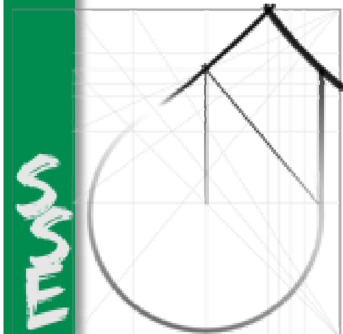
Zum Abstraktionsniveau von Architekturbeschreibungen

Software-Architektur 2011

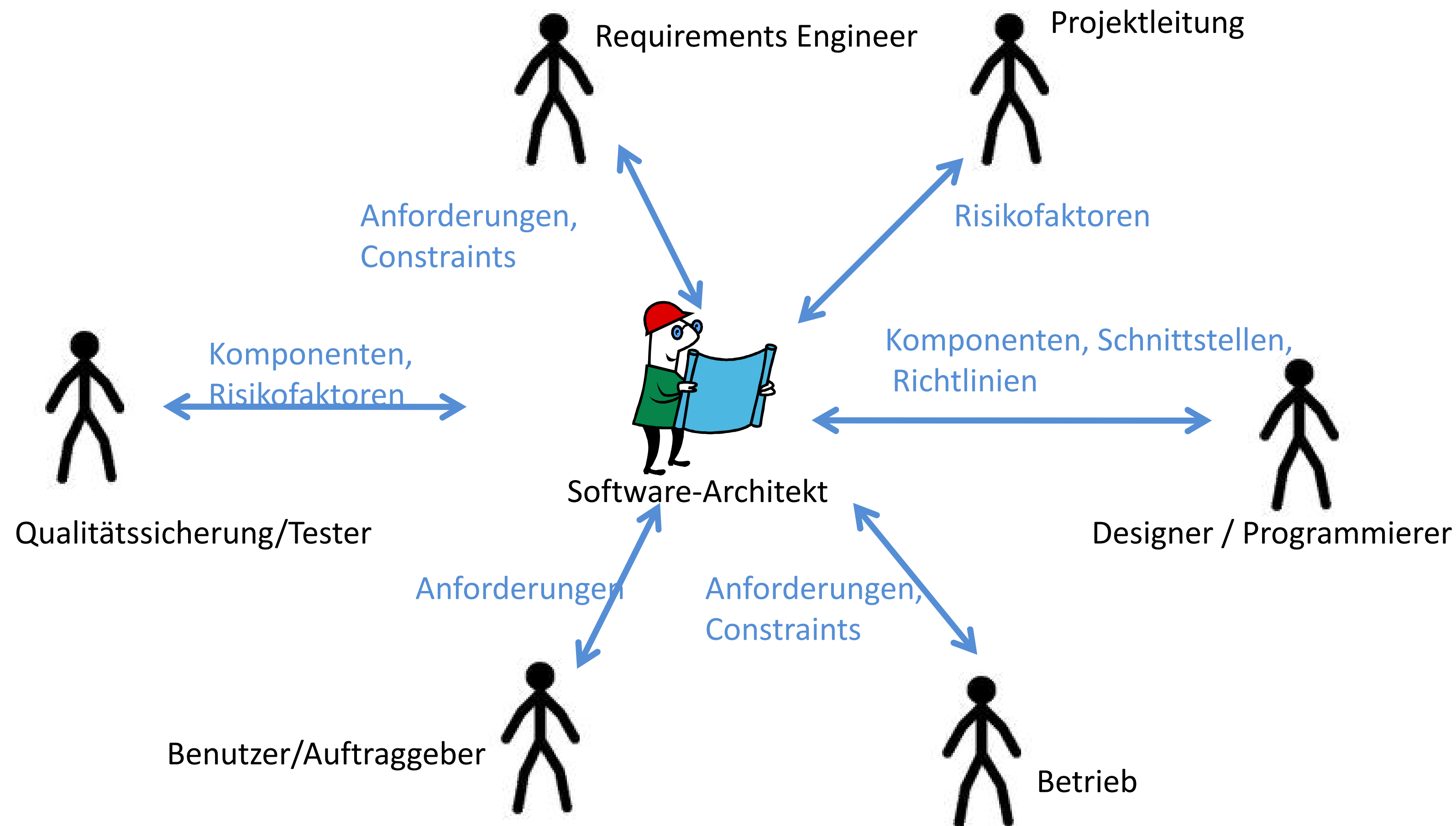
Workshop „Softwarearchitektur: Abstraktion und Visualisierung“

