

# Von Tigern und Indien

Ein Offshoring-Erfahrungsbericht

Dr. Andreas Kotulla





# **Keine Tiger in Indien**

**Tagebuch einer Offshoring-Idee**

# Outsourcing

- Geringes Risiko
- Erhöht Geschwindigkeit
- Erhöht Qualität
- Erhöht Wettbewerbsfähigkeit und reduziert Kosten
- Milliardengeschäft

Wipro chief –  
Indian companies heading for the top  
Difference between Indian outsourcers and big services  
companies blurring

European boom for tech  
service firms  
...revenues exceed €2bn during 2006...

Capgemini boosts  
presence in India  
The deal values Kanbay at \$1.25bn

LogicaCMG opens offshore  
BPO centre in the Philippines  
Manila facility to kick off with around 200  
staff...

Qantas outsources IT to India  
in \$147m deal TCS and Satyam share the spoils

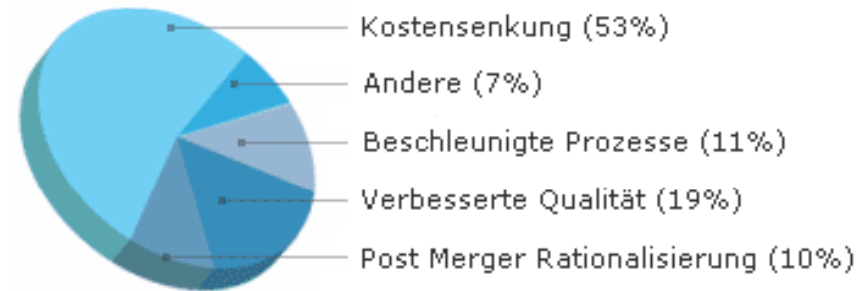
BT offshoring \$1bn in support to India  
Tech Mahindra to support telco and its customers

Vodafone outsourcing deal: And the  
winner is...  
...spent £560m on both last year

# Offshoring-Motivation

## Erwartung:

- Reduzierung Kosten
- Innovation
- Fokus auf Kerngeschäft
- Produktivitätssteigerung
- Erhöhung Wiederverwendbarkeit
- Schnellere Markteinführung
- Höhere Qualität
- Erhöhung Flexibilität
  - Inhouse-Ressourcenknappheit country
  - Erhöhung Entwicklungskapazität



Deloitte Research, Dezember 2003

Warnung: >70% der Offshore-Outsourcing-Projekte enden „abnormal“!  
(Deloitte Research)

# Beispiel Russland

- Anforderungen und Fachkonzepte Inhouse
  - Festpreisangebot an russische SW-Entwicklungsfirma
  - Geschätzter Entwicklungsaufwand: 10 Mannjahre, Dauer: 1 Jahr
  - Monatliche Treffen
  - Moderne Techniken (C++, ATL, ActiveX,...)
  - Ziel: Anschließende Wartung wieder Inhouse
- ⇒ Ergebnis: 90% der Anforderungen  
15% mehr Budget / Zeit



# Aber: Inhousing (fast) unmöglich

- Zusätzliche libraries verwendet (STL, Rogue Wave)
- Nicht buildbar (proprietäre MAKE-Umgebung)
- Dateiabzug nicht vollständig
- Russische Code-Kommentare

```
Private Sub Up_Str_App()  
TM = Time           'Прочитать текущее время  
TTMS = Len(TM)     'Длина строки с временем  
TTM = Mid$(TM, 7, 2) 'Выделить секунды  
  
For i = 0 To TTM     'Повторить цикл столько  
                    раз, сколько секунд  
Next i  
'-----  
' Заполнить верхнюю строку массива field(i, 5)  
For i = 0 To 7  
    field(i, 5) = Int((20 * Rnd) + 1)  
Next i  
'-----  
'Заполнить верхнюю строку номиналами, копируя  
их из Label56  
  
Field_Fill 'Перерисовать значения на экране, с  
новыми значениями верхней строки  
End Sub
```

