ENTWICKLERTAG

Größe zählt

...

Bild: © kliverap / freeimages.com

Zürich

Alfred Wullschleger, Schweizer Nationalbank: ***„Ein Softwaresystem ist wie die Stadt Zürich…“***

Bild: © Südliche Aussicht von der Weid bei Zürich, Heinrich Keller, 1880 / ETH Zürich

Lebenszyklus

- Entwicklung
- Inbetriebnahme
- Betrieb
- Außerbetriebnahme/Ablösung

TCO laut Gartner

What the CFO Wants to Know and the Project Manager Wants to Hide

Aus: „The Cost and Risk of Application Development Decisions“, Gartner Research

Aufwand Neuentwicklung

- XP: four project variables: Resources, Scope, Quality, Time (c2-Wiki)
- Also ist der Aufwand eine Funktion von
  - Funktionsumfang
  - Qualität

Betriebsphase

- Wartung
  - Fehlerbehebung
  - Anpassung an Änderungen in Umgebung/Infrastruktur
- Weiterentwicklung
  - Neuerungen
  - Änderungen

Aufwand Betriebsphase?

- Sollte doch gleich wie für die Neuentwicklung sein…
- …oder?

Ein Experiment

- 1 Aufgabe
- 4 Systemhäuser
- 4 Implementierungen
- Zuweisung an Nutzer per Zufallsgenerator
- mehrere Jahre Nutzung
- Dann in allen Systemen die gleichen Änderungen vornehmen

Ein Experiment

- Abschätzung der Wartbarkeit von unabhängigem Experten
- Lag völlig daneben

=>

- Tatsächlicher Aufwand hängt NUR vom Codeumfang ab!
- Qualität/Technical Dept spielt KEINE Rolle!

http://www.leanway.no/code-is-the-problem/

Bild: © freedesignfile.com

Aufwand Betriebsphase

- NICHT gleich wie für Neuentwicklung!
- Rückwirkung auf bestehendem Code

=>

- Aufwand ist Funktion von
  - Funktionsumfang
  - Qualität
  - Codeumfang
- Die Abhängigkeit vom Codeumfang ist nichtlinear

Einfluß von Qualität/Technical Dept

- Sicher nicht null
- Aber wesentlich kleiner als Codeumfang
- Codemetriken konnten bislang empirisch nicht mit der Wartbarkeit korreliert werden

Maximierung der Wartbarkeit

- Der Codeumfang ist wegen des nichtlinearen Einflusses der wichtigste Parameter
- Die Stärke der Nichtlinearität wird bestimmt durch
  - Architektur
  - Technologie
  - Qualität

Ansätze zur Maximierung der Wartbarkeit

- Codeumfang minimieren
- „Gute“ Architekturen implementieren
  - Lose Kopplung
- Geeignete Technologien auswählen
  - Lose Kopplung
- Gute Codequalität erreichen
  - Lose Kopplung

Funktionsumfang klein halten

- YAGNI (You ain‘t gonna need it)
- YDNIA (You don‘t need it anymore)
- Konsequent wertorientierte Priorisierung: nur das implementieren, was Wert für den Kunden darstellt

Codeumfang klein halten

- YAGNI
  - Keine unnötige Funktionalität implementieren
  - Nicht „auf Vorrat“ codieren
  - Nicht optimieren, bevor es notwendig ist (premature optimization)
- YDNIA
  - Nicht mehr benötigte Funktionalität (sicher) entfernen

Codeumfang klein halten

- „Most simple thing that could possibly work“
  - Selbst bei klar definierten Aufgaben, gibt es viele Wege zum Ziel
  - Studie „Code Similarities Beyond Copy & Paste“ von Elmar Jürgens, Florian Deissenböck, Benjamin Hummel:
    109 unterschiedliche Lösungen, kürzeste Implementierung 8 Zeilen, längste 55 Zeilen, mit unterschiedlicher Verschachtelungstiefe und cyclomatischer Komplexität
- Ockhams Rasiermesser: ***„Entitäten dürfen nicht über das Notwendige hinaus vermehrt werden.“***

