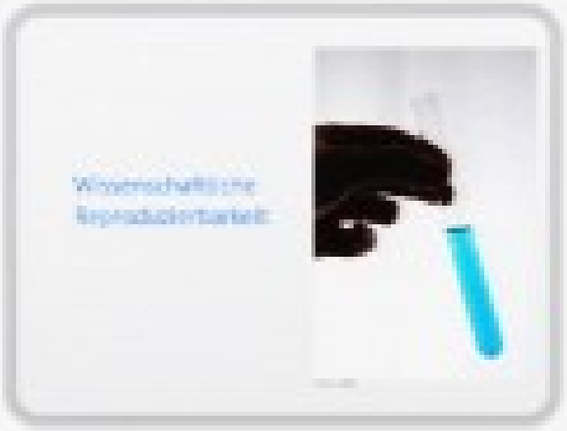

TNG TECHNOLOGY CONSULTING

# Agile Softwareentwicklung: Das soll sich wirklich lohnen?!

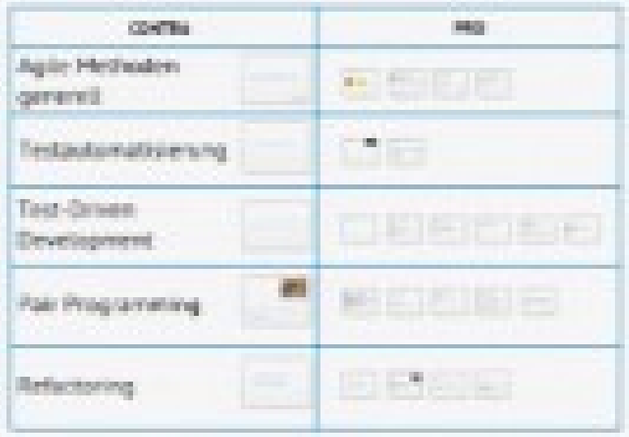
Eike Reinel
TNG Technology Consulting GmbH

TNG Big Techday 2014, 23.5.2014

# Agenda

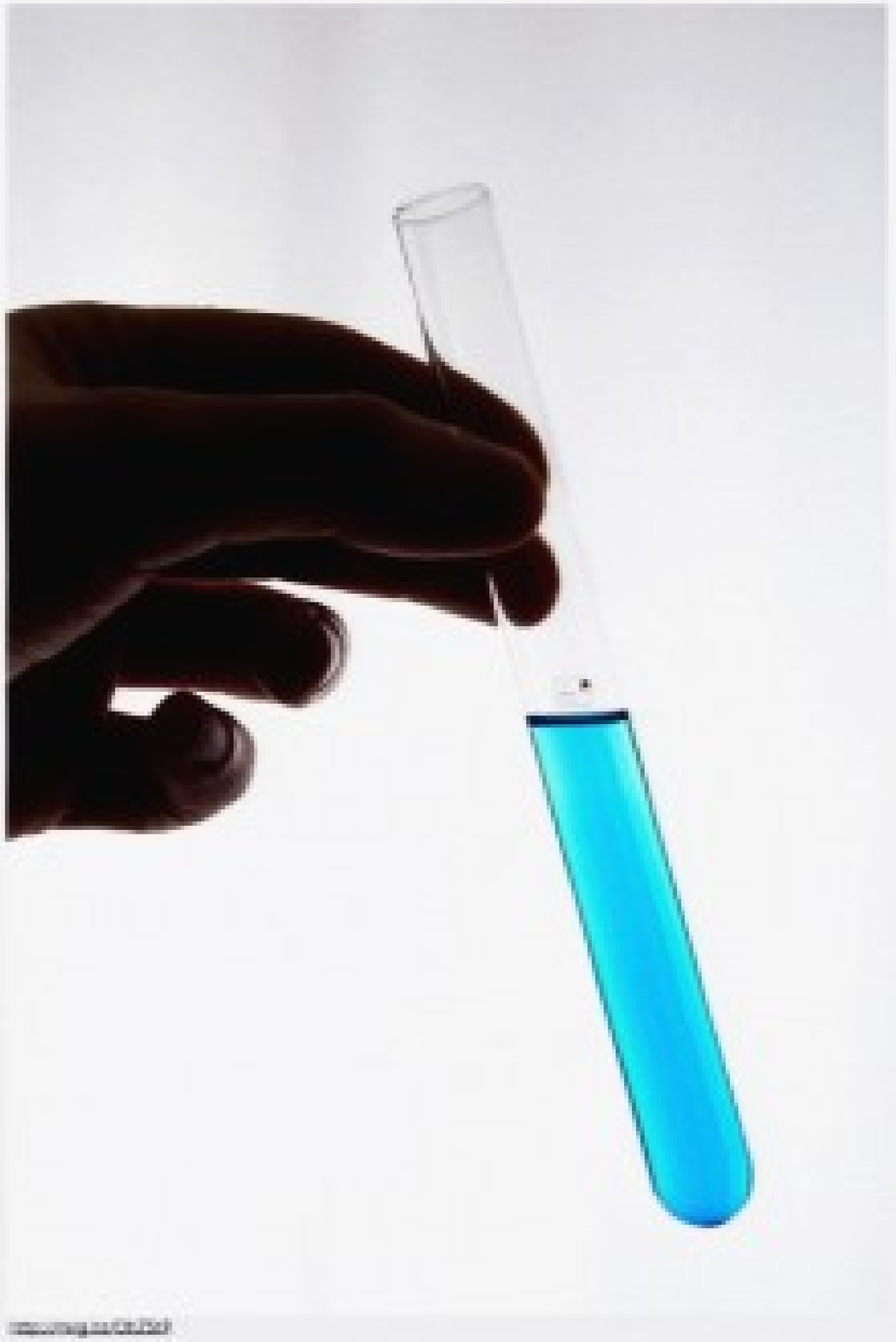






| | |
|---|---|
| Agile Methoden generell | |
| Testautomatisierung | |
| Test-Driven Development | |
| Pair Programming | |
| Refactoring | |