# Beispiel Rumänien



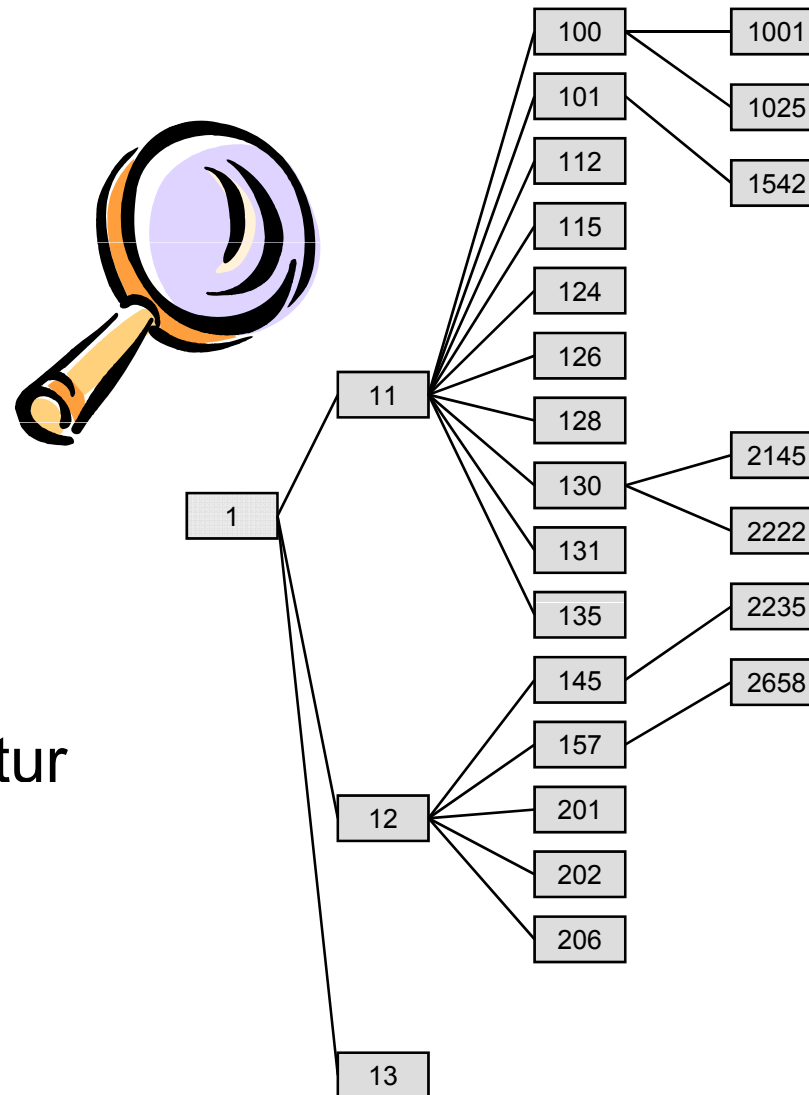
- Anforderungen und Konzepte Inhouse
- Intensiver, regelmäßiger Mitarbeiteraustausch
- Festpreis
- Geschätzter Entwicklungsaufwand: 4 Mannjahre
- Auftrag: Zukunftsfähiger JAVA-Auskunftspilot mit einfacher GUI, plattformunabhängig, hoch performant
- Ziel: Anschließendes Insourcing

⇒ Ergebnis:      100% der Anforderungen  
                         0% Budget bzw. Zeitüberschreitung



# Aber:

- GUI vollständig in Macromedia Flash geschrieben
- Lieferung inkl. völlig unbenutzter 3rd-pty-Produkte
- Keinerlei Systemübersicht
- Kryptische Ablagestruktur (nur nummeriert)

