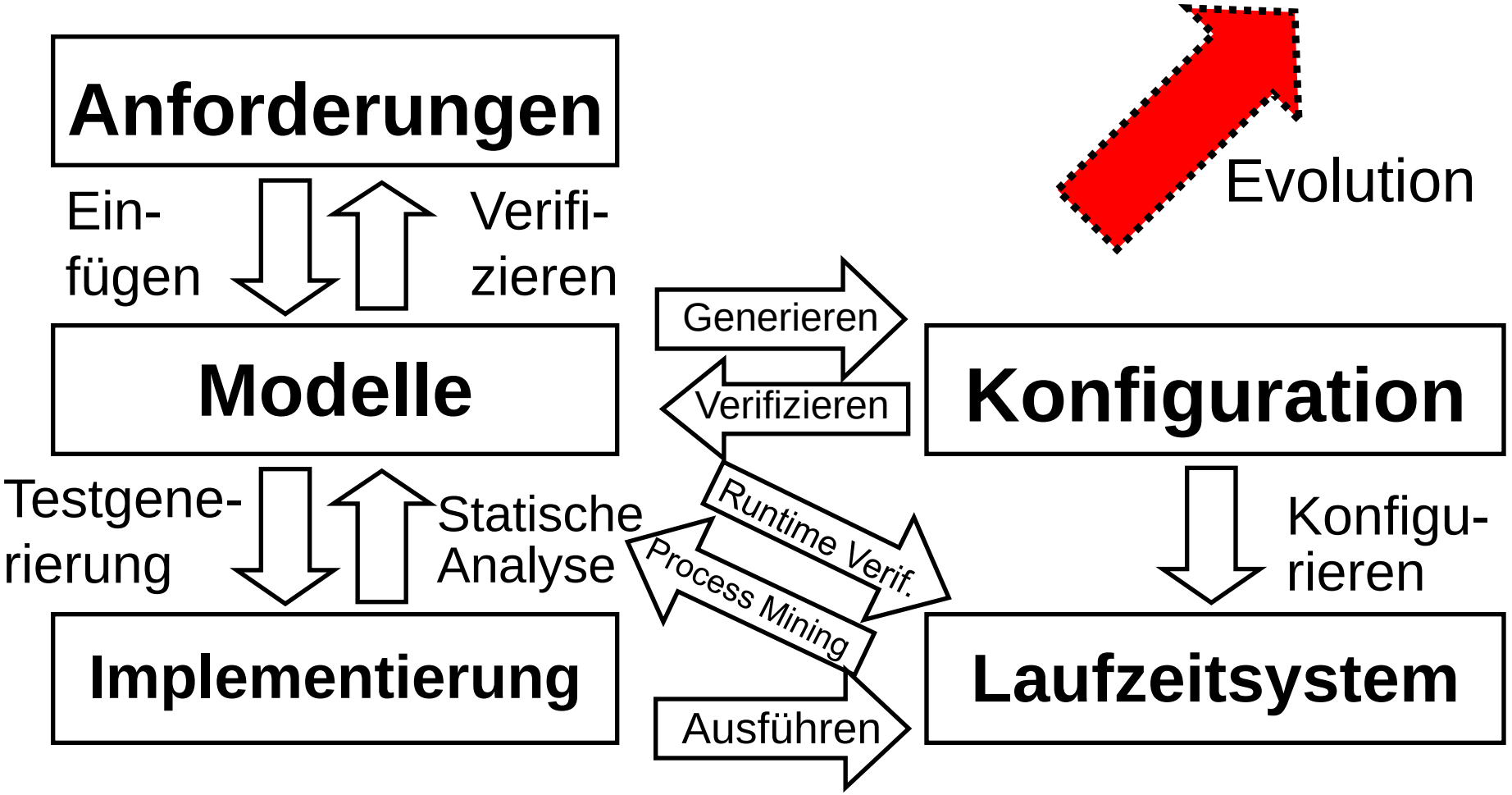
# Evolution vs. semantische Konsistenz – Einige Resultate

**Jan Jürjens**

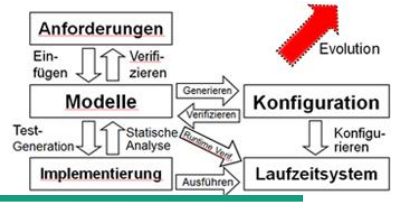
TU Dortmund

http://jan.jurjens.de

# Ansatz: Modellbasierte Qualitätssicherung



# Modellbasierte Qualitätssicherung vs. Evolution




Ziel: Vorhandene QS-Resultate soweit möglich wiederverwenden.