Reducing Complexity

- Kein „Reducing Complexity“, wenn die Komplexität der Domäne wesentlich für die Funktionalität des Systems ist!
- Einstein: ***„Man muß die Dinge so einfach wie möglich machen. Aber nicht einfacher.“***

Bild: © hardwear / spreadshirt.de

Redundanz vermeiden

- Kein Copy & Paste
- Möglichst EINE durchgängige Technologie
  - Vermeidet Wiederholungen an den APIs

Refactoring

- Refactoring ist Optimierung des Codeumfangs
- Refactoring als kontinuierliche Aufgabe

Vollständige Codebasis

- Boilerplatecode
- Code außerhalb der Codebasis: Frameworks, Bibliotheken
  - Z. B. Dependency Injection macht den Code umfangreicher, komplexer und schwerer nachvollziehbar: Funktionalität ist nicht mehr an nur einer Stelle implementiert, sondern befindet sich auch in externen Konfigurationsdateien
- Nutzen von Wiederverwendung nicht überschätzen! (kann eine Form von Premature Optimization sein)

Anforderungen an die Technologie

- Maximale Semantik
- Minimale Syntax
- „Sprechender“ Code
  - Muß auch nach Jahren noch verstanden werden
- DSLs
- DDD, „generische DSLs“

Formbarkeit

- Neuerungen und Änderungen sollten so unabhängig wie möglich vom existierenden Code sein
- Fernwirkungen im Code sollten vermieden werden
  - Technologisch bedingt (z. B. statische Typisierung)
    - siehe z. B.: „IEEE SOFTWARE: Dynamically Typed Languages“ von Laurence Tratt, Roel Wuyts
  - Implementierungsbedingt (Spaghetti-Code, etc.)
  - „Wellen“ und „Brüche“ im Codekörper
- Je weniger Auswirkungen auf andere Codeteile, um so kleiner ist der Implementierungs- und Testaufwand

Agile Prozesse

- Entwicklung und Betrieb sind bei fortgeschrittenen agilen Methoden nicht mehr getrennt
- Viele Agile Praktiken sind geeignet, die Codebasis klein zu halten
- Continuous Delivery lehrt uns: ***„When it hurts, do it more often“***

Unser Mantra

- Einfach klein ist schön!

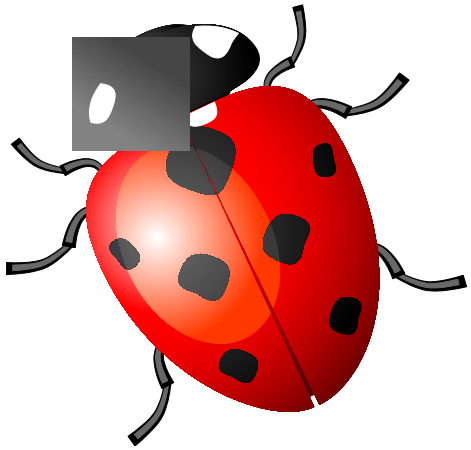

Bild: © Robertas Pezas / greatvectors.com

Praxisbeispiel

- Smalltalk: seit 1980 als Image-basiertes System kontinuierlich weitergegeben und weiter entwickelt
  - d. h. die Basis-Objekte von heute sind DIESELBEN Objekte von damals, aber weiterentwickelt
- In unserer Produktentwicklung haben wir in den letzten Jahren unter Wahrung der Rückwärtskompatibilität ca. 30% Code entfernt
- Dadurch wurden Ressourcen frei für Innovationen

Ihre Erfahrungen und Ideen

- .....
- .....
- .....
- .....

ENTWICKLERTAG

Helge Nowak

hnowak@cincom.com

Twitter: @nowagil

Cincom

ENTWICKLERTAG

meet the SPEAKER @speakerlounge

1. OG DIREKT ÜBER DEM EMPFANG