# Wissenschaftliche Reproduzierbarkeit



## Versuche unter Laborbedingungen sind sehr schwierig

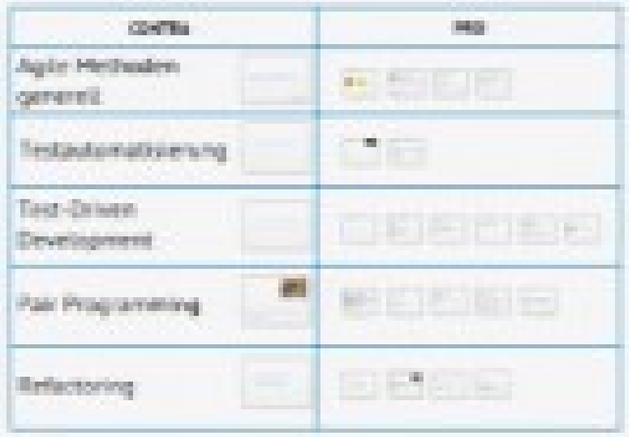

- Teams  und Teammitglieder sind unterschiedlich, Vergleichbarkeit ist nicht unbedingt gegeben

- Das selbe Problem ist durch das selbe Team nicht wiederholt mit unterschiedlichen Methoden ohne Vorkenntnisse lösbar

- Große Projekte unter "Laborbedingungen" zu untersuchen ist zu teuer (à la "Der Termin" von Tom DeMarco)

# Agenda





| | |
|---|---|
| Agile Methoden generell | |
| Testautomatisierung | |
| Test-Driven Development | |
| Pair Programming | |
| Refactoring | |



# Agile Softwareentwicklung - Das kann sich doch nicht lohnen!

| CONTRA | PRO |
| --- | --- |
| Agile Methoden generell | |
| Testautomatisierung | |
| Test-Driven Development | |
| Pair Programming | |
| Refactoring | |

"Bei agilen Methoden weiß ich zu Beginn nicht, ob das Projekt "in time, budget und scope" fertig wird!"

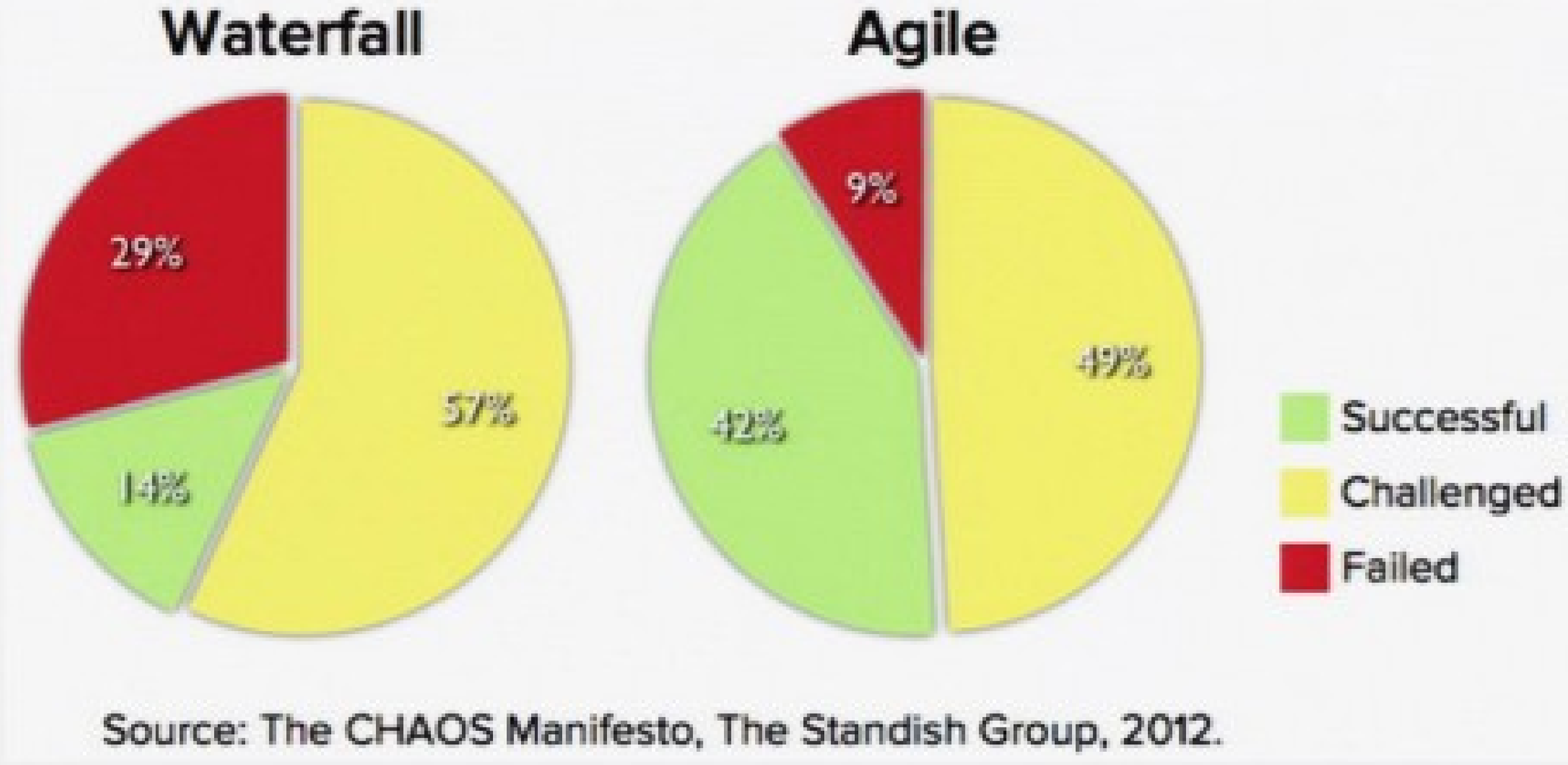
Weitere Befürchtungen:

- Mir fehlt als Auftraggeber wegen Pull-Prinzip, Selbstorganisation und flexiblem Scope die Kontrolle!