=> Unter welchen **Bedingungen bewahrt Evolution Anforderungen** ?

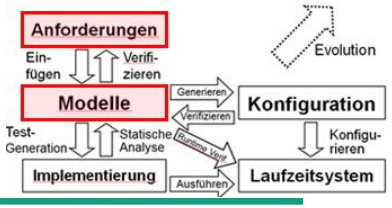
Bei Antwort auf diese Frage:

- Erneute Überprüfung nur benötigt, wenn Voraussetzungen für Bewahrung der Anforderungen unter Evolution nicht gegeben.
- Nur die Software-Teile erneut überprüfen, für die dies notwendig.

**Noch besser:** Evolutions-Alternativen automatisch auf Auswirkungen auf Anforderungen analysieren.

**Im Voraus:** Auswahl von Architekturalternativen, die bei zukünftiger Evolution die Bewahrung kritischer Anforderungen optimal unterstützen.

# Modellbasierte QS mit UMLsec




Konkreter Kontext für Evolutions-Resultate: **UMLsec**.

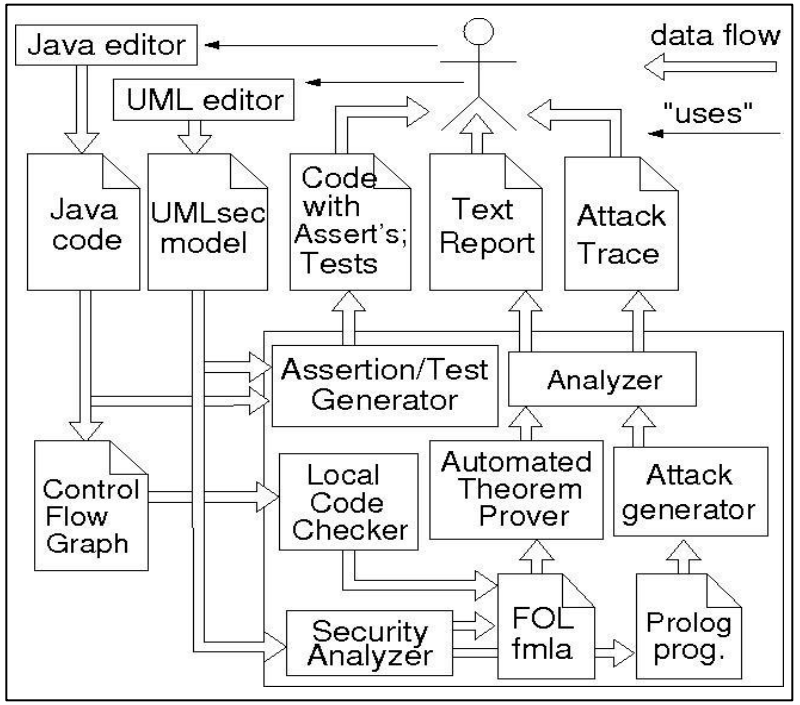
UML-Erweiterung für **Dokumentation** und **automatische Überprüfung** von kritischen Anforderungen (insbes. Sicherheits-Anforderungen).

**Idee:** Kritische Anforderungen als Markierungen (Stereotypen) in UML-Modell.

**Automatische Überprüfung** der Modelle und Implementierungen gegen Anforderungen.

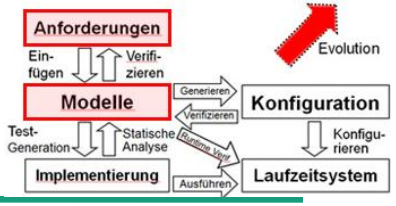
**Werkzeugplattform** CARiSMA (open-source, integriert in Eclipse).


