Replace a bank-wide core application

Lessons learned

Nikolaos Kaintantzis
nikolaos.kaintantzis@zuehlke.com

Twitter: @xnka

Das Altsystem

- Data Warehouse mit wichtigen Kennzahlen
- Jeder Mitarbeiter nutzt das System
- Schnittstellen zu internen und kommerziellen Produkten (z.B. Matlab und Excel)
- Eigene Skriptsprache und Compiler

Technologien:

- PL1-Kern
- Visual Basic Oberfläche

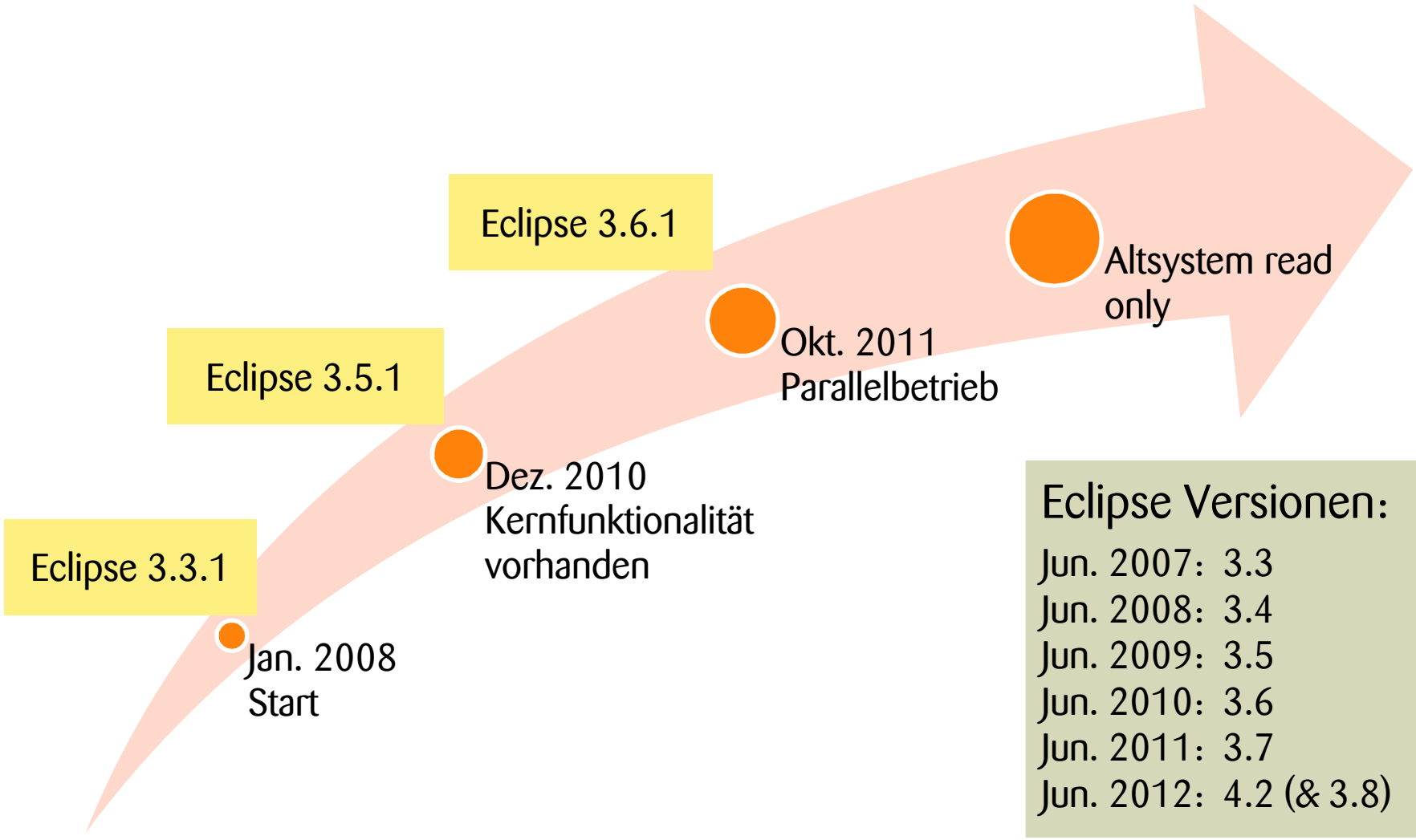
Das neue System

- Viele Stakeholder mit unterschiedlichen Wünschen
- Metadaten-Modellierung und Retrieval sind Systembestandteil
- Kern für weitere interne Produkte
- Neue Skriptingsprache, basierend auf Groovy

