Ich

http://www.xing.com/profile/Josef_Ornetsmueller

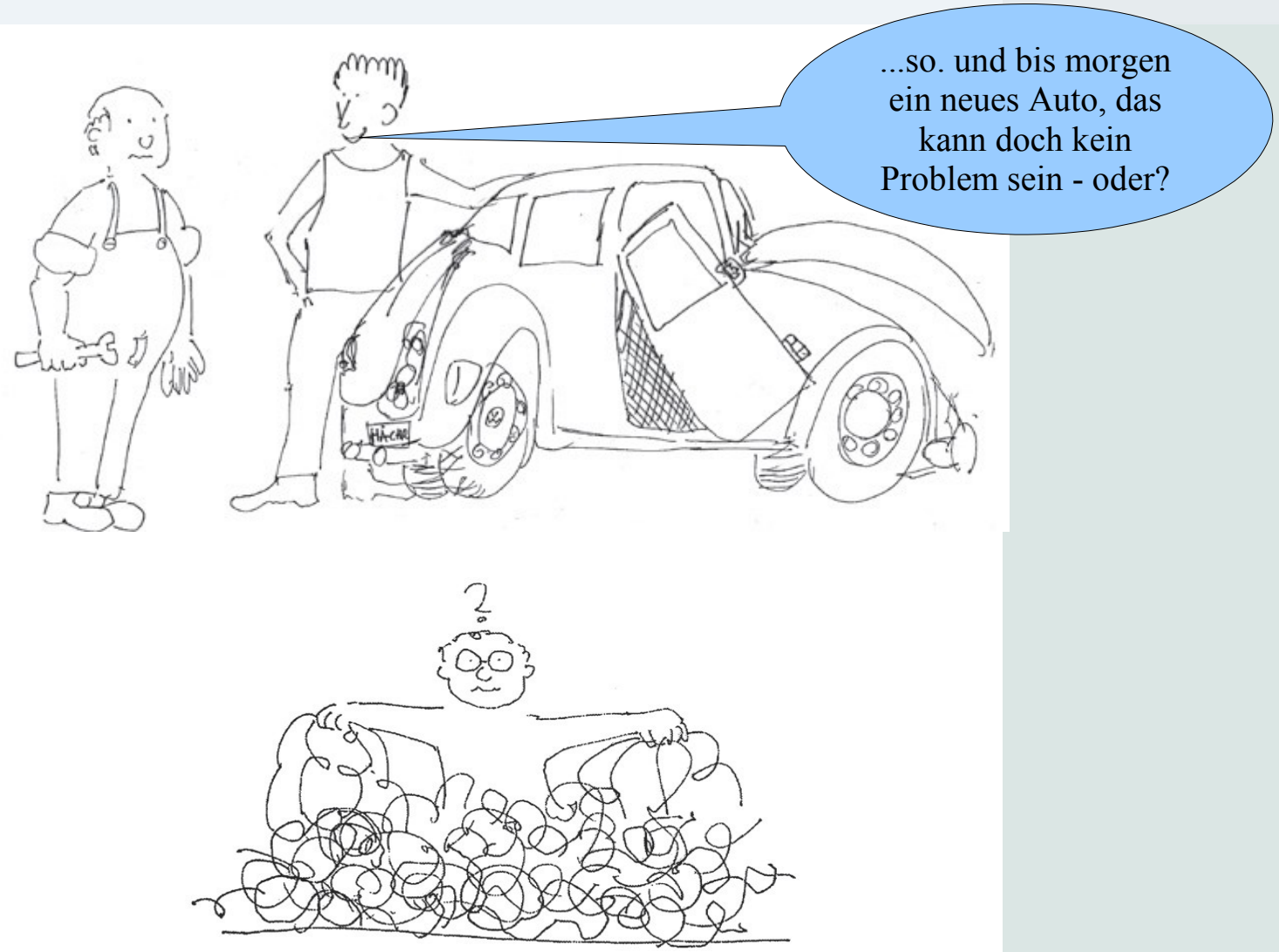
Fragen im Kontext eines Projekts

- Was ist eine gute SPU?
- Wie und was dokumentiere ich?
- Was ist eine Dreischichtarchitektur?
- Brauche ich ein automatisches Build?
- Welche Kenntnisse brauche ich?
- Woran erkenne ich eine gute Architektur?
- Wie sichere ich die Qualität?
- Was sind mögliche Erfolgsfaktoren?
- ...

Normative Kraft des Faktischen

- Gemeint ist damit, das Tatsachen, Voraussetzungen oder Sachzwänge die Regeln des Handelns diktieren.