- Ich habe keine Möglichkeit, Druck auf das Team auszuüben.

- Ich bin unsicher, ob mit agilen Methoden das Projektziel sicher erreicht werden kann!

# CHAOS Manifesto (Standish Group)




Source: The CHAOS Manifesto, The Standish Group, 2012.

Untersuchung von > 80.000 beendeter Projekte

Quelle:
CHAOS Manifesto 2012, The Standish Group

# Das FBI "Sentinel" Projekt

**Ziel:**
**Ersetzen eines papierbasierten Prozesses zur Verwaltung von Fällen durch eine elektronische Lösung**

2001-2005: Projekt "VCF" entwickelt durch einen externen Lieferanten, scheitert.
Kosten: **170 Mio. $**

2006: Das "Sentinel" Projekt geht an Lockheed Martin, Plan: 6 Jahre, 4 Phasen, geplant **450 Mio $**

2010: 1.Phase abgeschlossen, ein Teil der 2.Phase erledigt, Kosten: **421 Mio $**
Schätzung: weitere 6 Jahre und zusätzlich 351 Mio $ bis zur Fertigstellung

**Kosten:**
**591 Mio $**

Das FBI beendet den Vertrag und holt das Projekt zurück.

Scrum Studio im Hoover Building Untergeschoss, Entwicklerzahl von 400 auf 40 reduziert.

Projekt abgeschlossen nach **1 Jahr** für **30 Mio $**. Der Rollout erfolgte am 1.7.2012

Quellen:


- http://blogs.collab.net/agile/case-study-of-a-difficult-scrum-project-fbi-sentinel#.Uvic2nvl5d0%d
- Kim Schwaber, XP Days 2011 (http://www.scrumpub.net/wiki/pub/XPDays2011/JustWhenYouThoughtYouKnewScrum/XP_Day.pdf)
- http://www.justice.gov (Inspector General reports)
- Wallstreet Journal http://online.wsj.com/news/articles/SB10000872396390444813304577561361561332?mg=reno64-wsj&url=http%3A%2F%2Fonline.wsj.com%2Farticle%2FSB10000872396390444813304577561361561332.html

# Umfrageergebnisse von Scott Ambler (642 Befragte)

Eine Entscheidung für eine agile Vorgehensweise ist ein sehr geringes Risiko. Von den Befragten berichten...

5% von niedrigerer Produktivität

9% von niedrigerer Qualität

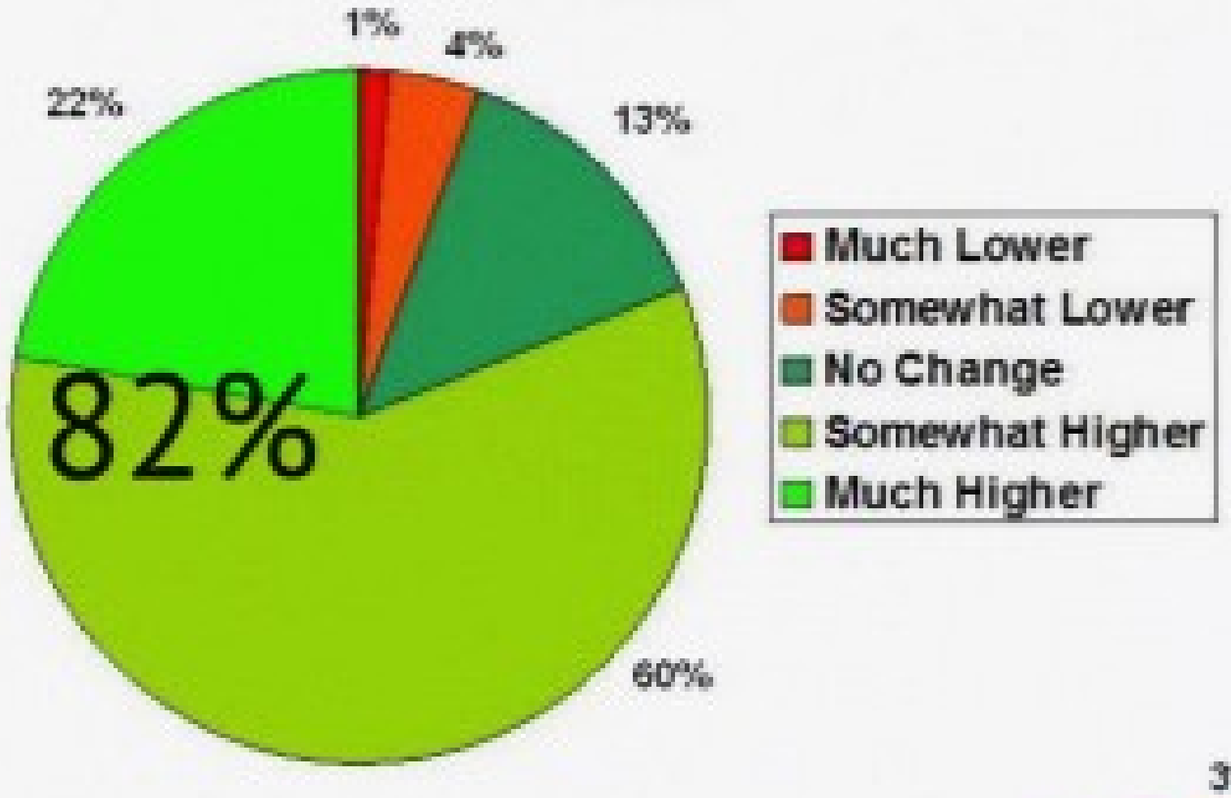
7% von niedrigerer Stakeholder-Zufriedenheit

Betrachtet man die "Somewhat higher" und "Much higher" Antworten, liegt eine Empfehlung auf der Hand...