Technologien:

- JEE-Server
- Eclipse RCP Client
- Groovy

Das Projekt



Eclipse Versionen:

Jun. 2007: 3.3
Jun. 2008: 3.4
Jun. 2009: 3.5
Jun. 2010: 3.6
Jun. 2011: 3.7
Jun. 2012: 4.2 (& 3.8)

Warum Eclipse RCP? (Argumentation Stand 2008)

- Wunsch nach Plattform
  - Aufgeräumte Oberfläche
  - Vorhandene Funktionalität nutzen
- Schlechte Kundenerfahrung mit WPF
- Andere Java-Alternativen schienen nicht zukunftssicher
- Web-Lösungen befriedigen Interaktionsbedürfnisse nicht
- Eclipse bei einigen Benutzern (Strukturverantwortliche) bereits anderweitig im Einsatz

Erste Eindrücke vom Fach resp. Produkt-Owner

zühlke
empowering ideas

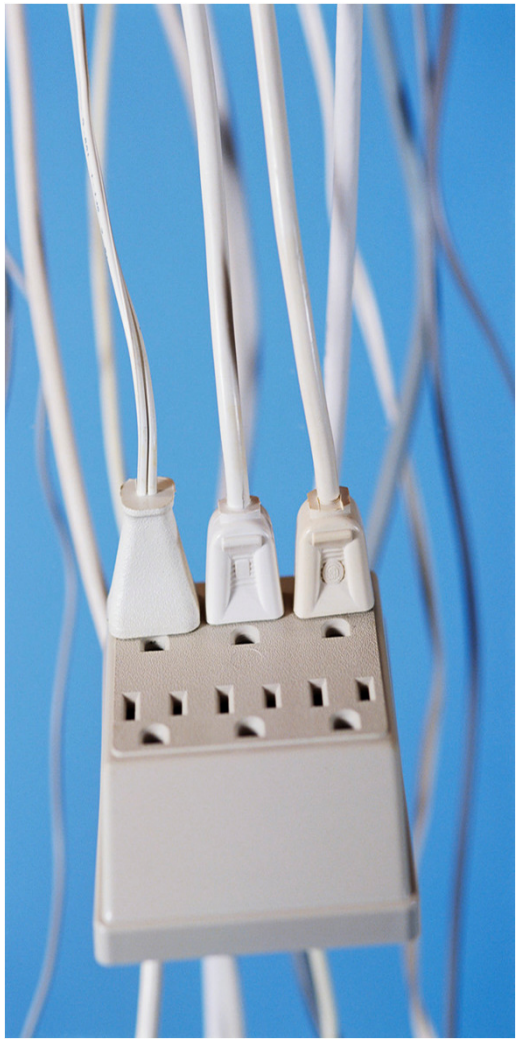
Nach 6 Wochen Client-Entwicklung:

Gründe:

- Fensteranordnung
- Baumdarstellung
- Erste XML-& UI-Editoren waren schnell da

Wieviele Plugins brauchen wir?