Sebastian Herold



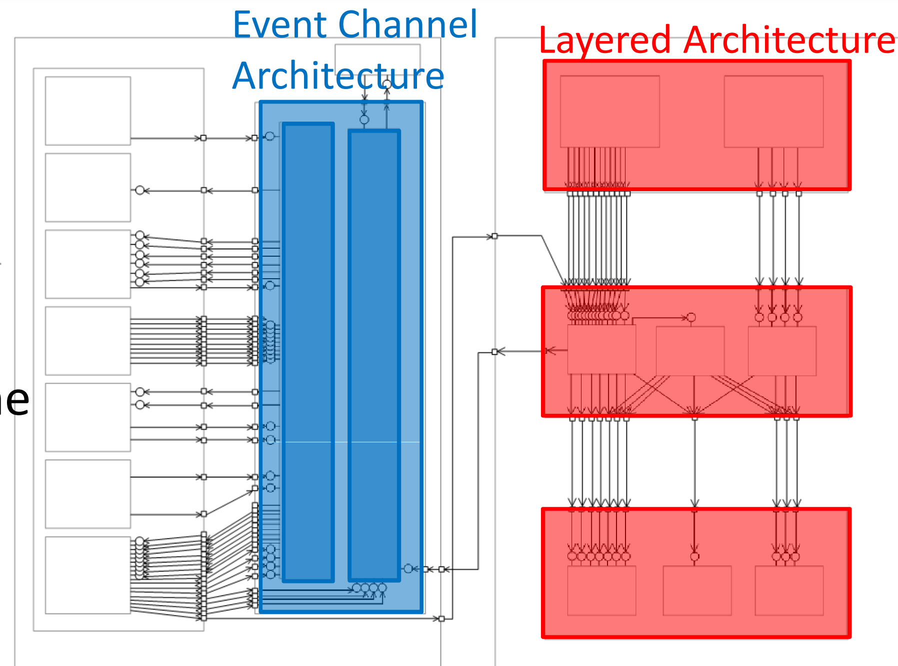
Technische Universität Clausthal
Institut für Informatik - Software Systems Engineering
Lehrstuhl von Prof. Dr. Andreas Rausch
Julius-Albert-Str. 4
38678 Clausthal-Zellerfeld