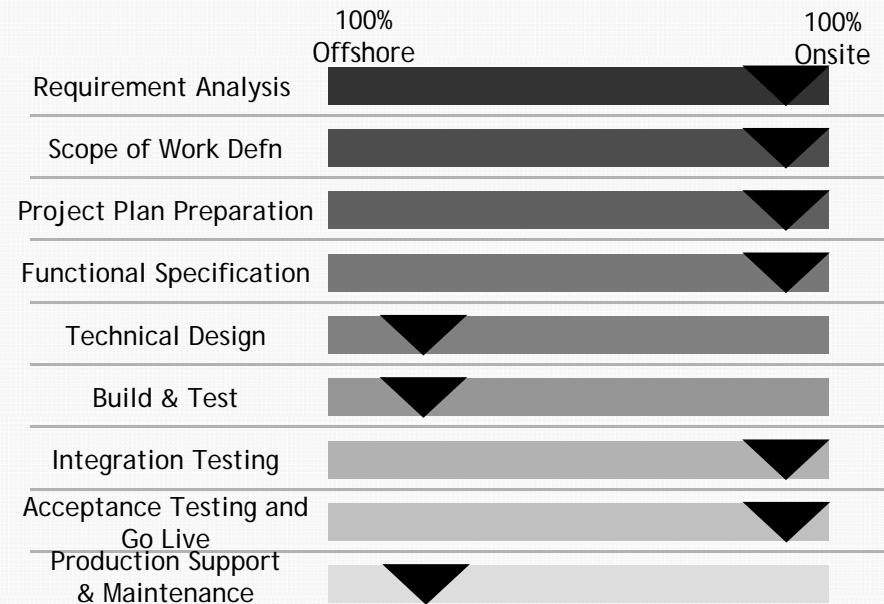


# Modelle

## Softwareentwicklung

- Vor-Ort Unterstützung
- **Vor-Ort/Offshore-Mix**
- Vor-Ort-Brückenkopf, Offshore-Entwicklung
- Inland-Brückenkopf, Offshore-Entwicklung
- Isoliertes Offshoring-Team
- Schlüsselfertige Offshoring-Lösung
  
- Auslagerung der Wartung zu einem Offshore-Partner

Typical Onsite Offshore Activity Mix  
(Application Development Projects)



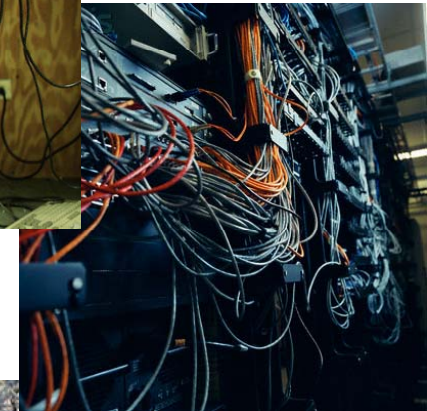
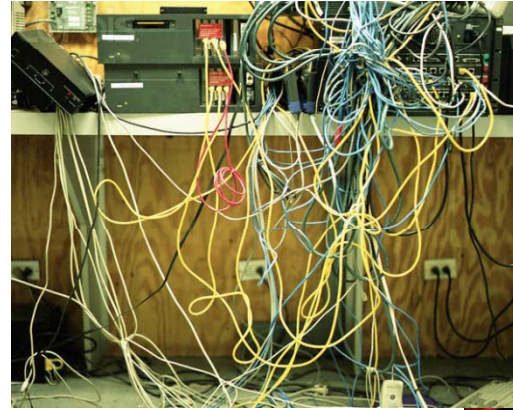


# Wie sieht der Code aus?

Erwartet

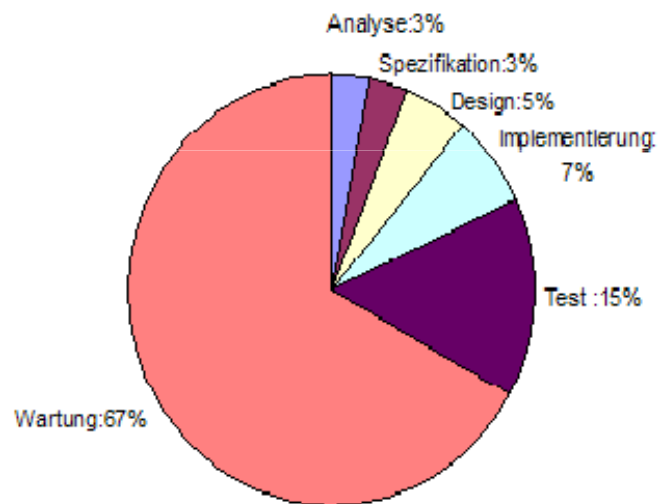


Geliefert?

