THOMAS MATZNER
Berater für Systemanalyse

Software-Konzeption Quick and Tidy

7it Forum
München, 19. 10. 2009

Die traditionelle Sichtweise auf Software-Konzeption befriedigt nicht

„Entweder den Termin halten – oder gute Entwurfsentscheidungen treffen“

Das heißt: Entscheidung für sauberen Entwurf nützt dem Entscheider nicht

Wird das Versprechen später wirklich gehalten?

Liegt Software-Konzeption im Trend?

- Etablierte Sprachen: UML, ARIS etc.
- Know-how vorhanden
- Technische Umsetzung durch OO-Sprachen, Frameworks etc. günstig
- Time to market – keine Zeit zum Überlegen…
- Agile Manifesto setzt (scheinbar?) Prioritäten weg von Architekturen, Modellen etc.

Der Widerspruch Quick or tidy besteht nur zum Schein

Schon immer gab es gut konstruierte Systeme, die in time and budget gebaut werden mußten.

Die Wartungsphase beginnt am 2. Tag

Also müssen wir gute Konzeption als Erfolgsfaktor darstellen

… und dieses Versprechen auch erfüllen

Software-Konzeption Quick and tidy: Vier Bausteine

1. Nicht alles gleich wichtig nehmen
2. Architekturziele kennen und verfolgen
3. Nichts zusätzlich machen
4. Grundmuster von Anwendungen einbringen

1. Nicht alles gleich wichtig nehmen

Dilemma der Schöpfer von Sprachen und Verfahren des Software-Engineering:

Die Welt will keine schlanken Lösungen.

So entstehen Sprachen wie

- UML 2: Fast 1000 Seiten nur für die Definition
- ARIS: > 100 Diagrammtypen

Die bloße Anzahl von Modellelementen ist kein Grund, stolz zu sein – eher verbrauchte Ressource…

Vorschlag für ein Ranking von Modellkonstrukten (i)

1. Klassenmodell (incl. Komponenten, Interfaces)
   - Grundstrukturen der Anwendung: „Naturgesetze“ anstatt Zufälligkeiten
   - Grundlage für Entwicklungsfähigkeit – auch schon kurzfristig
   - Single point of truth – Separation of concerns
   - Klare Komponenten- und Interface-Struktur ist oft Antwort auf Time to Market
2. Zustandsmodell
   - Führt zu vollständigen und fachlich konsistenten Funktionen
   - Stabil – im Gegensatz zu Workflows etc.

Vorschlag für ein Ranking… (ii)

3. Ablaufmodelle (Sequenzdiagramme o.ä.)
   - Veranschaulichung der abstrakten Klassen- und Zustandsmodelle
   - Aufspüren vergessener Funktionalität
4. Workflowmodelle
   - Wichtig für konkrete Anwendung, aber nicht als Grundbauplan
   - Systemfunktionen sollten „workflow agnostic“ sein
5. Use Cases
   - Kein strukturbildendes Konstrukt
   - Widerspruch zwischen „anschaulich“ und „präzise“ nur scheinbar aufgehoben

2. Architekturziele kennen und verfolgen

Von „Man muß *xyz* tun…“ zu „Wir wollen *xyz* tun, weil…“

Viele Architekturziele haben ihren Ursprung im Business. Diesen herausfinden, z.B.:

- Variable Oberflächen: Mobile Geräte, fremde Portale…
- Kooperation, M&A: Funktionen werden z.T. von Partnern erbracht
- Technologische Risiken: Welche Workflow-Engine wird überleben?

Konkrete Ziele lassen sich besser argumentieren und durchhalten als allgemeine Wahrheiten

Ziele ermitteln mit Stakeholdern im Business

Architekturziele sind Teil der Projektdefinition

Starkes Argument im Alltag: Wir wollen hier ein klares Interface, *weil*…

3. Nichts zusätzlich machen

„Konzeption kostet extra Zeit und Geld“ ist eine Täuschung.

Sie ist nur wahr, wenn man bei der Konzeption Probleme löst, die man gar nicht lösen muß.

„Jetzt muß ich eine Woche lang Dinge beschreiben, die hinterher keiner braucht“ –

„Jetzt soll ich ein Dokument lesen, in dem nichts Interessantes steht“ –

… sind Warnsignale

Das Modellkonstrukt-Ranking muß der Situation angepaßt werden (i)

1. Klassenmodell
   Vollständig, aber mit wechselnder Tiefe
   Triviale Daten und Funktionen (CRUD) nicht immer wieder beschreiben
   Man sollte sehen können, wo die schwierigen Teile der Anwendung sind
2. Zustandsmodell
   Nur diejenigen Objekte mit nichttrivialen Zustandsübergängen (ca. 25%)

Ranking anpassen… (ii)

3. Ablaufmodelle
   Nur an Schwerpunkten einsetzen, wo die Zusammenhänge diffizil sind und veranschaulicht werden müssen – der Aufwand ist vergleichsweise hoch
4. Workflows
   Nur dort, wo ein Workflow-Ansatz mit mehreren Beteiligten, Rollen, Aktivitätstypen Sinn macht
   Querschnittskonzepte nur einmal festlegen (Freigaben, Abbruch)
5. Use Cases – besonders schwierig, da Versuchung, die Inhalte der anderen Modelle zu wiederholen.

