Antipattern

Vendor Lock-In
und
Architecture by Implication

David Salz (david.salz@snafu.de)

Vendor Lock-In

- Architektur – Antipattern
- auch:
  - Product-Dependent Architecture
    (Produktabhängige Architektur)
  - Bondage and Submission
    (Gefesselt und Unterworfen)
  - Pottersville
    (nach dem Film „It‘s a Wonderful Life“, USA 1946)

It‘s A Wonderful Life

It‘s A Wonderful Life, USA 1946

Bilder: International Movie Database, http://www.imdb.com

Vendor Lock-In

- **Problem**: Abhängigkeit von einer proprietären Software („Produkt“)
- **Ursachen:** Faulheit, Apathie, Stolz, Ignoranz
- **Ausmaß:** komplettes System


Furchtbarer Fehler!
Dieses Dokument wurde mit Software2000 erstellt und kann mit Software99 leider nicht mehr geöffnet werden. Bitte erwerben Sie ein Update!
OK


Support abgelaufen!
Software2002 ist ab sofort erhältlich!
Software2001 kann leider nicht länger verwendet werden. Bitte erwerben Sie ein Update oder löschen Sie alle Ihre Dateien...
OK

Proprietäre Software

- **Proprietär**

- vom lat. Proprietarius
- ~ im Besitz von jemandem
- heute meist: urheberrechtlich bzw. durch Markenrecht geschützt

Proprietäre Software

proprietary

1. Im Marketing-Deutsch: überlegen, erfüllt von einzigartiger Magie durch die unübertroffene Brillanz der Hard- bzw. Softwaredesigner unserer Firma

From The Free On-line Dictionary of Computing (06 Jun 01)

Proprietäre Software

proprietary

2. In der Sprache der Hacker und User minderwertig; impliziert, dass ein Produkt nicht mit offenen Systemstandards konform geht und deshalb den Kunden völlig dem Hersteller ausliefert, der Service- und Updategebühren erhöhen kann, sobald er den Kunden mit dem ersten Kauf „eingeschlossen“ hat

From The Free On-line Dictionary of Computing (06 Jun 01)

Unsere Architektur ist Microsoft (Oracle, CORBA...).

bedeutet:

Wir haben keine Architektur.
Wir sind komplett von Microsoft abhängig!

Symptome

- Update des Produktes und plötzlich läuft unsere Applikation nicht mehr...
- d.h. zwangsweise gleiche Update-Zyklen
- Features werden verschoben, weil Features des Produktes verschoben werden
- Wissen der Entwickler ist mit jedem Produktupdate überholt...

Konsequenzen

- Falls wir ein Produkt-Update verpassen, müssen wir meist ganz von vorn anfangen...
- Kein offenes System!
- Falls das Produkt sich in die falsche Richtung entwickelt oder irgendwann nicht mehr fortgesetzt wird...

Typische Ursachen

- Produkt nach Marketing- statt nach technischen Gesichtspunkten gewählt
- Applikationsprogrammierung „verlangt nun mal Wissen über das Produkt“
- Kein technischer Ansatz für Isolierung vom Produkt

Soziale Aspekte

Wer profitiert eigentlich von meinem Unglück... ?

- Meist wird ein Vendor Lock-In akzeptiert, weil der Hersteller bestimmte Features verspricht. Einen Vertrag darüber gibt es aber nicht...
- Aber auch der Hersteller hat Interessen...