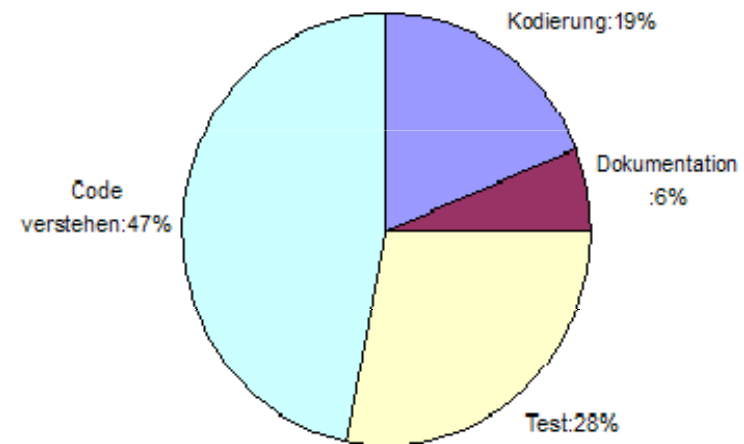


# SW-Kosten gesamter Lebenszyklus

Aufwände: \*



Wartungskosten: \*



\* Accenture, Microsoft, SQS, Balzert

# Brennende Fragen



---

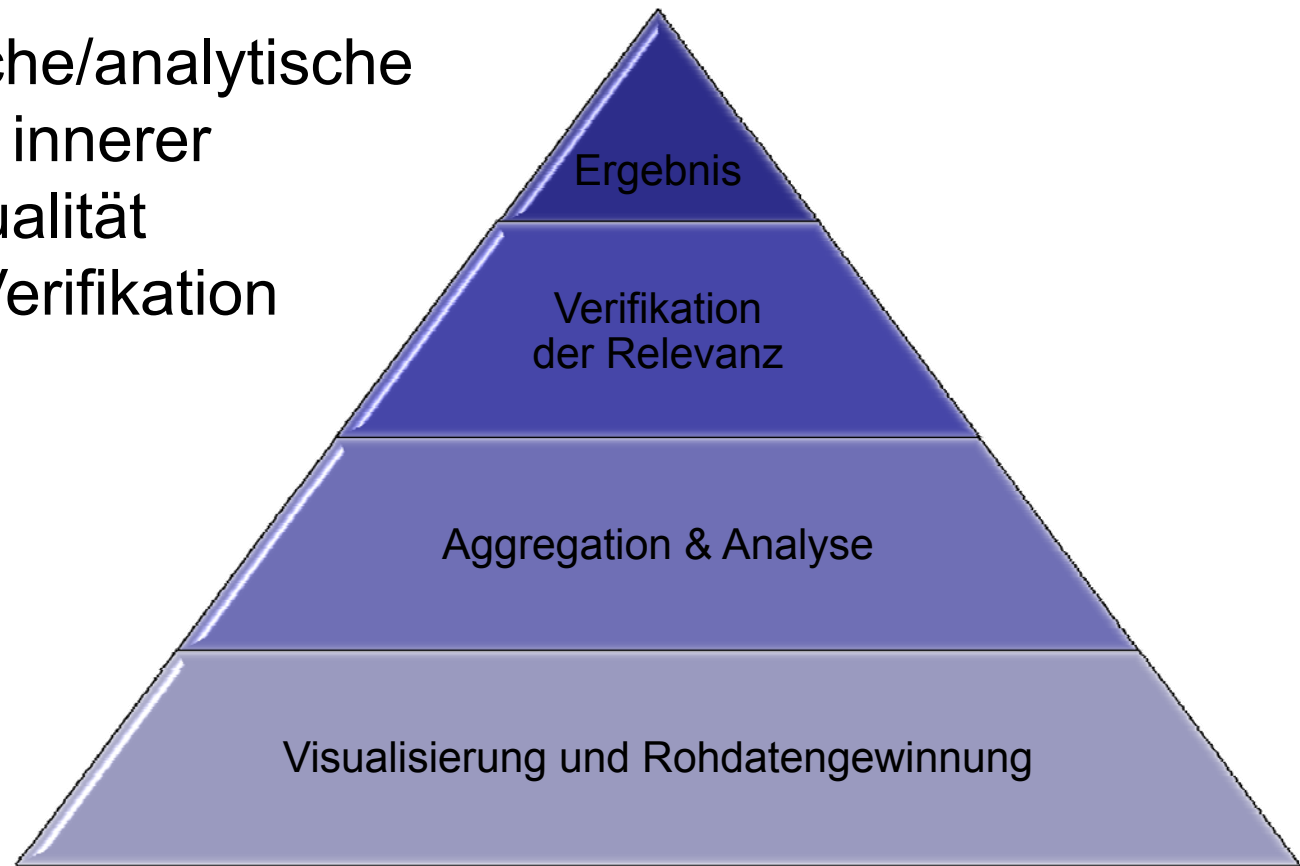
- An welchen Krankheiten leidet mein System? Kann ich diese vorab diagnostizieren?
- Was bedeutet das bezüglich Wartung, Fehleranfälligkeit, Verständlichkeit?
- Wie hoch sind die Folgekosten? Kann ich diese begrenzen?
- Wo sind die Schwachstellen, die Probleme bereiten werden?
- Wie kann ich mein System verbessern?
- Stimmt meine Implementierung mit dem Entwurf überein?