[http://carisma.umlsec.de] [Jürjens: Secure systems development with UML. Springer 2005. Chines. Übers. 2009]




[Jürjens, Sound Methods and Effective Tools for Model-based Security Engineering with UML, ICSE'05]
[Höhn, Jürjens: Rubacon: automated support for model-based compliance engineering. ICSE '08]

# Evolution vs. Design- / Architekturprinzipien




Betrachte Designtechniken und Architekturprinzipien, die Management von Evolution unterstützen.

**Frage**: Unter welchen Voraussetzungen bewahren sie Anforderungen ?

**Designtechnik**: **Verfeinerung von Spezifikationen.** Unterstützt Evolution zwischen Verfeinerungen einer abstrakten Spezifikation.

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

**Architekturprinzipien**: **Modularisierung** unterstützt Evolution, indem Auswirkungen auf Systemteile beschränkt bleiben. Verschiedene Dimensionen:

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

[Hatebur, Heisel, Jürjens, Schmidt: Systematic Development of UMLsec Design Models Based on Security Requirements. FASE'11]

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

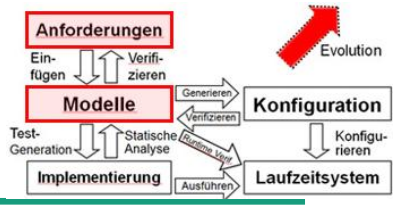
[Deubler, Grünbauer, Jürjens, Wimmel: Sound development of secure service-based systems. ICSOC'04]

[Jürjens, Houmb: Dynamic Secure Aspect Modeling with UML. MoDELS'05]

Jeweils Resultate zu Bedingungen für Bewahrung von Anforderungen.

Im Folgenden am Beispiel von Sicherheitsanforderungen.

# Designtechnik: Verfeinerung




Bei verhaltensbewahrender Verfeinerung würde man Bewahrung von Verhaltens-Anforderungen erwarten.

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

**Beobachtung**: Problematisch: Vermischung von Nichtdeterminismus zur Unterspezifikation bzw. als Sicherheitsmechanismus.
Unser Spezifikationsansatz trennt beide Aspekte.

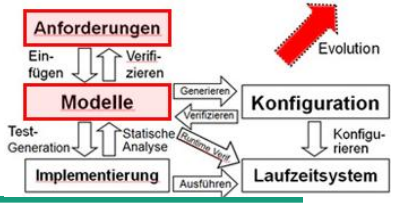
**Resultat**: Verfeinerung bewahrt wichtige Verhaltensanforderungen.

**Nachweis:** mittels formaler Semantik.

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

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

# Architekturprinzip: Modularisierung




**Problem**: Verhaltensanforderungen i.A. nicht kompositional.

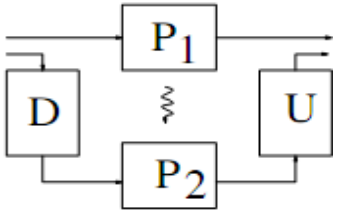
**Frage**: Unter welcher Bedingung bewahrt Komposition Anforderungen ?

**Lösungsansatz**: Anforderung als „Rely-guarantee“-Bedingung definieren.

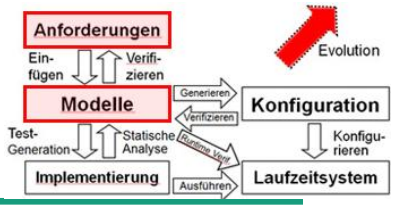
**Resultat**: Für so definierte Anforderungen Bedingungen für Kompositionalität.

