Model-based Security, "Build Security In", (**US Dep. Homeland Sec.)**
Controlled Evolution of Access Rules in Coop. Information Systems (Rinderle, Reichert)

2007: "Model-Driven Security: Enabling a Real-Time, Adaptive Security Infrastructure" (**Gartner-Report**):
"Model-driven security is embryonic, but it will have a significant impact as information security infrastructures become increasingly real time, automated, and adaptive to changes in organizations and their environments."

2007: Security monitors for UML policy models (Massacci et al.)

2008: Executable misuse cases for security concerns (Whittle et al.)

2009: UMLsec Buch: Übersetzung ins Chinesische (Jürjens)

2010: Conceptual Business Process Security Modeling (Riesner, Pernul)

2011: Modeling process-related RBAC models with extended UML activity models (Strembeck et al.)

2012: Towards Model-Driven Development of Access Control Policies for Web Applications (N. Koch et al.)

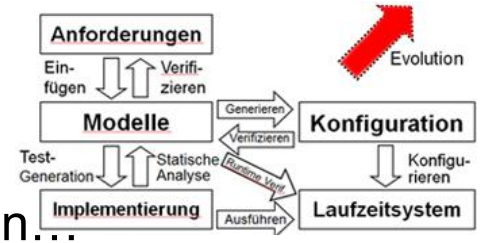
…

Zusammenfassung:
Sichere Evolution für verteilte Software-Systeme

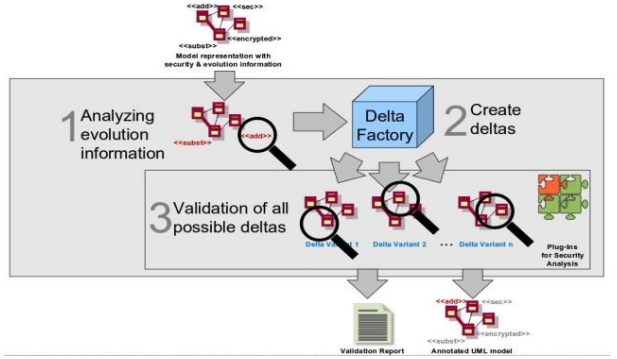
Evolution: Herausforderung für Qualitätssicherung.

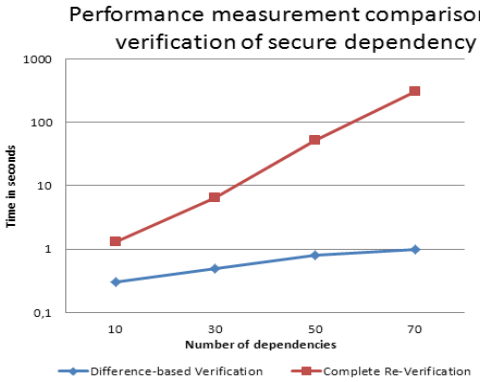
Frage: Wann QS-Ergebnisse wiederverwenden ?

Resultate: Bedingungen für Bewahrung von Anforderungen…



- … durch Design-/Architektur-Prinzipien (z.B. **Verfeinerung**, **Modularisierung**).
- Damit **Evolutions-basierte Verifikation**:
  - **Modellebene**
  - **statische Analyse** / **Run-time Verification**.
- Implementiert in **Werkzeugen**: erhebliche Performanz- und Skalisierungsverbesserung.





Validierung: Erfolgreicher Einsatz in der Praxis.

Aktuell: Migration in Clouds