zühlke
empowering ideas



Beim Projektstart werden Plugins falsch geschnitten

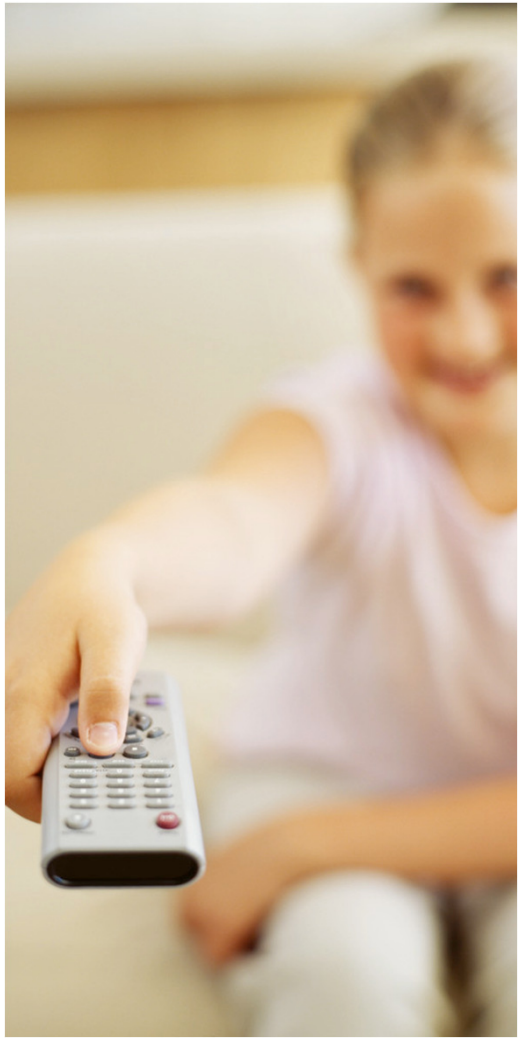
Ansatz: Benutzersicht einnehmen

- Was soll zur Laufzeit austauschbar sein?
  → in der Regel: nichts
  - 3 Plugins (Libs, App, Hilfe)
  - 1 Feature (mit Produkt)
  - 1 Fragment (Test)

Ansatz: Wiederverwendung

- Welche Funktionalitäten werden durch andere Applikationen wiederverwendet?
  → eigene Plugins und Fragmente hierfür

Eclipse RCP und Usability

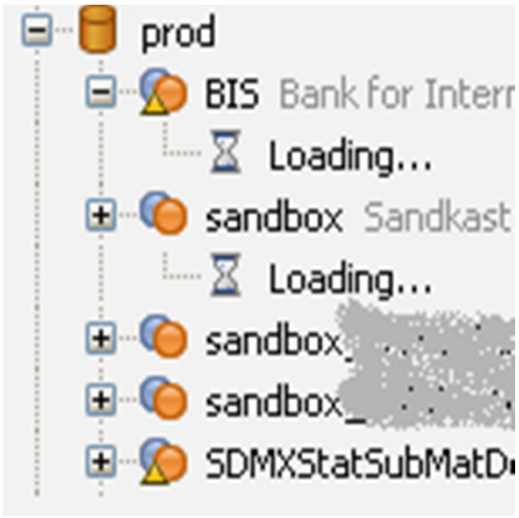


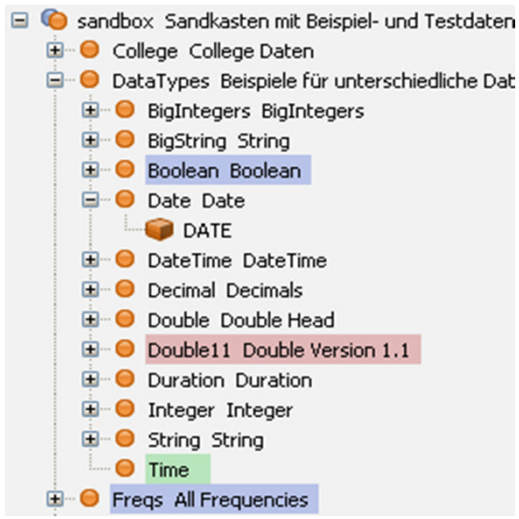
Hauptproblem (abhängig von der Benutzergruppen):

Flexibilität vs. Benutzerführung

- Umpositionierung von Editoren und Views kann nicht von jedem Benutzer rückgängig gemacht werden
- Zu viele Operationen ersichtlich / möglich
- Perspektiven sind technisch motiviert und werden nicht verstanden
  → Perspektiven müssen von der Arbeit der Benutzer getrieben sein
  → Mehr Redundanz zwischen den Perspektiven
- Wie soll ein Arbeitsfluss abgebildet werden?