Normative Kraft des Faktischen



Qualitätssicherung

- Die Philosophie ist es, Software so zu produzieren, dass sie weitgehend fehlerfrei ist.
  Um dies zu gewährleisten setzen wir ein breites Spektrum an qualitätssichernden Maßnahmen ein.
  Vielfältige Tools, die aus dem Open Source Bereich oder selbst angefertigt sind, nehmen uns das manuelle Testen großteils ab.
- Bereits zu Beginn des Projekts wurde eine QS-Schiene eingezogen, die aus Code Reviews, Design Tagen, gemeinsamen Analysephasen, dem CruiseControl und obligatorischen JUnit-Tests besteht.
- Komplexe, sich wiederholende Designmuster werden in FAQs und im Sourcecode direkt dokumentiert. Best Practices in Form von Entwicklerforen, ergänzen deren Einführung.

Qualitätssicherung

- Großer Wert wird auf jederzeitiges Refactoring gelegt, könnte dies nicht mehr gemacht werden, wäre es nur eine Frage der Zeit bis das System nicht mehr wartbar sein würde.
- Es sollen möglichst viele robuste, etablierte Open Source Produkte zum Einsatz kommen. Die Evaluierung soll möglichst kurz sein.
- Die Skalierbarkeit der Anwendung muss durch Hardwareerweiterungen gegeben sein. Wie weit, legen Auftraggeber und Architekt fest.
- Die Entwicklung von möglichst abgeschlossenen Komponenten kann auch extern erfolgen.
- Es sollen möglichst viele Programmfehler bereits zur Compilezeit erkannt werden, d.h. spartanischer Einsatz von Reflection.

Meine typischen Programmgrössen

- Client:
  - 300 000 LOC
  - 600 Klassen
  - 300 Reports
  - 30 Module
- Server:
  - 400 000 LOC
  - 2 500 Klassen
- Tabellen: 300

Anforderungen an Mitarbeiter

- Sehr gute Kenntnisse in der OO Programmierung
- Erfahrungen mit der J2EE
- Gute Kenntnisse in SQL
- Hibernate, JDBC sind bekannt
- Dreischichten Architekturen ein Begriff sein
- Analytische Denken wird vorrausgesetzt
- Den Anwender – den Kunden – verstehen können; social skills
- Eine rasche Auffassungsgabe haben
- …

Dokumentation, Methodik

- Jeder Entwickler arbeitet an Modulen, es gibt kein Codeownership.
- Programmierkonventionen kümmern sich um einen homogenen Code.
- Die Entwickler arbeiten vertikal, d.h. sie analysieren das Arbeitspakt, schätzen den Aufwand und Fertigstellungstermin, entwerfen und kodieren von der GUI bis zum Datenbankfeld weitestgehend selbstständig. Der horizontale Ansatz, wo ein Team den Server ein anderes den Client und wieder ein anders die DB bedient hat sich nicht als zweckmäßig erwiesen. Es wurden hier zu viele Fehlentscheidungen an den Schnittstelle getroffen, Erwartungen nicht erfüllt, es kam zu großen Wartezeiten auf neue Features, etc.

Dokumentation, Methodik

- Die Dokumentation der Analyse, des Designs, der Termine, der FAQs, erfolgt in eine Entwicklerhomepage (Wiki). Hierdurch haben örtlich externe Mitarbeiter Zugriff auf aktuelle die Informationen.
- Die Dokumentation gliedert sich in drei wichtige Bereiche.
  - Projekte: Aktuelle Beschreibung der Arbeitspakte pro Mitarbeiter mit Reihung.
  - Module: Jedes Modul hat hier sein wachsendes Pflichtenheft.
  - FAQs: Lessons learnd und Best Practices werden hier dokumentiert und direkt wenn möglich im Sourcecode verankert.

Architektur

- Die Architektur besteht aus drei Schichten die an den Schnittstellen lose gekoppelt sind.
- Der Client hat keine Business Logik und ist als 'Thin Client' ausgeführt.
- Das MVC-Pattern bestimmt die Architektur. Über Business Delegates geht der Client zu seinem Model (Sever).
- Beim UI hat sich eine Art 'Karteireitertechnik' durchgesetzt. Jedes Modul ist damit gleich aufgebaut.
- Der Client ist in Swing realisiert - AWT hat nicht die Mächtigkeit, SWT nicht die Unabhängigkeit.

Architektur

- Im Server ist die Business Logik angesiedelt. Session Facades übernehmen die Anfragen von außen, und reichen diese, wenn nötig an Business Objects weiter. Entity Beans (CMP) kümmern sich um die Speicherung.
- Die Trennung der Schichten erledigen die Business Delegates, die DTOs und die Session Facades.
- Die einzelnen Module definieren die Subsystemgrenzen und damit auch die Subsystemschneidung.
- Wie oben erwähnt, ist die Businesslogik in den Session Facades und Business-Klassen zentriert, was heißt, dass jedes andere Modul sich nur via Session Facades an andere Module wenden darf.