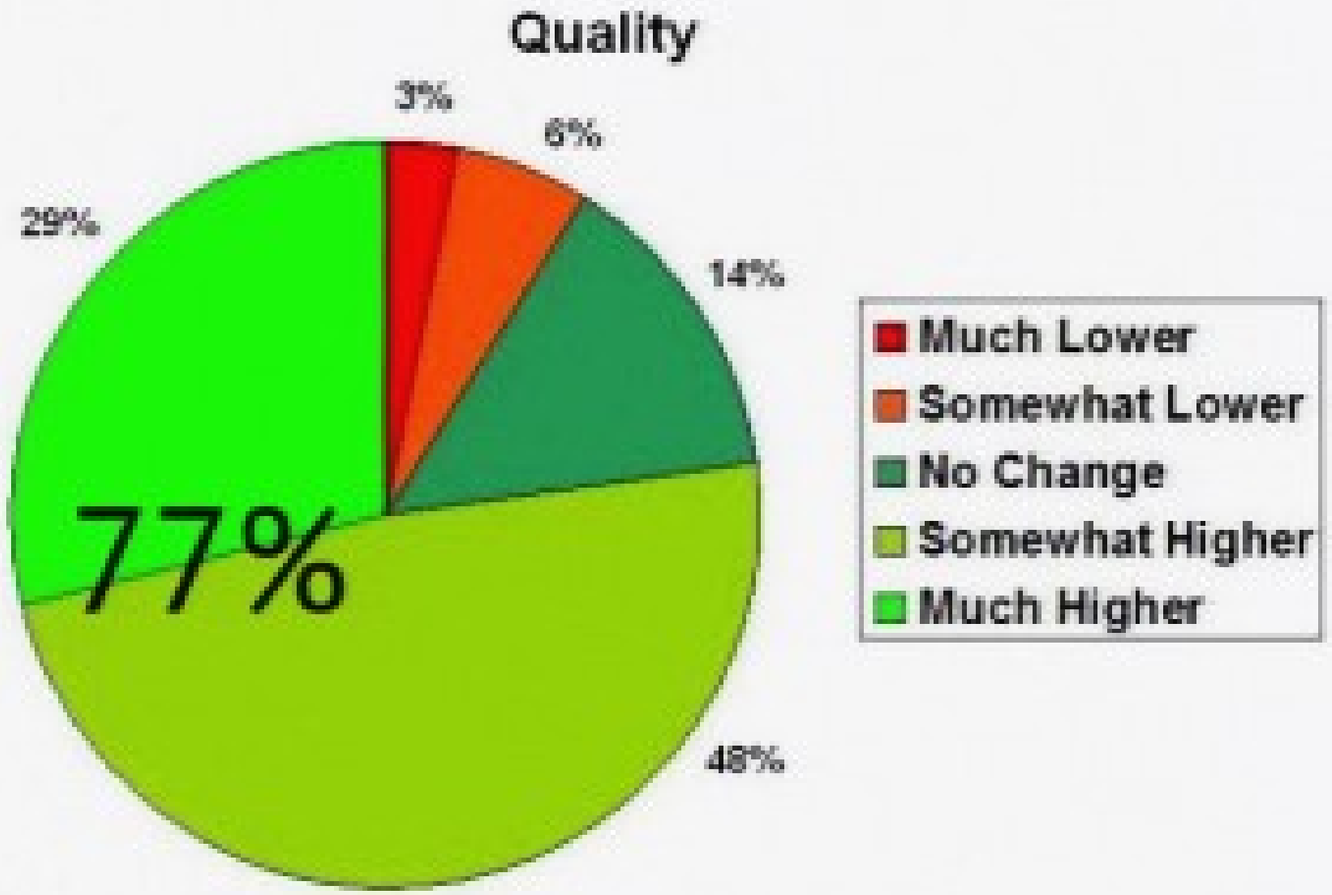
Quelle:
Scott Ambler / Ambysoft, http://www.ambysoft.com/surveys/agileFebruary2008.html

This survey was performed in early February 2008 and there was 642 respondents. The survey was announced in the blog of Ian Eriksson, the Dr. Dobb's Journal editor.