Die Antwort liegt im Quellcode!

# Bewertungs-Pyramide

Ansatz:

1. Automatische/analytische Bewertung innerer Softwarequalität
2. Manuelle Verifikation



# Beispiel: Gott-Klasse



## Beschreibung

Eine Klasse beinhaltet die gesamte Intelligenz des Systems. Diese „Gott-Klasse“ führt die Mehrheit der Arbeiten selbst durch, nur wenige triviale Aufgaben werden an andere Klassen delegiert.

## Problem

Schwer wiederverwendbar. Schwer verständlich. Grundidee der Objektorientierung verletzt.

## Erkennung

- Groß und komplex
- Massive Nutzung von Daten anderen Klassen (direkt und durch Access-Methoden)
- Geringe Kohäsion in Methoden der Klasse (Methoden gehören nicht logisch zusammen)

## Refactoring

- **Aufsplitten der Klasse**

# Beispiel „Shotgun surgery“

## Beschreibung

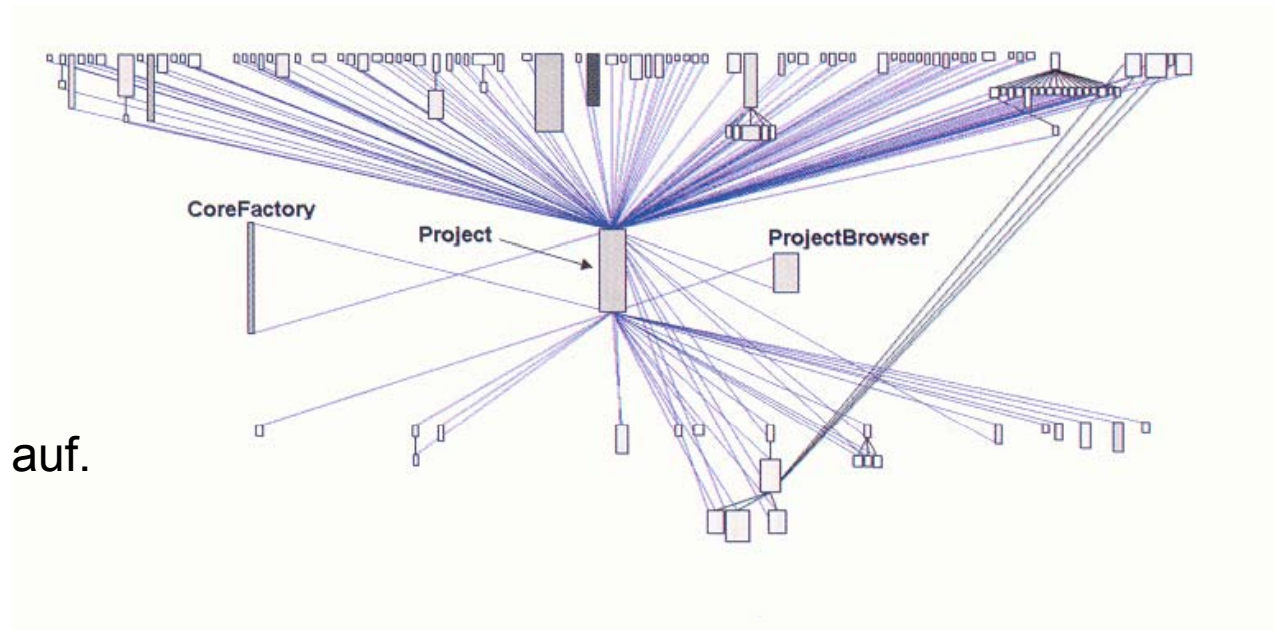
„Eingehende“ Kopplung:  
Sehr viele verschiedene  
Module rufen eine Funktion auf.