**Nachweis:** mittels formaler Semantik.

**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 $m$.*



# Evolution auf Design-Ebene: Differenz-basierte Verifikation

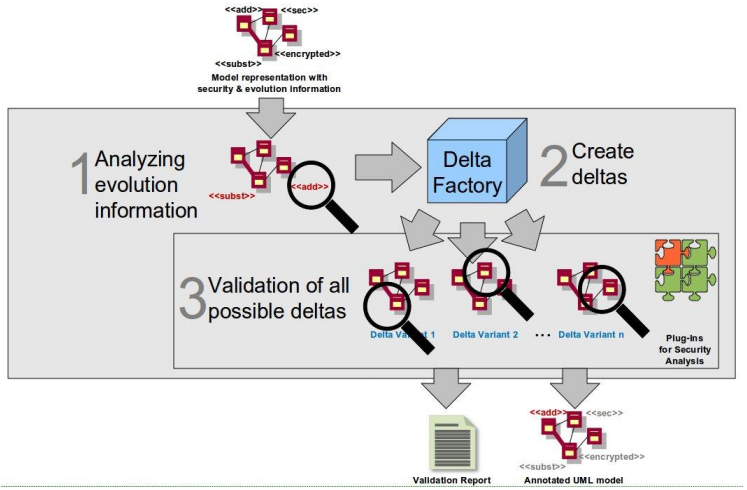


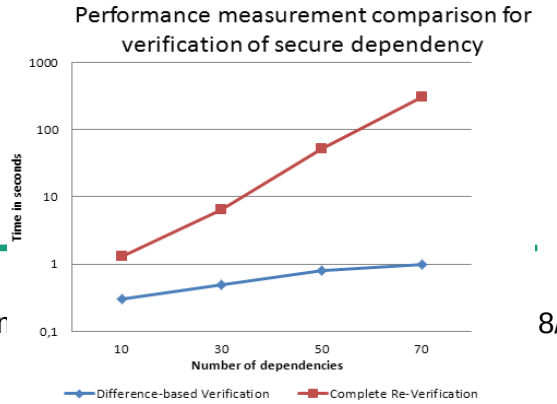

## Differenz-basierte Verifikation – Idee:

- Erstmalige Verifikation: Registrieren, welche **Modellelemente** bei Verifikation gegebener Eigenschaft **referenziert**.
- Im verifizierten Modell **gespeichert**, inkl. Teil-Resultate („proof-carrying models“).
- Mit Heuristiken **Bedingungen an Änderungen** definiert, die Verifikationsergebnis bewahren.
- **Evolutions-Differenz** zwischen altem und neuem Modell berechnen (z.B. mit SiDiff [Kelter]).
- Nur die **Modell-Teile re-verifizieren**, die
  1) sich **geändert** haben und
  2) in der Verifikation **referenziert** wurden und
  3) deren Änderung die Bedingungen **nicht erfüllt**.

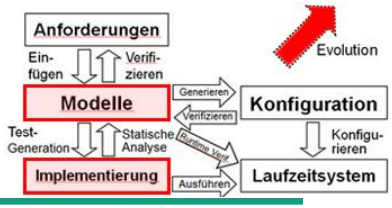
Erheblicher **Performanzgewinn** gegenüber einfacher Re-Verifikation.

**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 $X \in \mathbf{Exp}$ 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 $E \neq E'$.*
...