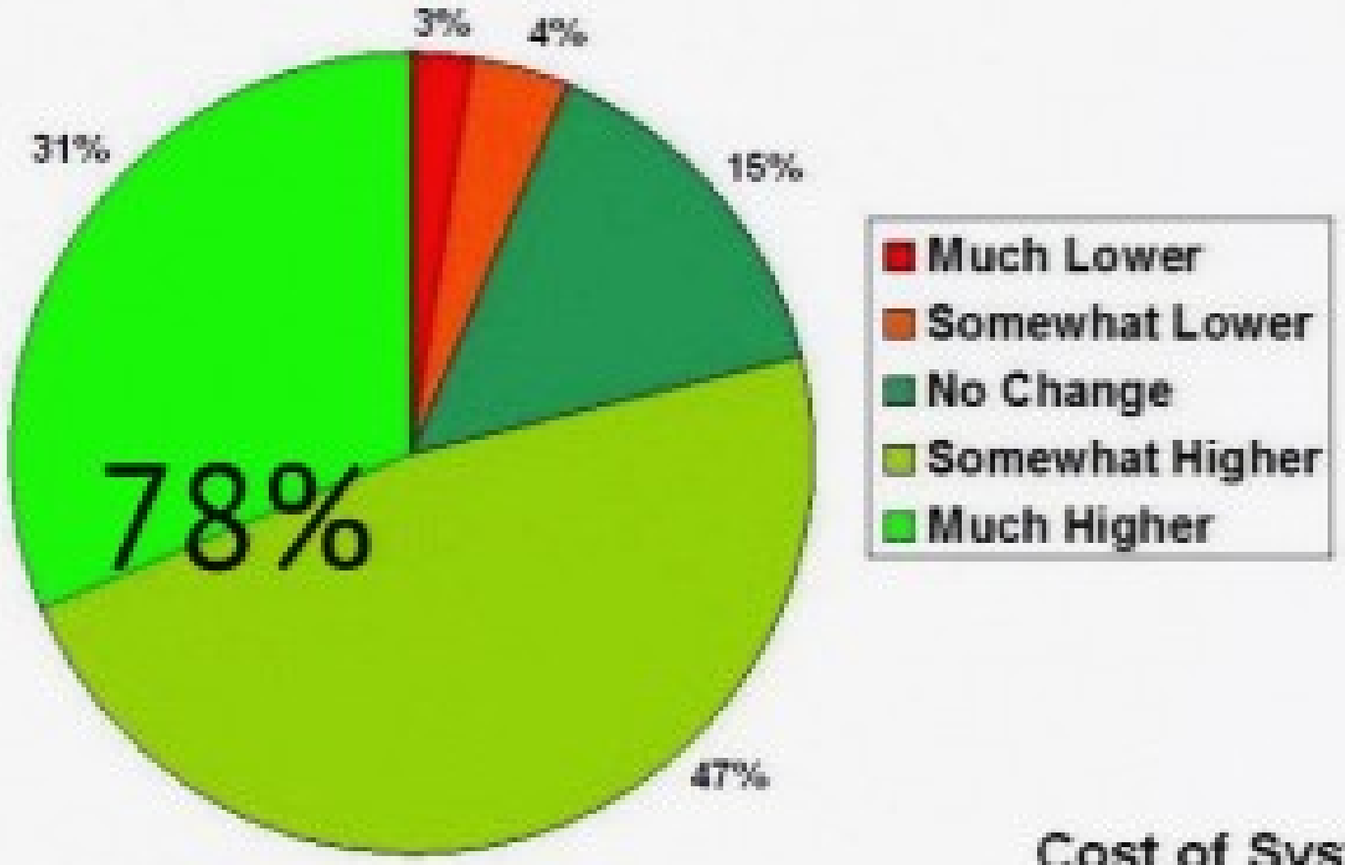
## Productivity



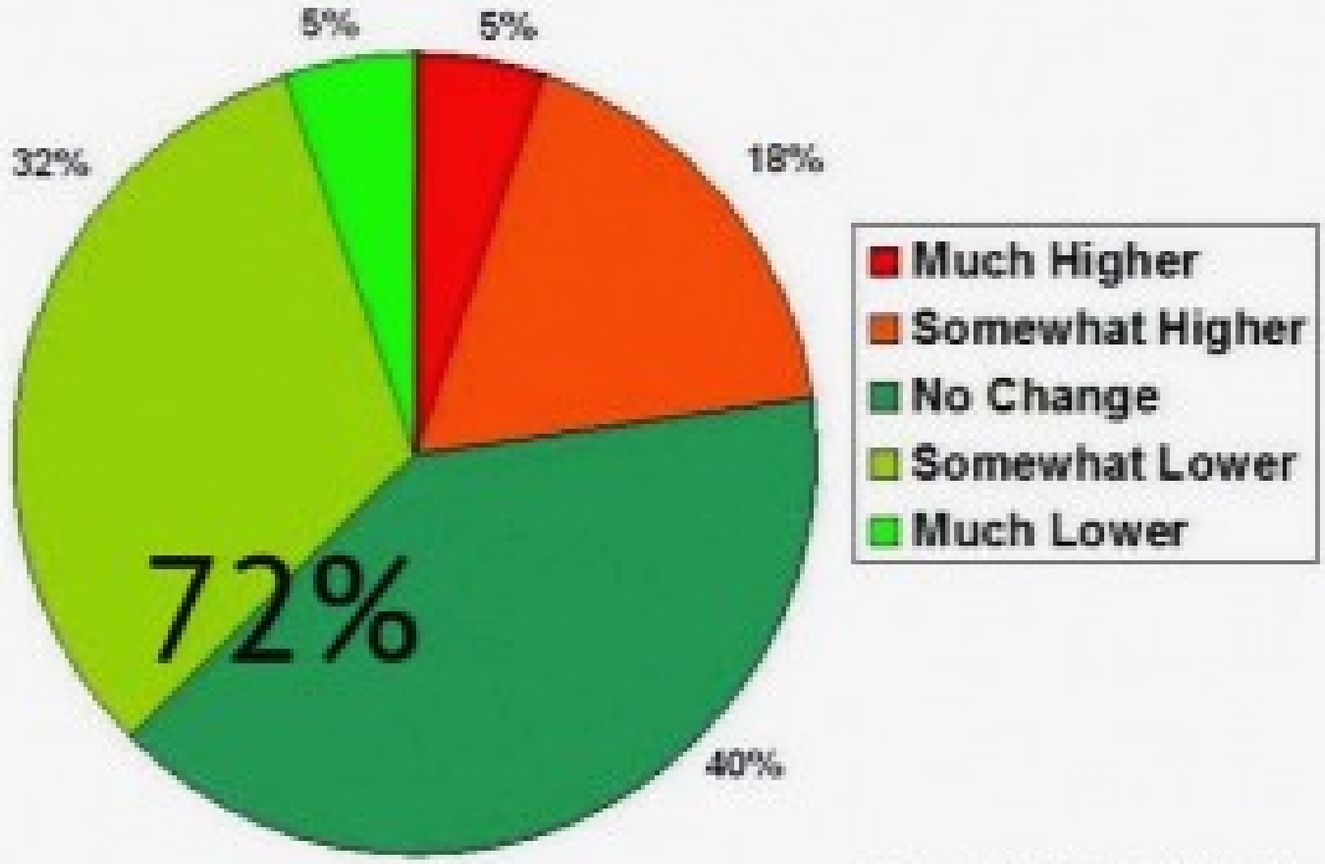

## Quality



## Business Stakeholder Satisfaction




## Cost of System Development




Source: Dr. Dobb's Journal 2008 Agile Adoption Survey

# Empirische Beobachtungen...

- Mit agilen Methoden lassen sich auch umfangreiche Aufgabenstellungen zielgerichtet und effizient erfolgreich lösen.