## Problem

Eine Änderung in dieser Methode zieht Änderungen in allen aufrufenden  
Funktionen nach sich, Wartung wird erschwert.

## Erkennung

- Aufruf einer Methode von sehr vielen verschiedenen anderen Methoden
- Aufrufende Methoden gehören zu vielen verschiedenen Klassen



# Komplexität

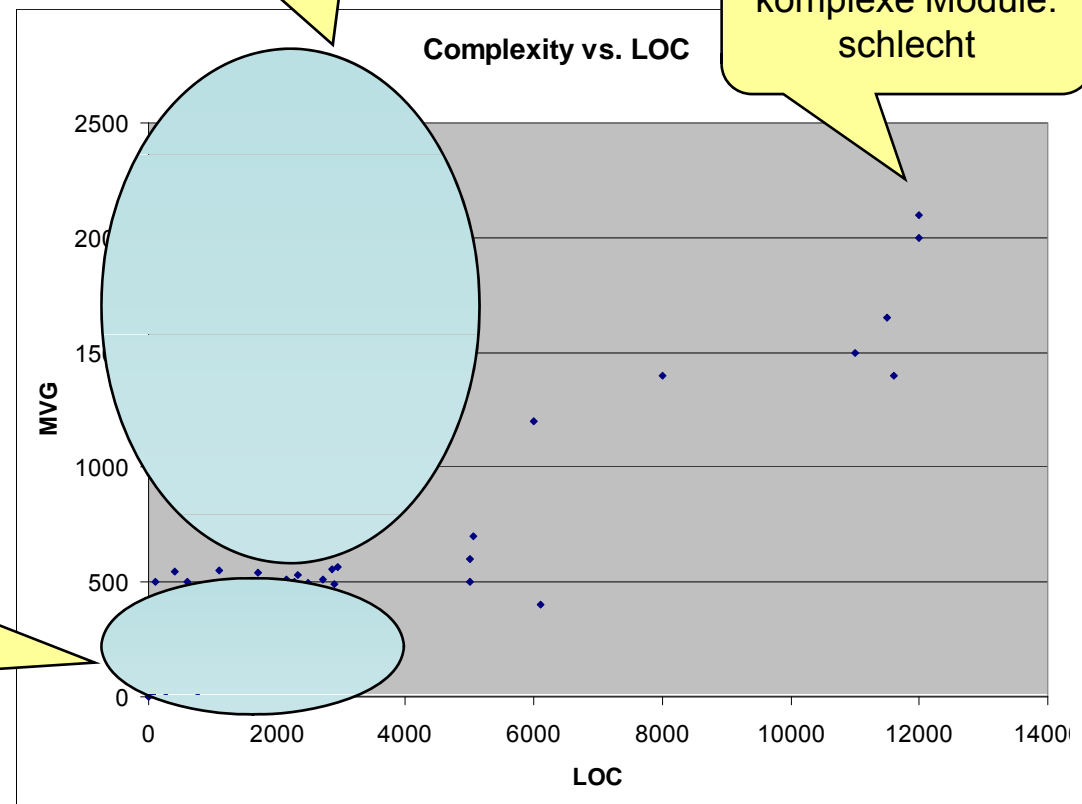
Komplexer Code ist schwer verständlich, schwieriger zu warten und enthält statistisch gesehen mehr Fehler. Es ist daher wichtig die komplexen Teile einer Software zu kennen, diese besonders ausführlich zu **testen**, gut zu **dokumentieren** und die Komplexität gut zu **kapseln**.