# Verbindung von Modell und Implementierung bei Evolution



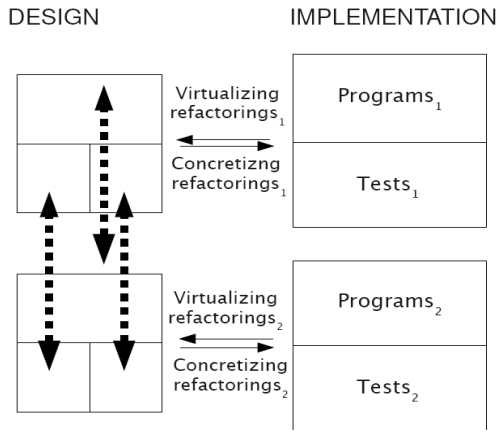

**Ziel: Nachverfolgbarkeit** von **Anforderungen** vs. **Implementierung** bei **Evolution** bewahren.

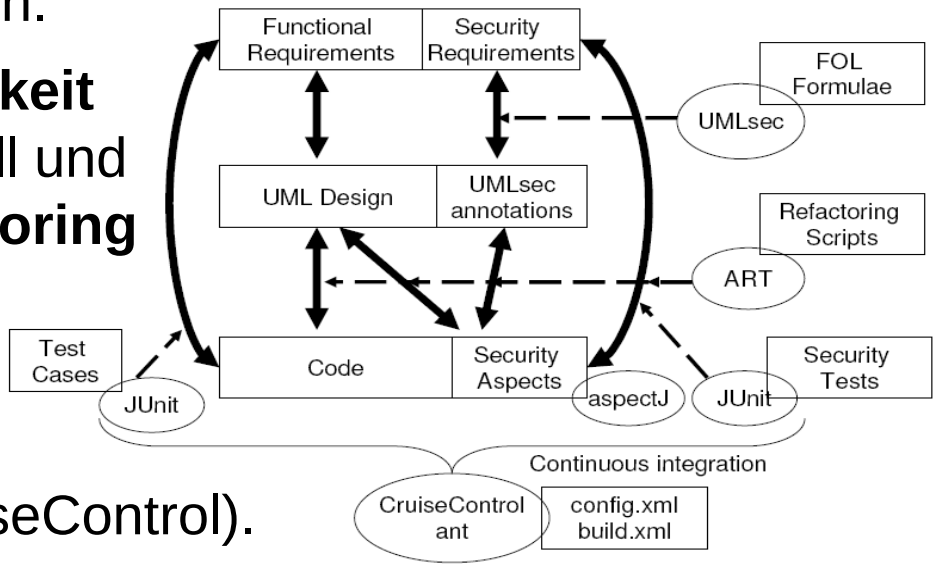
**Beobachtung**: Kann Evolution oft reduzieren auf:

- Hinzufügen / Entfernen von Systemteilen.
- Grundlegende Refactoring-Operationen in Reaktion auf System-Änderungen.

=> **Automatische Nachverfolgbarkeit** der Änderungen zwischen Modell und Implementierung z.B. mit **Refaktoring Scripts** (Eclipse Refactoring Language Toolkit).

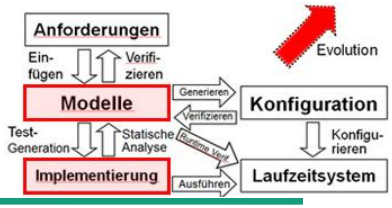
Modell - Implementierung synchron mit Continuous Integration (CruiseControl).




[Bauer, Jürjens, Yu: Run-Time Security Traceability for Evolving Systems. Computer Journal '11]

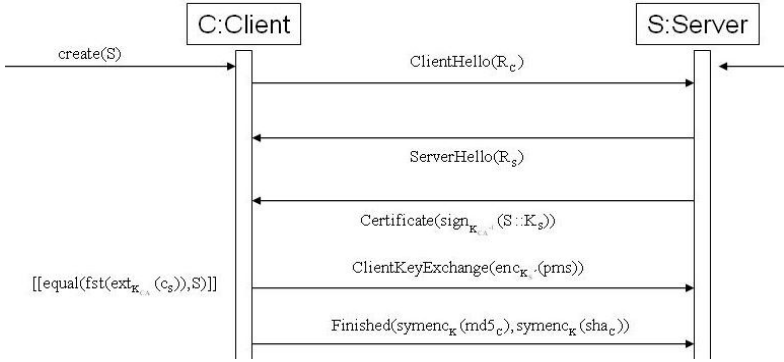
# Anwendung: Java Secure Sockets Extension




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

Mehrere Schwachstellen identifiziert.

Resultate zeigen: auch auf **größere Evolutionen** anwendbar.

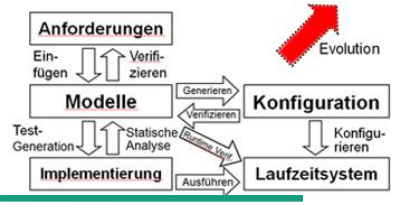



[Yu, Jürjens, Mylopoulos: Traceability for the maintenance of secure software. ICSM'08]

| Symbols | Program entities | Identif. | Refactoring op. |
|---|---|---|---|
| 1. $C$ | clientHello | C | rename.type |
| 2. $S$ | serverHello | S | rename.type |
| 3. $P_{\text{ver}}$ | 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. $S_{\text{id}}$ | 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 |