Software Architecture: Stakeholders



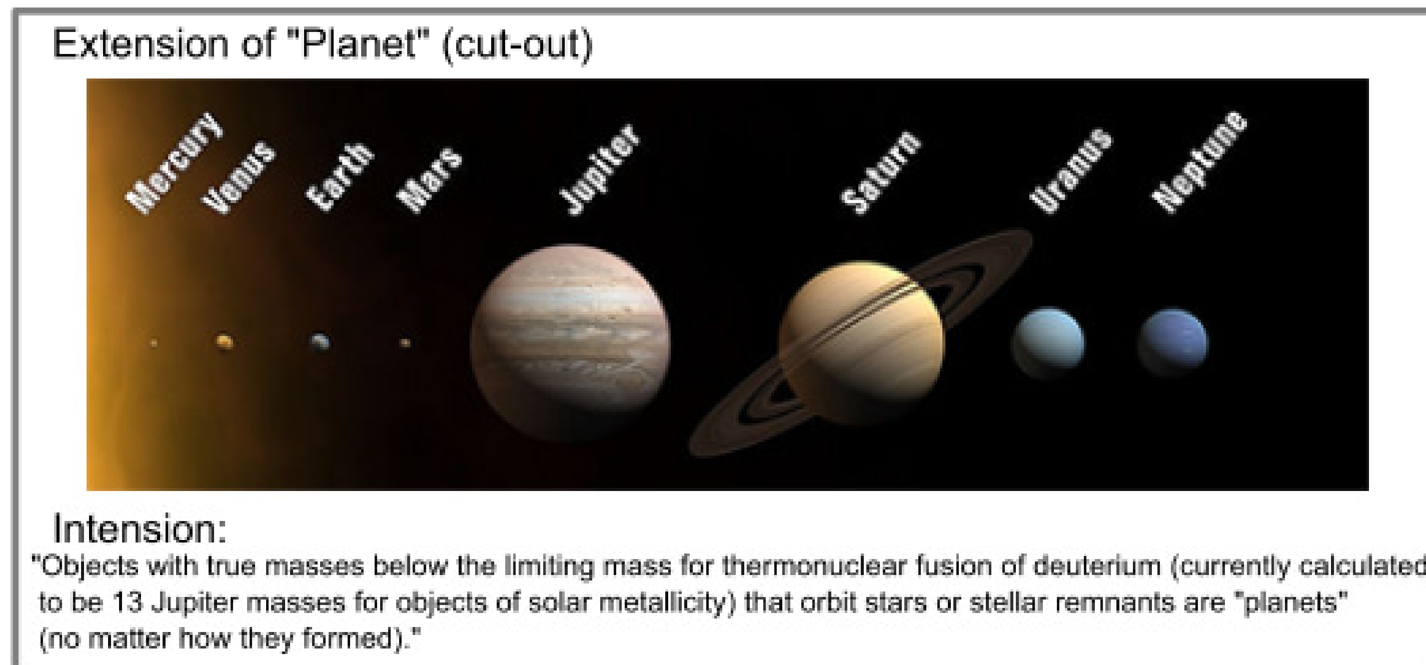
Motivation

- Thema: Überprüfung der Konformität von “Ist”-Designs und – Implementierungen und Soll-Architekturen
- Konformität ist ein wichtiges Qualitätskriterium für Softwaresysteme
- Probleme dabei
  - Überprüfung braucht Tool-Support
  - Heutiger Tool-Support nicht flexibel genug hinsichtlich Vielfältigkeit von Konformitätsbedingungen und Artefakten
  - Tiefergehende Ursache: Fehlende geeignete und abstrahierende Formalisierung / Beschreibung von Architekturen



Unterscheidung Architektur, Design, Implementierung

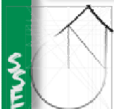
- Zugrunde liegende Frage: Welche Abstraktion bietet eine Softwarearchitektur sinnvollerweise (im Vergleich zur Implementierung/Design)?
- Beobachtung: Implementierungen sind *extensionale* Aussagen, Architekturen *intensionale*



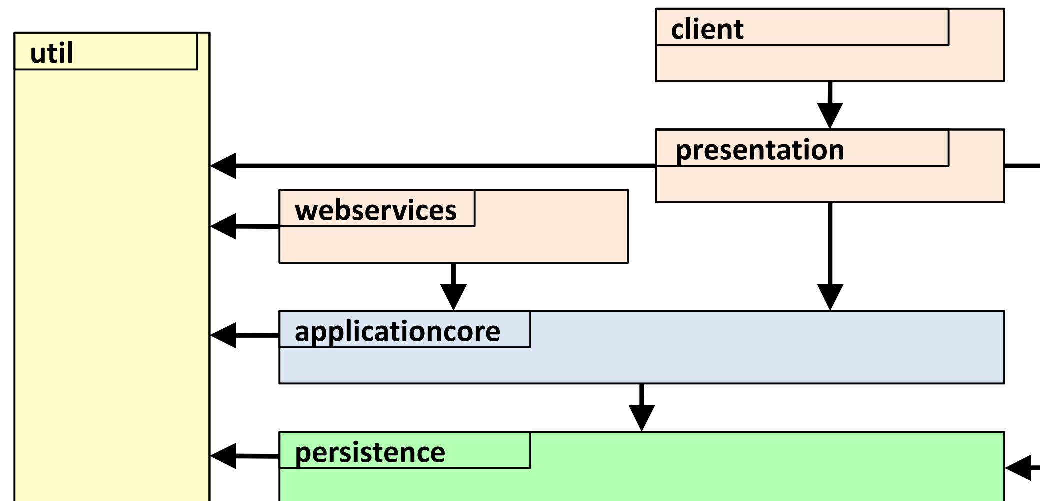
Klassifizierung von Entwurfskonzepten

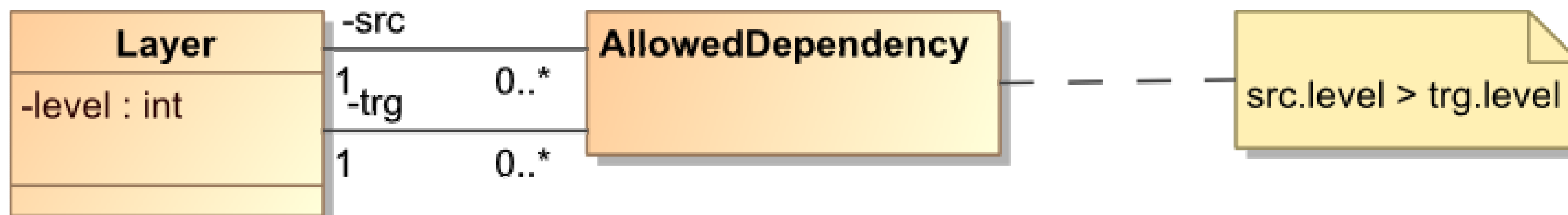
intensional		extensional
nicht-lokal	lokal (=stabil gegenüber Erweiterungen)	
GUI Layer / App Layer: GUI («component» SeatGUI), allowed to use, Application («component» SeatManager, Observer, Observable)	Observable: +registerObserver(observer : Observer) : void, +removeObserver(observer : Observer) : void, +notify() : void; 0..1 -observable; notify has to call observers.update() method; 0..* +observers; Observer: +update() : void	GUI: «component» SeatGUI, observer : FreeSeatsDialog, FreeSeatsDialog, Observer, seats, Observable
Architekturkonzepte	**(Fein-)Designkonzepte**	**Implementierungs-konzepte**

Klassifizierung nach: Eden, A., Hirshfeld, Y, and Kazman, R.: Abstraction classes in software design.
IEE Proceedings - Software, 153,4, (2006), 163-182.

