ARCHITEKTUR TAG

Sieben Geheimnisse erfolgreicher SW-Architekten ①

"Seven Secrets every architect should know" Frank Buschmann

Disclaimer: Kein Anspruch auf Vollständigkeit

- Drive Design through User Tasks
- Be minimalist
- Ensure visibility of domain concepts
- Use ~~uncertainty~~ uncertainty as a drive
- Design between things
- Pay attention to implicit Assumptions
- Eat your own dog food

TODO:
Interaktion zwischen Mensch und Maschine für die, die mit dem System (arbeiten) (Nutzer), und für die, die im System (arbeiten) (Techniker, Entwickler...), geeignet berücksichtigen.
System administrieren und Konfigurieren ist auch eine Anwendung und sollte aufgabenorientiert aufgebaut sein.

→ User tasks
- Anwendungsszenarien UND Entwicklungsszenarien
  wie wird angewendet? wie wird entwickelt?
- vgl. Larry Constantine "Software for use" (Lit.)
  "Unusable Software is useless"
- Walking skeleton: A set of end-to-end slices through the system that correspond to architecturally significant use tasks - with users being end users and developers
  (Alistair Cobburn)
  Vorteil gegenüber Prototyp: Man braucht es am Ende nicht wegzuwerfen

↑
... in Produktqualität!

Sieben Geheimnisse ... (2)

- Architekt muss Entwürfe da abholen, wo sie sind, sonst hat man auch mit dem besten und coolsten Design als Architekt verloren

  ↑ Statt: "Oh, die sind alle zu blöd, das umzusetzen."

→ Minimalismus

- Perfektion ist, wenn man nix mehr wegnehmen kann
  (A. de Saint-Exupéry)

- Erweiterbarkeit: Geht irgendwie immer. Besser: Wenn man nachweisbar "toten Code" auch wegnehmen kann, ohne daß der Rest nicht mehr funktioniert.

- Compression instead of abstraction (wo es passt)

  ✓ relates to directness of expression

  ↳ concerns removal of specific detail

  ↳ compression can come from careful abstraction

- vgl. Ward Cunningham (Literatur); Zitat: "What is the simplest design that possibly could work?" - Nicht nur von den Alternativen, die ich kenne?

→ Visibility of domain concepts

- Statt String und int Modellieg der Domäne, Bsp

Date
Integer day, month, year
String getDate() Integer getDateInMonth() Integer getMonth() Integer getYear()

ISBN
String isbn
String asString()

("Google?!")

- Aussagekräftige Benennung: Bsp Produktkatalog statt Gutscheine

Sieben Geheimnisse ...

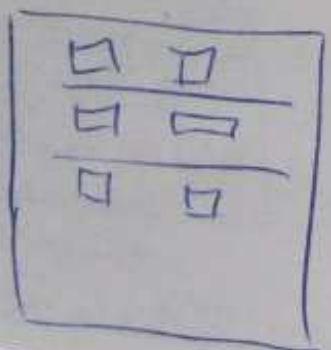
- Explizit: Benennung, Komponenten, Performanz ... mix Implizites
- "Telefontest": Jed anderen, der das beschriebene Design mit Papier + Stift + Basic UML Kenntnissen anfasst
- Ausdrucksstarke Sprache

→ Unsicherheit

- Unsicherheit nicht als Ausrede für Unentschiedenheit und Vermeidungsverhalten
- vgl. Kevlin Henney (Lit.) "97 things every architect should know"
- Decide explicitly (!) to not decide now (!!)
  - ↳ zurück ins Haifischbecken, nochmal rein an die Requirements, oder: Walking Skeleton und Feedback dazu einholen oder: aktive Design Reviews / oder: Change Request Process anfordern
  - ↳ NICHT: Feigheit vor dem Feind
- "Shot gun surgery" Eine kleine Stelle Code ändern, ups, dann muss ich die und die Stellen auch noch ändern, und deswegen dann noch viel mehr und dann deswegen ....

→ Design between things

- Design along the white space, im Niemandsland zwischen den Komponenten
- Bsp Design:



Striche waren das Problem; die Striche stellen immer Middleware dar. Wo kam das Problem her? Architekt: "Middle ware ist doch common infrastructure." Nur, dass die sich selbst entwickelt hat

Sieben Gebeimnisse ... (4)

- Aufgabe des Architekten:
  Interfaces - complete, meaningful
  - role-specific, stability
  Interaction - simple, meaningful
  - end-to-end quality
  Integration - all kinds of integration

→ Pay Attention to implicit Assumptions

- vgl Koskinen: http://users.jyu.fi/~nkoskinen/sun-costs.htm
  50% der Zeit bei Wasky verbringen Entwickler damit, den Code zu verstehen (ec kommt nur string und int vor ?!?)
  Was hilft? Domänenkonzepte klar machen (S.O., Benennung EG)
- Verstehen ist da Kern; Nachmachen und Runtertippen ist nicht sonderlich schwierig
- selbstverständliche Eigenschaften (Performance, Wartbarkeit...) sind entscheidend (bringen keinen Gewinn, aber wenn sie fehlen, wird es evtl richtig teuer!) → Architekten müssen auf Nichtgesagtes achten (Selbstverständlichkeiten)
  (Erwartungen)

→ Eat your dog food

- Joel Spolsky: Architect Pedestrian statt
  Architekt Astronaut (okay, braucht man auch, aber nicht nur)

- Architekten sollten tunlichst auch selbst implementieren, z.B. als Pair Programming. Nicht täglich, aber regelmäßig:
  "Ist meine Architektur programmierbar?"
- "Programmierer sind die Programmiersprache der Architekten"