Die Limits





Grundproblem: Eclipse war gedacht für Entwicklungsumgebung

Folgerung: schwach in allem anderem

- Tabellen (Excel-like)
- Loading-Knoten in Bäume
  (weil Datenbeschaffung lange dauert)
- Farbe in Bäumen (v.a. am Anfang)
- Formulare (v.a. am Anfang)
- (cooles Aussehen)

Kritikpunkte

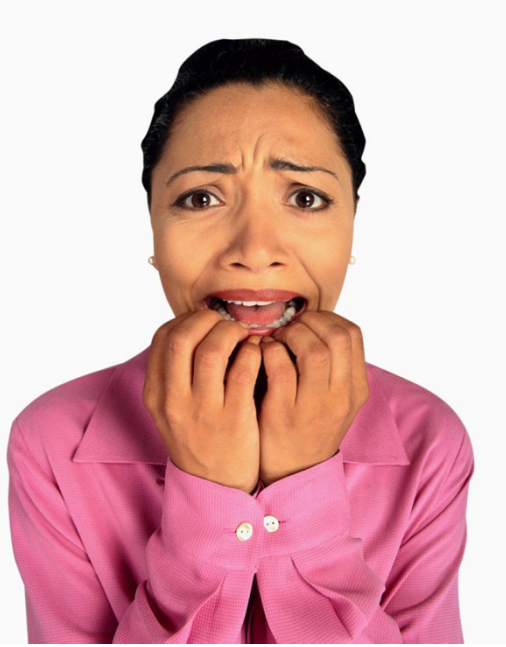
- APIs organisch gewachsen
  - Redundanzen
  - Inkonsistenzen
- Codequalität und Javadoc in einigen Bereichen wünschenswert
- Nicht immer werden Interfaces benutzt
- Sich ändernde UIs sind aufwändiger zu implementieren als in Swing
- Welche Style-Bits machen Sinn für diese und jene Komponente?

Albträume

zühlke
empowering ideas

Headless-Build mit Maven (vor Maven 3 und Tycho)

- Viel Zeit in eigene Maven-Plugins investiert
- Umgestellt auf Gradle



UI-Testing (Ist nicht nur ein Eclipse RCP-Thema)

- Extrem aufwendig / zu wenig Ressourcen
- Zu häufig fehlgeschlagene Tests ohne richtigen Fehler
  - Timing Probleme (Lösung: besseres Tool)
  - Ab und zu unvorhergesehene aufpoppende Jobs-Monitor-Dialoge

Groovy Integration

Die DSL

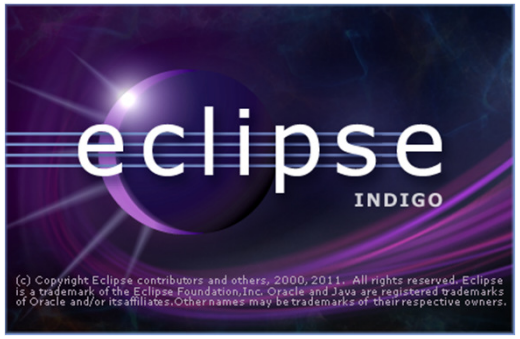
- Fast alle UI-Operationen sind auch über Scripting zugänglich
- Interne DSL auf Basis von Groovy
- Zuerst: Facade mit Methoden und Wrapper über Typen und Operationen
- Heute: Objektorientierte Bibliothek und DLS-Tricks

Groovy-Plugin verbessert sich immer mehr (dank Teilnahme an Foren)

- Probleme z.B. bei Code Completion (u.a. Performance)
- Java-Doc-Fehler bei überladen Methoden