- Die Erfolgswahrscheinlichkeit und Kundenzufriedenheit ist höher als bei klassischen Methoden.

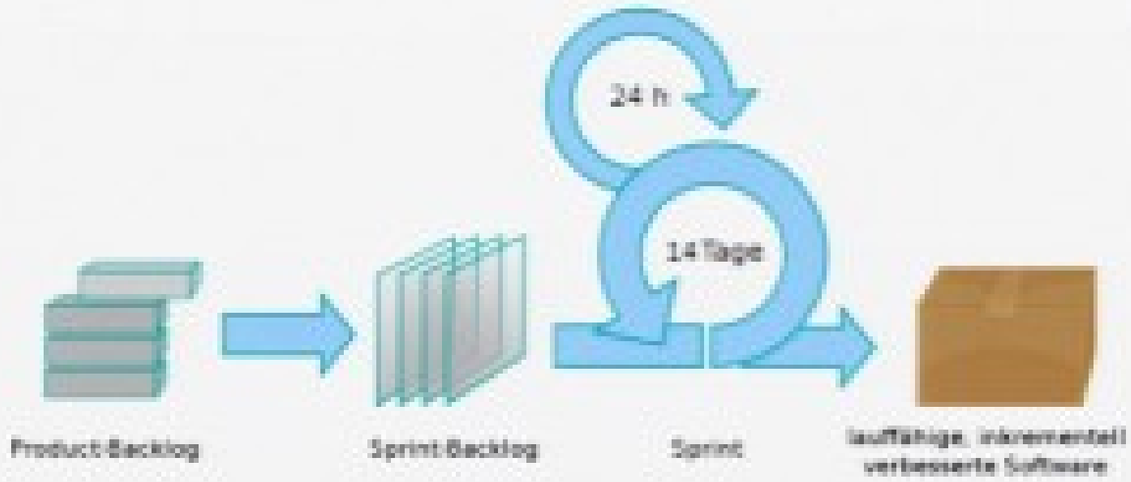
- Die Kosten sind niedriger.

- Die Qualität der Produkte ist höher.

"Der Aufwand für Testautomatisierung ist mir zu hoch!
Wir brauchen mehr Features!"

# Qualitative Betrachtung von Testautomatisierung


# Scrum Qualitätsanforderung




- Scrum fordert ein produktionsreifes Produkt zum Sprintende!
- => Ohne konsequente Testautomatisierung ist Scrum nicht anwendbar!
- Damit fallen mit fehlender Testautomatisierung wesentliche Vorteile von Scrum weg!

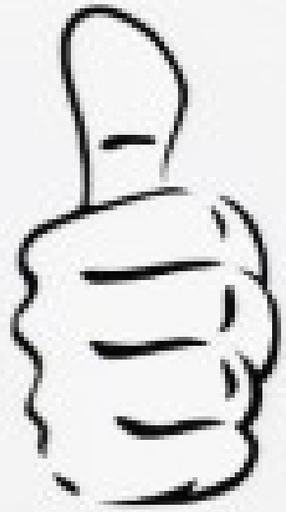
## Aufwand für Fehlersuche und -behebung

Pro Phase, in der ein Fehler später entdeckt wird,
steigt der Aufwand zur Behebung um eine Größenordnung!

# x 10

Quelle: Humphrey, W.S., Introduction to the Personal Software Process. 1997: Addison-Wesley

# Effekte niedrigerer Fehlerraten



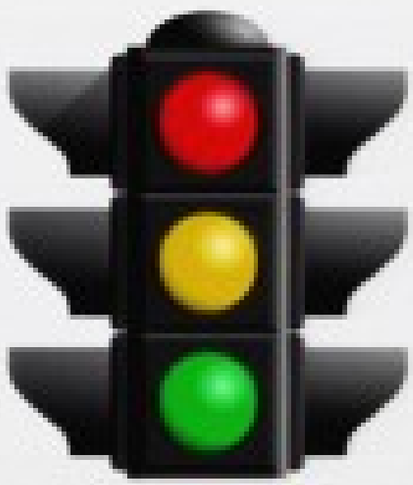
Stark reduzierte Fehlerraten führen zu...

- Weniger nicht-wertschöpfender Nacharbeit
- Mehr Zeit für wertschöpfende Arbeiten (z.B. Implementierung neuer Features)

# Mehr Vertrauen in den Code

- Eine hohe Testabdeckung bildet ein Sicherheitsnetz für Änderungen am Code
- Späte Änderungen werden dadurch erst möglich
- Diese Sicherheit reduziert Streß
- Das Motto "Never touch a running system!" wird überflüssig

## Einarbeitung neuer Teammitglieder



Durch die Absicherung wird die Einarbeitung neuer Mitarbeiter wesentlich einfacher, schneller und sicherer.

Ein neuer Mitarbeiter, der Code verändert, bekommt sofort automatisch Feedback, ob er bestehenden Code beschädigt hat oder nicht.

# Quantitative Betrachtung von Testautomatisierung

Quantitativ sehr schwierig zu berechnen, da kurzfristige und langfristige Effekte eine Rolle spielen.