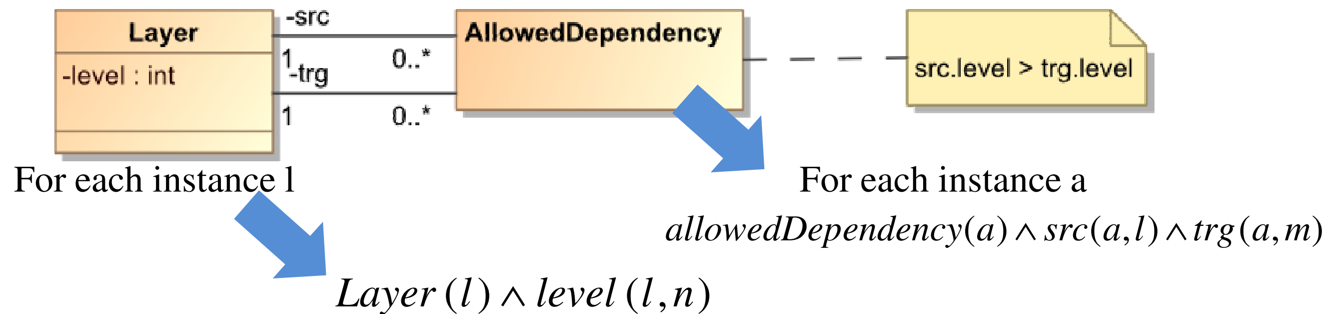


Beispielarchitektur





Architekturen „als logische Ausdrücke“



For each instance l

$$Layer(l) \wedge level(l,n)$$

For each instance a

$$allowedDependency(a) \wedge src(a,l) \wedge trg(a,m)$$

$$\neg\exists m : Layer(m) \wedge \neg allowedDependency(l,m) \wedge dependency(l,m)$$

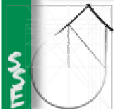
Architekturregel

$$dependency(l,m) = \exists p,q : Package(p) \wedge Package(q) \wedge mapping(l,p) \wedge mapping(m,q) \wedge packageDependency(p,q)$$

$$packageDependency(p,q) = \exists e_1,e_2 : containsElement(p,e_1) \wedge containsElement(q,e_2) \wedge \left(\bigvee_{i=1,2,\ldots 5} depend_i(e_1,e_2) \right)$$

$$depend_1(e,f) = Component(e) \wedge Interface(f) \wedge (provides(e,f) \vee requires(e,f))$$

$$depend_2(e,f) = Interface(e) \wedge Interface(f) \wedge (subtypes(e,f) \vee refersTo(e,f) \vee \exists s,p : Signature(s) \wedge Parameter(p) \wedge definesSignature(e,s) \wedge ((hasParameter(s,p) \wedge parameterType(p,f)) \vee returnType(s,f)))$$



Fazit

- Eine Architektur besteht aus
  - Bausteinen und Beziehungen des Systems bzw. aus denen ein System bestehen soll (im Fall einer *Soll*architektur) und
  - *Architekturregeln,* die globale Bedingungen für die Verfeinerung der Architektur formulieren
- Was sind geeignete Visualisierungstechniken für Architekturregeln?
  - Beschreibungstechniken fokussieren stark auf extensionale Aspekte von Architektur

Literaturhinweise

- A. H. Eden, A., Y. Hirshfeld, and R. Kazman: *Abstraction classes in software design*. IEE Proceedings - Software, 153(4), 2006.
- A. H. Eden and R. Kazman: *Architecture, design, implementation*. Proc. of the 25th Int. Conf. on Software Engineering (ICSE), IEEE Computer Society, 2003.
- C. Deiters, P. Dohrmann, S. Herold, and A. Rausch: *Rule-Based Architectural Compliance Checks for Enterprise Architecture Management*. In: Proc. of the 13th IEEE EDOC Conference, 2009.
- S. Herold. *Checking Architectural Compliance in Component-Based Systems*. In SAC ’10: Proceedings of the 2010 ACM Symposium on Applied Computing.