Generelle Regeln zum Vermeiden von zeitraubendem Leerlauf

Die Copy-Taste ist in der Konzeption genauso verpönt wie in der Programmierung.

Querschnittskonzepte erkennen und nur einmal beschreiben (etwa Pflege von Referenzdaten).

Die verschiedenen Modelle aufeinander beziehen: Z.B. Multiplizität im Klassenmodell ist Bedingung auch für alle Funktionen.

4. Grundmuster von Anwendungen einbringen

„Wir entwerfen unser System entlang der konkreten Anforderungen unserer Anwender heute.“

Das ist gut für Workflow und GUI – meist nicht gut für die Grundstrukturen der Anwendung.

- Organisation und Abläufe sind das Flüchtigste in einem Unternehmen, einem IT-System.
- Es stimmt nicht, daß alle paar Jahre die Business-Prinzipien revolutioniert werden.
- Die kaufmännischen Prinzipien (Venedig, Renaissance) gelten auch heute noch.

Gute Nachricht: Die Grundlagen des Wirtschaftens und Produzierens sind langlebig

Viele Vorgänge lassen sich als Auftragsabwicklung darstellen:

- Leistungsangebot besteht
- Welche Leistung wird benötigt?
- Kann sie rechtzeitig erbracht werden?
- Dann beauftragen wir sie.
- Jetzt wird der Auftrag ausgeführt.
- Wenn etwas schiefgeht, läuft der Prozeß ein Stück zurück.
- Auftrag fertig, Übergabe.

Deshalb ist mir auch die Zustandsmodellierung so wichtig…

Beispiele für Auftragsabwicklungen

- Kundenauftrag, klar. Lager-, Liefer-, Produktionsauftrag
- Auftrag an Personen im Unternehmen, Information zu einem Thema zu liefern
- Auftrag an ein Nachbarsystem, Informationen zu einem Thema zu liefern

Das Grundmuster hilft

- Gerüste von Klassen- und Zustandsmodellen hinzustellen, kein „weißes Blatt Papier“ mehr,
- zu fragen, wo die interessanten Funktionen sind, etwa: Kann sich nach Start der Produktion der Auftragsinhalt noch ändern?

Weitere Beispiele für Grundmuster

- Historisierung und Terminierung
- Pflege von Stamm- und Referenzdaten
- Abwicklung von Workflows
- Abgleich von Datenbeständen

Klappt das auch in der Praxis? Ein paar Fallbeispiele (i)

- ADAC Mitgliederverwaltung
  - Entwicklung 1986..1988 unter hohem Zeitdruck
  - Konsequentes Design nach Datenabstraktion und Schichtenarchitektur
  - Persistenzframework
  - GUI- und Controller-Framework
  - … das alles mit PL/1, CICS und Adabas
  - Aussage zwanzig Jahre später: Das System ist immer noch nicht Legacy, es bildet alle aktuellen Business-Anforderungen gut ab.

Fallbeispiele (ii)

- Auftragsabwicklung eines Versandhändlers
  - System 5 Jahre zuvor neu entwickelt. Wo klemmt es?
  - Anbindung von Geschäftspartnern
  - Eigener Kundenauftrag. Aber fremder Lager- und Lieferauftrag (Wein). Gar kein Lager- und Lieferauftrag (digitale Ware)
  - Fazit: Es wurde versäumt, die passenden Abstraktionen im System abzubilden.

Fallbeispiele (iii)

- Warenwirtschaftssystem eines Großhändlers
  - Zwei der Gründe für Neuentwicklung liegen direkt im Datenmodell: Nur ein Preis je Artikel, Vermischung von Geld- und Warenfluß
  - Hemmnis für neue Geschäftsmodelle, etwa Führung von Herstellerlagern
  - Aus dem vorigen Beispiel gelernt, also:
  - Separierung Kundenauftrag, Lagerauftrag, Versandauftrag
  - Aufwand für die Grundstrukturen: enthalten in 22 BT für die IT-Strategie dieses Bereichs

Es bleibt spannend. Ein paar nach wie vor schwierige Aufgaben:

- Strukturen gleichzeitig technisch präzise und für den Anwender anschaulich darstellen: Offen – vermutlich unlösbar.
- Die richtige Abstraktion finden. Wurde mit Strukturierter Programmierung, Datenabstraktion, OO, MDA, SOA immer wieder versprochen: Machbar, erfordert Willen.
- Das Wichtige vom Unwichtigen trennen: Erfahrung, Politik…