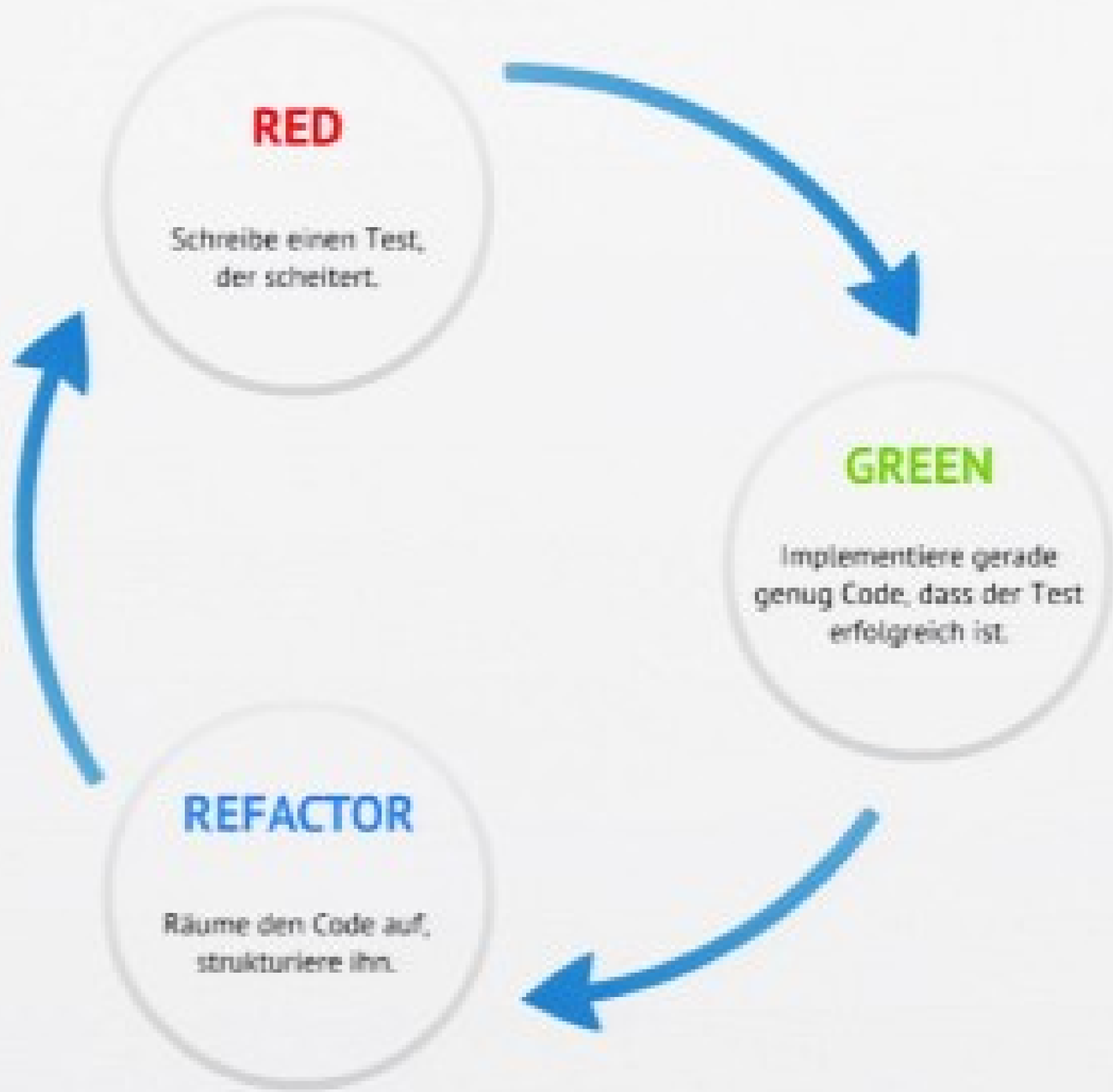
- Aufwandsreduktion durch Reduktion von Fehlern insgesamt bzw. durch früheres Entdecken von Fehlern und damit Beseitigung mit niedrigerem Aufwand
- Geringere Einarbeitungsaufwände für neue Mitarbeiter
- Geringeres Risiko von Fehlern in Produktion
  - Vermeidung unzufriedener Kunden
  - Vermeidung von Reputationsverlust

- Aufwandserhöhung zur Erstellung der automatisierten Testfälle
- Kosten für Testumgebungen und -tools

"Test-Driven Development ist Verschwendung!

Wir brauchen mehr Features, hab' ich doch schon gesagt!"

# "Test-Driven Development" in a nutshell



# Vorteile von Test-Driven Development

- Reduktion von Fehlern
  - Features, die nicht funktionieren, erzeugen keinen Business Value!
  - Durch TDD erreicht man von Anfang an hohe Qualität und erzeugt damit kontinuierlich Wert.

- Fokussierung
  - Hilfe für Entwickler, sich auf eine Aufgabe zu konzentrieren und mentale Entlastung von der Gesamtkomplexität des Systems.
  - Vermeidung von "Over-Engineering".

- Design und Architektur klarer

- Risikoreduktion bei parallelem Arbeiten
  - Erst wenn alle Tests lokal beim Entwickler erfolgreich laufen, wird der Code in die Versionsverwaltung eingecheckt.
  - Viel geringeres Risiko, die Arbeit anderer  zu beschädigen.

- Schnellere Time-to-market für neue Features

# Die Kosten bei Einführung von TDD

- Mehr Testcode wird erzeugt
  - Verhältnis Testcode : Produktionscode oft bis zu 2 : 1
- Anpassung der Testfälle ist notwendig
- Umstellung der Arbeitsweise
  - Ablegen althergebrachter Arbeitsweisen!
  - Zuerst wird ein Team in der Liefergeschwindigkeit für neue Features langsamer, einerseits durch den Lernaufwand, andererseits durch das Abzahlen technischer Schulden, die entdeckt werden.
  - Mit wachsender Erfahrung und Verbesserung des Codes wird das Team aber schneller als zuvor.