Top 5 Module Name	MVG
Modul 1	2923
Sample2	1666
Modul 3	1321
Modul 4	1045
Beispiel 5	

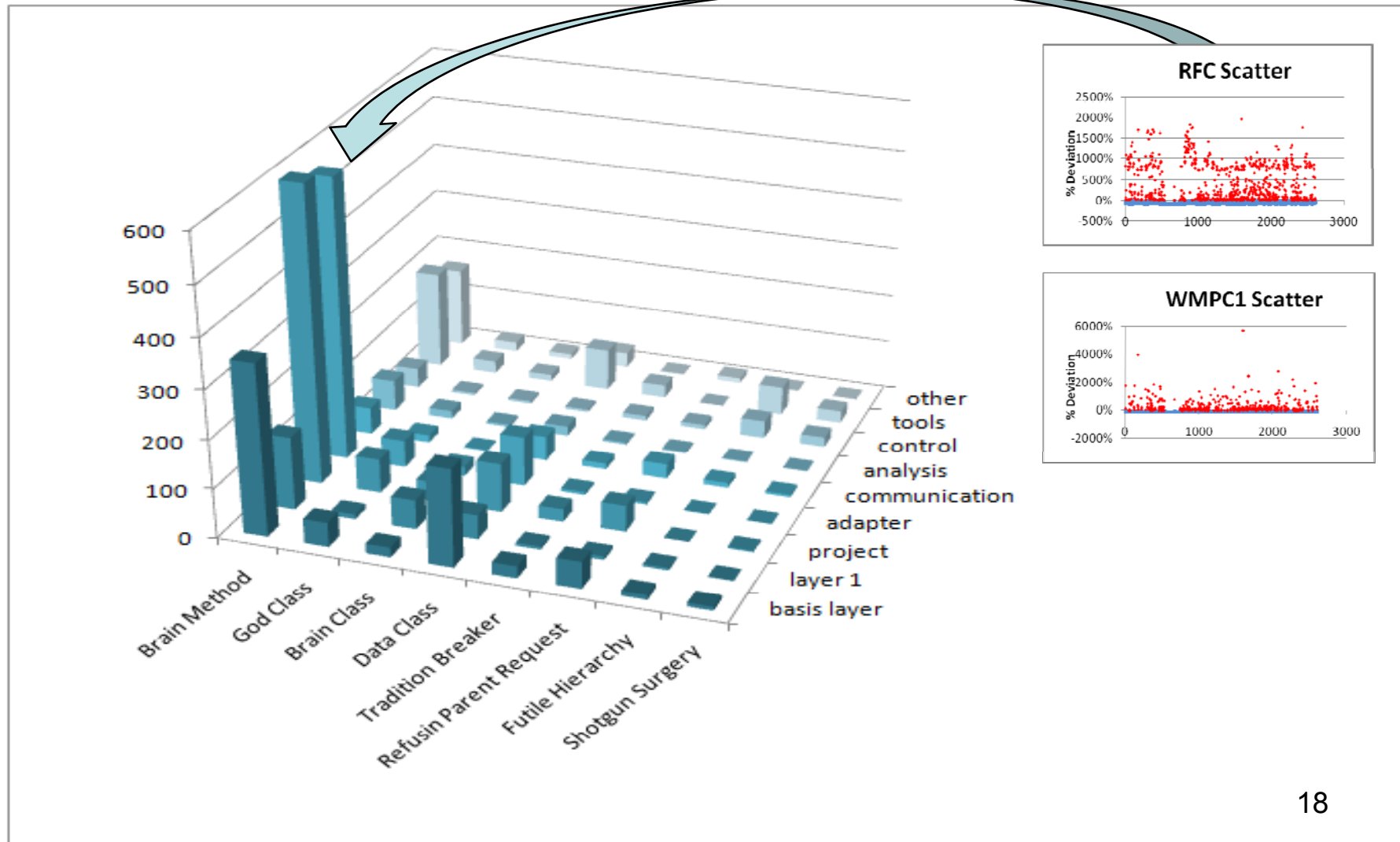
Werte in diesem Bereich:  
langsames/lineares  
Wachstum: eher gut