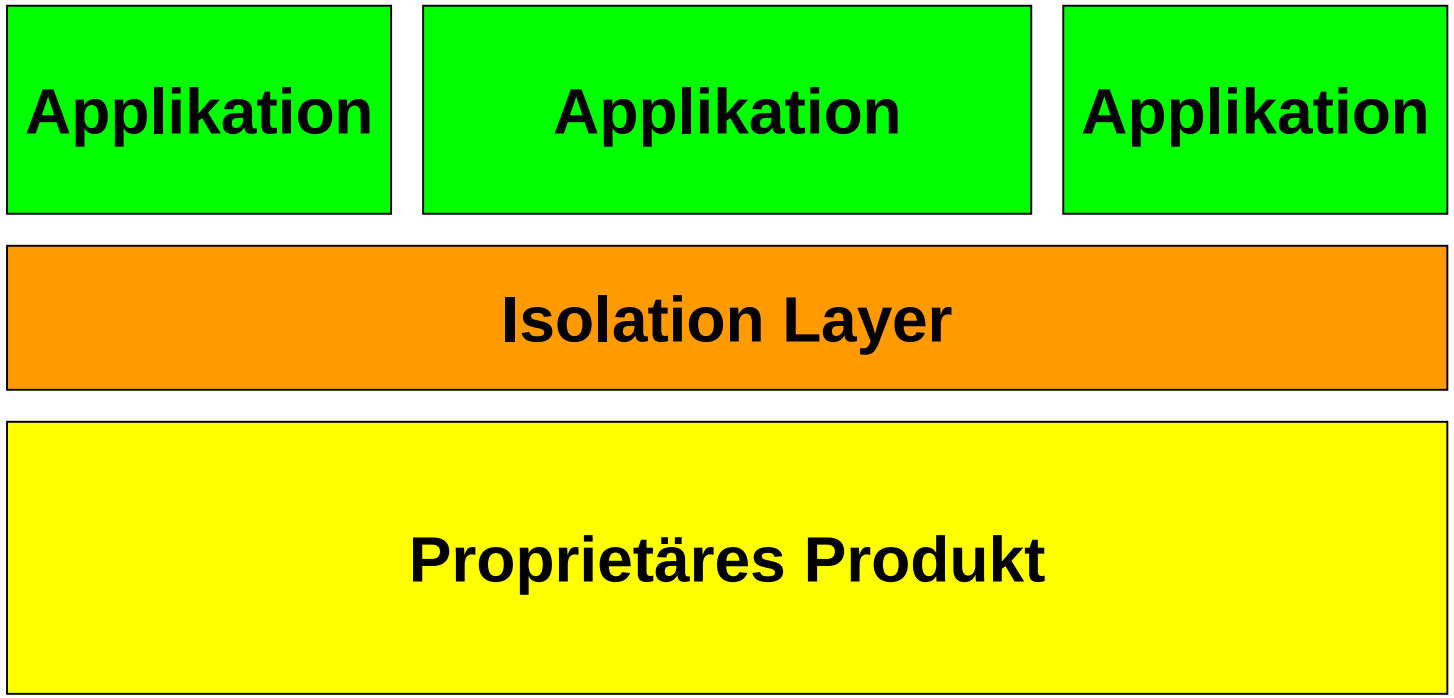
Ausnahmen

- Falls das Produkt den größten Teil der Funktionalität (und des Codes) unseres Produktes ausmacht, sind diese Nachteile akzeptabel.
- Auch als „Testballon“ oder erste Erfahrung auf einem neuen Gebiet

Refactoring

Die Lösung:

Isolation Layer (Isolationsschicht)



Voraussetzung:

- es gibt zwei trennbare Schichten
- Veränderungen im zugrundeliegenden Produkt sind zu erwarten...
- ... oder Migration auf oder Unterstützung von anderen Produkten geplant ...
- ... oder das Interface des Produktes ist zu primitiv oder zu flexibel für uns

Vorteile

- (+) trennt Applikations- und Interfaceentwicklung (= 2 Entwicklungsteams)
- (+) einfacheres & besseres Interface
- (+) Unterstützung mehrerer Infrastrukturen transparent möglich
- (+) Risiken und Kosten durch Abhängigkeit vom Produkthersteller gemindert

Nachteile:

- (-) Isolation Layer muss gewartet werden, möglicherweise auf mehreren Plattformen
- (-) Interface des Isolation Layer muss initial festgelegt werden
- (-) Änderung der Interfaces erfordert Koordinationsaufwand

Verwandte Pattern

- Adapter Pattern (Object Wrapper) ist praktisch ein Isolation Layer für ein einzelnes Objekt
- Profile Pattern – man kann den Isolation Layer als ein spezielles Profil einer Middleware betrachten

Architecture by Implication

Architecture by Implication

- Etwa: Architektur durch Verwicklung
- Architektur-Antipattern

- **Problem:** Mangelnde Planung
- **Ursachen:** Stolz, Faulheit
- **Ausmaß:** komplettes System

Symptome

- fehlende Architektur und fehlende Vorausplanung
- Probleme mit Komplexität, Performance und Technologie kommen erst später zum Vorschein
- neue Technologien werden ignoriert
- keine Notfallpläne

**Sowas haben wir schon mal gemacht,
da müssen wir nicht neu planen...**

oder auch:

**Kein Plan überlebt den ersten Kontakt
mit dem Feind...**

Konsequenzen

- schlechte Performance
- übermäßige Komplexität
- missverstandene Anforderungen
- schlechte Benutzbarkeit
- ... schlimmstenfalls Projektabbruch

Typische Ursachen

- kein Risikomanagement
- Selbstüberschätzung der Manager / Architekten / Entwickler
- angebliche frühere „Erfahrung“ im Problembereich
- Architektur entsteht durch Programmierung statt Planung

Ausnahmen

- wenn ein neues Projekt tatsächlich nur minimal von einem schon durchgeführten abweicht
- Forschungsprojekte zur Orientierung in einem neuen Problembereich

Refactoring

- **Lösung:** eine vernünftige Architektur und Planung

- Wichtig: Sichtweisen & Stakeholder
- Stakeholder ist jeder, der ein Interesse am Projekt hat
  - der Kunde
  - die späteren Nutzer
  - die Entwickler
  - ...

Refactoring

- 1. Definition der Ziele der Architektur
  - Was wollen wir erreichen?
  - Welche Stakeholder gibt es?
- 2. Definition der Fragen
  - Welche Architektur-Fragen müssen für die Stakeholder beantwortet werden

Refactoring

- 3. Sichtweisen festlegen
  - Architektur findet immer aus verschiedenen Sichtweisen statt
- 4. Analyse der Sichtweisen
  - Detaillierte Ausarbeitung der Architektur
- 5. Konsistenzprüfung
  - Sichtweisen müssen konsistent sein

Refactoring

- 6 Sichtweisen und Bedürfnisse abstimmen
  - Sichtweisen sollten die Fragen der Stakeholder beantworten
  - Systematisch nach Lücken suchen
  - Notfalls neue Sichtweisen einführen

Refactoring

- 7. Die Architektur kommunizieren
  - allen Stakeholdern
  - ... besonders den Entwicklern
  - Dokumente anfertigen
- 8. Validation
  - Architektur und Implementation sollten übereinstimmen
  - Bei Unterschieden muss Implementation oder Dokumentation geändert werden

Quellen

- Anti-Pattern Homepage
  www.antipatterns.com
- „Anti-Patterns“, William J. Brown et al
- Free Online Dictionary of Computing
- NetWords, www.Networds.de

Vielen Dank für‘s Zuhören!

Fragen?