Groovy-Plugin ist Hauptgrund dafür, die Eclipse RCP-Version zu aktualisieren

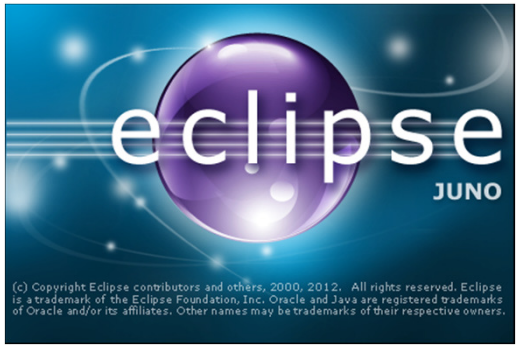
Welche Eclipse-Version?



Im Moment 3.6.1 (Version von Herbst 2010)

Abhängigkeit von Libraries

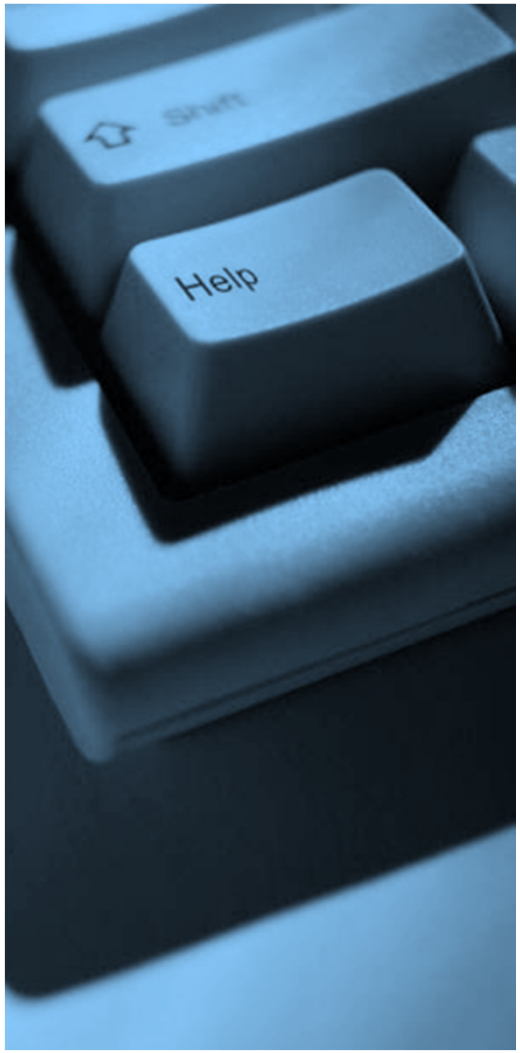
- Neuestes Groovy-Plugin unterstützt offiziell 3.7 und 4.2
- Nebula-Grids (Tabellen) sind für Version 3.6, 3.7 und 3.8



Eclipse Juno bevorzugt 4.2 vor 3.8 – unserer Meinung nach ein Jahr zu früh

- Viele UI-Bibliotheken sind für 4.x nicht verfügbar

Hilfesystem und Dokumentation



Hilfe ist eigenes Plug-In

Tool: Docbook (XML)

Mit XSLT wird die Eclipse-Hilfe generiert

Code-Snippets sind ausführbar

Einstieg über

- Hilfe-Menü
- Links im Javadoc der Scripting-Sprache

Fazit

1. Plattform nimmt dem Entwickler viel ab
   → erste schnelle Erfolge verblüffen das Fach
2. 3.x ist eine stabile und etablierte Plattform für Rich Clients
3. Tabellen (etwa so wie Excel) sind teuer
   → Framework-Evaluation notwendig
4. Literatur und Foren sind technisch und nicht business- oder benutzergetrieben
   → ein Entwickler/Architekt sollte häufig Benutzersicht einnehmen
5. Aktives Mitmachen in Foren sehr empfehlenswert
6. Maven nur ab Version 3 (mit Tycho) und nur dann, wenn man entsprechend den Vorschlägen von Maven arbeitet, sonst Gradle

Fragen?

zühlke
empowering ideas