Architektur

- Grundsätzlich wird nur am Server auf die Datenbank zugegriffen. Wir beschränken uns auf folgende Arten des Zugriffs:
  - über Entity Beans (CMP)
  - über eine DAO-Schicht
  - über Hibernate, ein Fastlanereaderframework greift hier ausschließlich lesend zu
- Eine weitere wichtige Anforderung, ist der „kaum merkbare“ Tausch der SQL Datenbank. Damit ist es möglich, rasch „beliebige“ SQL Datenbanken – je nach Kundenwunsch - an das System „anzudocken“.

Architektur

- Die Tabellen werden auf die einzelnen Module aufgeteilt und sind dementsprechend gegliedert.
  Hierdurch ist eine klare Trennung der Tabellen und damit Verantwortung der verschiedenen Module möglich.

Automatisches Build

- Um den aktuellen Zustand der Software jederzeit überprüfen zu können, setzen wir CruiseControl ein. Dieses steuert die täglichen Builds, drei davon sind mit anschließend ausführlichen Tests ausgestattet. Kann nach dem Einchecken eines Entwickler nicht kompiliert, gebaut werden, oder ein Test schlägt fehl, informiert CruiseControl die Entwickler mittels E-Mails über diesen Sachverhalt und dokumentiert den Fehler in der Entwicklerhomepage.
- CruiseControl und die integrierten Testwerkzeuge liefern eine virtuelle Vollzeitarbeitskraft, die „gewissenhaft und unermüdlich“ Tests durchführt, Fehler aufdeckt und somit wenn man so will, auch die Qualität der Software steigert.

Automatisches Build

- CruiseControl kombiniert mit Ant steuert den regelmäßigen Ablauf der Tests auf einer unabhängigen Maschine und liefert übersichtliche Ergebnisse.
- Ant ist ein plattformübergreifendes Build-Tool, das über eine XML-basierte Configurationsdatei auch multiple Projekte steuern kann, die automatisch oder manuell aufgerufen werden können.
- JUnit, HttpUnit, und WAP-Unit, alle Open Source kümmern sich um die Webapplikationen.
- Abbot, ebenfalls Open Source, ist ein Testwerkzeug für Java Swing Applikationen und steuert den Client.

Automatisches Build

- JDepend (Open Source): Auszug aus der Originalbeschreibung: “This parser ‘traverses a set of Java source file directories and generates design quality metrics for each Java package’. It allows to ‘automatically measure the quality of a design in terms of its extensibility, reusability, and maintainability to effectively manage and control package dependencies.’
- JavaDoc (Open Source): Auszug aus der Originalbeschreibung: „Javadoc is a tool for generating API documentation in HTML format from doc comments in source code.“
  So wird eine automatische Dokumentation für alle Schnittstellen kreiert, auch im Hinblick auf eventuelle externe Entwicklungen des Clients.

Automatisches Build

- DB-Manager (Eigenentwicklung): In fast allen Datenbankprojekten wird das Update von einer ausgelieferten Version zur nächste oder n-ten Folgeversion sträflichst übersehen.
- Der DB-Manager ermöglicht es jedem Entwickler, jederzeit seine Tabellenänderungen zu Hause oder im Office durchzuführen. Des Weiteren bietet er das beliebige Heben von einer Version auf eine beliebig nächste. Kommt eine neue Datenbank wie z.B. Informix dazu, wird lediglich der DB-Manager um diese Datenbank erweitert. Ausgehend von DB-Manager werden alle integrierten Datenbanken auf den gewünschten Stand gebracht, ohne sich mit diversen Script-Sprachen, etc. und Unterschieden herumschlagen zu müssen (Java macht’s möglich).

Automatisches Build

- Der DB-Manager bietet eine grafische Oberfläche, die nicht nur die Gesamtheit an Updates ablaufen lässt, sondern auch nur bis zu einem bestimmten Zeitpunkt, so kann man einen älteren Client, der vielleicht bei einem Kunden noch im Einsatz ist, mit der angemessenen Datenbank testen und die Konsistenz ist gewährleistet.
- Über CruiseControl werden die Daten in mehreren Stufen eingespielt. Die Daten sind gekapselt und daher auch für andere Zwecke wiederverwendbar. Der Datenbankaufbau von Grund auf gewährleistet konsistente Daten.