Viele Werte in diesem  
Bereich indizieren  
exponentielles Wachstum:  
schlecht

Sehr lange &  
komplexe Module:  
schlecht



# Antipatterns Übersicht



# Auswirkung

---



- **Wartungskosten**  
Komplexe Klassen, hohe Kopplung und wenige Kommentare machen es schwierig, den Quellcode zu verstehen. Dadurch steigt der Aufwand für Wartung, zukünftige Erweiterungen oder der Fehlersuche. Es wird schwierig, neue Entwickler schnell einzuarbeiten. (Vor allem in Festpreisprojekten kann durch die Möglichkeit, Code schnell ändern oder adaptieren zu können, die Marge vergrößert werden).
- **Qualität**  
Je komplexer die Klassen (Größe und zyklomatisch) und je höher die Kopplung, desto höher wird der Testaufwand und desto höher die Wahrscheinlichkeit, dass Probleme in der Testphase nicht gefunden werden. Während der Supports wird es schwieriger Problemursachen zu identifizieren. Stabilität ist schwieriger zu erreichen.
- **Time to market**  
Der Aufwand um komplexen Code zu erweitern ist höher, die Entwicklung langsamer; Auswirkungen der Änderungen auf Architektur und Funktionalität sind für die Entwickler schwer zu verstehen. Seiteneffekte in Gottklassen und –methoden sind schwer vorhersagbar aber sehr wahrscheinlich.
- **Wiederverwendbarkeit**  
Gottklassen und –methoden sind schwierig wiederzuverwenden, Code wird dupliziert oder noch einmal neu entwickelt.

# Resümee

---



Mit:

- Sorgfältiger Auswahl von Partner und Projekten
- Ausreichend Aufschlag auf Aufwandsabschätzungen
- Geduld und Ausdauer, ausreichend Anlaufzeit
- Modell: “Verlängerte Werkbank”
- Genauen Vorgaben und Kontrollen bezüglich
  - Prozess & Meilensteinen
  - **Qualitäts-KPIs (insb. “innere” Code-Qualität-Messung)**
  - Team-Mitarbeitern
  - Mikro-Management

konnte Offshore-Outsourcing erfolgreich durchgeführt werden



- Beratung Offshoring  
Prozesse, Partner, Voraussetzungen
- SW Analysen & Assessments
  - Transparenter Blick auf die Software und ihrer Struktur – schnell und kosteneffektiv
  - Sofortige Identifikation verbesserungsfähiger Bereiche, Früherkennung von Mängeln
  - Momentaufnahme als Basis zur Messung von Softwarequalität und deren Änderung über die Zeit

# Vergleich



## Inner quality assessment

### Traditional approach

Size	1.000.000 LOC
Review-Speed	125 LOC / H
Effort	8000 HOURS
*Effort	<b>4,5 years</b>

\*Assumption 220 working days / year, 8 h / day

### Telisis Method

Size	1.000.000 LOC
Preparation	2 DAYS
On site data gathering	10 DAYS
Analysis	6
Resume / Recommendations	2
Effort	<b>20 days</b>

« Effort level »



# Ausblick



---

Heutige Kernprobleme in der SW-Entwicklung:

- Steigende Komplexität
- Explodierende Wartungskosten
- Mangelhafte Qualität

Zukunft:

- Automatisierung der Software-Entwicklung – Einführung Softwarefabrik, die den Outsourcing Trend optimiert und standardisiert.
- Dadurch starke Kosteneinsparung
- Volle Transparenz des Codes
- Garantierte Qualität

# Kontakt!

---



Dr. Andreas Kotulla  
Geschäftsführer Telisys GmbH  
E-Mail: [andreas.kotulla@telisys.de](mailto:andreas.kotulla@telisys.de)  
Tel.: 02241-879-4234  
Mobil: 0176-48320404

# Meinungen von Kunden

---



»Ich war erstaunt, dass nach nur vier Tagen vor Ort die Problemstellen identifiziert waren. Einen Großteil der Empfehlungen haben wir bereits umgesetzt.«

Leiter Softwareentwicklung, Verteidigungsindustrie

»Sehr zuverlässig.«

Abteilungsleiter Softwareentwicklung

»Ich hätte nicht gedacht, dass diese Methode so detailliert und vollständig ist. Ich will dies für mehr Projekte.«

Bereichsleiter

»Dies ist eine komplett neue Dimension«

Direktor System Consolidation, führender Netzbetreiber in Deutschland