# Technische Validierung



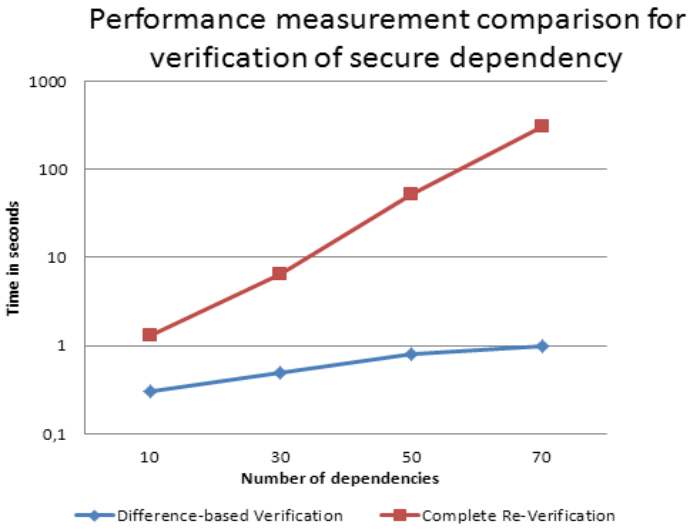

- **Korrektheit**: mittels formaler Semantik
- **Vollständigkeit**: jede Modelltransformation darstellbar als Folge von Löschen, Ändern und Hinzufügen von Modellelementen

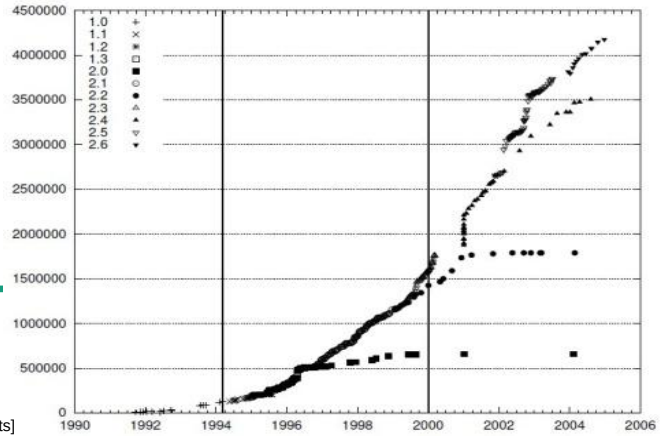
Performanzgewinn am **größten**, wenn **Differenz << Software.** Beispiel-Resultat:

- Evolutions-basierte Verifikation: Laufzeit **linear** in Softwaregröße (bei gleichbleibender Differenzgröße)
- Komplette Reverifikation: Laufzeit **exponentiell** in Softwaregröße

Ist erfüllt:

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



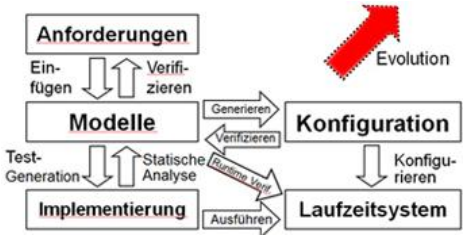




[Robles et al.: Evolution and Growth in Large Libre Software Projects]

# Zusammenfassung: Modellbasierte Qualitätssicherung bei Evolution

**Evolution:** Herausforderung für Qualitätssicherung.

**Frage**: Wann können QS-Ergebnisse nach Evolution wiederverwendet werden ?




**Resultate**: Bedingungen für Bewahrung von Anforderungen…

- … durch zentrale Design-/Architektur-Techniken für Evolution (z.B. **Verfeinerung**, **Modularisierung**).
- … unter Modellevolution („**Differenz-basierte Verifikation**“).
- Evolutions-basierte **statische Analyse** und **Run-time Verification**.
- Implementiert in **Werkzeugen**: erhebliche Performanz- und Skalisierungsverbesserungen gegenüber einfacher Re-Verifikation.

**Geplant**: Migration in Clouds, Software-Produktlinien