Automatisches Build

- Ablauf der täglichen Tests
  - Startdatenbank beinhaltet alle Daten, die für eine Neuinstallation bei einem Kunden benötigt werden.
  - Aufbau der Grunddaten: Sinnvolle Daten für die hausinternen Tests (z.B. für fachliche Tests, Reporttests etc).
  - Fachliche Tests: Testen, ob das Programm fachlich richtig arbeitet (z.B. Werte überprüfen).
  - Reporttests: Hier wird getestet ob die Reportvorlagen in Hinblick auf die Serverimplementierung konsistent sind. Die Reports werden mit Echtdaten angedruckt.
  - HTTP-Tests: Testet der Web-Applikation, die mittels Webservices realisiert ist
  - WAP-Tests: Testet WAP-Apps.
  - Abbot: Den Abschluss der Test bildet Abbot, ein Open-Source-Testwerkzeug für Java-Oberflächen, das Tests direkt auf der Swing-Oberfläche durchführt und somit aufwändige manuelle Tests zu einem großen Teil ersetzt.

Automatisches Build

- Obwohl die Erstellung von Tests zeitintensiv ist, amortisiert sich ein Test nach wenigen Läufen. Es können damit einfache, kurze Abläufe und auch große Usecases simuliert werden, die nacheinander ablaufen und aufeinander aufbauende Werte benötigen und diese weiter bearbeiten und abprüfen.
- Diese Oberflächentests prüfen die Verfügbarkeit und Funktionalität von Oberflächen-Elementen und über Vergleiche von Werten und Zuständen die inhaltliche Korrektheit der Applikation, zusätzlich laufen diese Tests in erhöhtem Tempo ab, die die Geschwindigkeit des Programms ausschöpfen und jeden menschlichen Tester an Genauigkeit und Schnelligkeit überflügeln.

Automatisches Build

- Eine der Herausforderung für die Entwickler ist, die Tests flächendeckend zu organisieren, damit mit einer sinnvollen Anzahl möglichst alle relevanten Elemente und wichtigen Werte der Oberfläche permanent zu kontrollieren. Fehler und Veränderungen in der Anzeige und Funktionalität werden innerhalb von wenigen Stunden aufgedeckt und der betroffene Mitarbeiter wird mittels E-Mail verständigt. In der Folge wird der entsprechende Bereich der Software manuell von dem Entwickler kontrolliert, Fehler ausgebessert und durch abgeänderte Tests geprüft.
- Ausblick auf zukünftige automatische Multiclient-Tests, die den Zugriff mehrerer Clients in einer produktiven Umgebung simulieren.

Erfolgsfaktoren

- Durchgehende Architektur des Systems
- ein „Playing“ Architekt
- schnell technische als auch organisatorische Entscheidungen von den passenden Teammitgliederen getroffen
- motiviertes kleines Team
- „Psyche“ des Auftraggebers
- schwierigen Phasen häufiges, für den Auftraggeber passendes Controlling vorsehen
- zentral gepflegte, verbindliche Entwicklerhomepage etablieren.
- Musterlösungen im Sourcecode zu dokumentieren
- Jederzeit refactorfähig zu sein

Erfolgsfaktoren

- Einbindung von Studenten für Diplom- und Baccelaureatsarbeiten, spart Budget und gibt neue Impulse
- virtuelle CruiseControl-Mitarbeiter einführen
- Softwareproduktion so anlegen, dass ein „Zuhausearbeiten“ zur Freude wird, d.h. kleine „PC-Rüstzeiten“
- gemeinsam erarbeitet QS-Maßnahmen zu dokumentieren und zu leben
- UML: Sequencediagramme; Statediagramme; Klassendiagramme
- VHIT ist super!
- Team vs. Einzelkämpfer; Was bin ich?
- Am Beginn: Beobachten und Schlüsse ziehen.

Erfolgsfaktoren

- Team organisieren; Admins benennen; Jourfix einführen; Wann Wer Was.
- KISS: Keep it simple, stupid (Dieses Designprinzip beschreibt die möglichst einfache, minimalistische, sowie im Nachhinein leicht verständliche Lösung eines Problems, welche meistens als die optimale Lösung angesehen wird.)
- Eine Lösung, von allen Seiten betrachten; 'herumgehen'; Konzept gegenchecken; zuletzt codieren.
- Statefull vs. Stateless wann was?
- 2-Schicht – 3-Schichtarchitekturen; Vor- Nachteile?
- Schätzverfahren: „Blitzen“.

Erfolgsfaktoren

- Kleine Übungsprojekte machen:
  - Swing GridbagLayout; JNDI; Spring; Servlet; MVC; Sun Tutorials; DB-Zugriffe: JDBC, Hibernate,
- Ziel setzen und überprüfen
- Analyse: Papier, Bleistift, Radiergummi
- Was vom wie trennen
- CRUD, EVA
- Weiterbildung:
  - Lesen, lesen lesen
  - Code lesen
  - OOP München
  - JAX besuchen
  - Java Report oder ? lesen
  - Management, Führung, Teamarbeit
  - In Newsgroups mitdiskutieren

Kontakt

eMundo GmbH
Petersbrunnstraße 19
A - 5020 Salzburg
Tel: 0662 847351
Fax: 0662 847351-99

josef.ornetsmueller@emundo.at

Fragen