# Fallbeispiel Microsoft und IBM

Untersuchung von 3 Teams bei Microsoft, 1 Team bei IBM, die TDD eingeführt haben.

- Entwicklungsaufwand gestiegen um **15-35%** (Managementschätzung, ohne Berücksichtigung der Einsparungen in Testphasen und in Produktion/Support)
- Pre-release Defects gesunken um **40-90%**

Quelle
"Realizing quality improvement through test driven development: results and experiences of four industrial teams",
Nachiappan Nagappan & E. Michael Maximilien & Thirumalesh Bhat & Laurie Williams

# Fallbeispiel von Mike Cohn

- Unternehmen mit Qualitätsproblemen
  - Niedrige Kundenzufriedenheit
  - Reputation war beschädigt
  - Features konnten nicht mehr zeitnah implementiert werden

- **10 Fehler pro 1.000 NCSS** (non-comment source statements) von Kunden in den 6 Monaten nach einem Release gemeldet

- Maßnahme: Es wurde Scrum und TDD in der Firma eingeführt

- Die Fehlerrate sank auf **3 Fehler pro 1.000 NCSS**
  - Anteil der Altfehler unklar
  - Die Fehlerrate durch neuen Code war wahrscheinlich niedriger

- Konservativ geschätzt wurde eine Verbesserung von ca. **80-90%** erreicht

- Die Produktivität gemessen in Function Points pro Personenmonat stieg von **7 auf 23 (x 3,3)**

Quelle
'Implementing Lean Software Development From Concept to Cash', Mary Poppendieck

# Ist TDD eine gute Investition?

**IMHO: Ja.**

- Fokus auf Qualität führt zu hoher Produktivität
- "Denken in Tests" führt zu klaren Anforderungen
- Saubere Architektur
- Kein Over-Engineering
- Top-Qualität
- Spaß an der Arbeit durch Fokus auf Featureentwicklung statt Fehlerbehebung
- Kundenlob statt Schelte :-)

"Zwei Mitarbeiter an einem Computer sind Verschwendung!

Einer tut ja gar nichts!"

# Studie der University of Michigan

## Betrachtete Faktoren:

Wirtschaftlichkeit

- IBM: Kosten pro vom Kunden gemeldeten Fehler **> 8.000 US$** [1]
- Zeit für Behebung der Fehler im Vergleich zur Extra-Zeit für Pair Programming je nach Organisation zwischen Faktor **15** und Faktor **60**
- Entwicklungskosten: statt Erhöhung um **+100%** eine Reduktion um **-15%**

Kontinuierlicher Review

- Das Beheben eines Fehlers wird pro zusätzlichem Prozessschritt um den **Faktor 10** teurer [2]
- Mit Pairing werden Fehler oft bereits sofort entdeckt und damit vermieden

Designqualität

- Teams generieren mehr Lösungsoptionen als Einzelpersonen
- Der Code wird kürzer
- Die Qualität von Architektur und Design steigt

Quelle:
[1] Humphrey, W.S., A Discipline for Software Engineering. SEI Series in Software Engineering, ed. P. Freeman, Musa, John. 1995: Addison Wesley Longman, Inc.
[2] Humphrey, W.S., Introduction to the Personal Software Process. 1997: Addison-Wesley
Artikel "The Costs and Benefits of Pair Programming", Alistair Cockburn, Laurie Williams, http://www.cs.pomona.edu/classes/cs121/supp/williams_progpair.pdf

# Studie der University of Michigan (2)


Personal- und Projektmanagement

- Risikoreduktion => Truckfaktor sinkt

Problemlösung

- Paare lösen Probleme schneller als Einzelpersonen

Lernen

- Beide Entwickler lernen permanent voneinander
- Unerfahrene Entwickler lernen schneller, wenn sie mit einem Experten zusammen entwickeln

Quelle:
Artikel "The Costs and Benefits of Pair Programming", Alistair Cockburn, Laurie Williams, http://www.cs.pomona.edu/classes/cs121/supp/williams_proppm.pdf

# Studie der University of Michigan (3)


Teambuilding und Kommunikation

- Menschen lernen, zusammenzuarbeiten
- Teamwork wird gefördert
- Menge und Frequenz der Kommunikation nehmen zu
- Insgesamt steigert dies die Teameffektivität

Zufriedenheit der Teammitglieder

- Anonym befragte Teammitglieder haben klar Pair Programming bevorzugt

Quelle:
Artikel "The Costs and Benefits of Pair Programming", Alistair Cockburn, Laurie Williams, http://www.cs.pomona.edu/classes/cs121/Supp/williams_pppgm.pdf

# Fallstudie "C3 Projekt, Chrysler"

Ziel: Implementierung eines Gehaltsabrechnungssystems
Größe: 2.000 Klassen, 30.000 Methoden)

- Ausschliesslich Pair Programming

- In den letzten 5 Monaten vor Livegang waren die einzigen Fehler, die Unit- und Systemtest unerkannt passiert haben, von Entwicklern verursacht, die allein programmiert haben.

- Statistische Untersuchungen zeigen, dass Fehlerbehebung nach einem Release an Kunden um **> 100x** mehr kosten kann, als sie bereits während der Entwicklung zu finden und zu beheben.

- Bei gleicher geforderter Qualität sind Entwickler, die in Paaren arbeiten **40-50% schneller** als einzeln arbeitende Entwickler.

Quelle:
Artikel: "Strengthening the Case for Pair-Programming", Laurie Williams, Robert Kessler, Ward Cunningham, Ron Jeffries
http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-SWT2-2012/doc/WilKesCun00.pdf

# Fazit zu Pair Programming...

"Two programmers in tandem is not redundancy -
it's a direct route to greater efficiency and better quality."

Larry Constantine

Professor in the Center for Exact Sciences and Engineering
Ehem. Professor am MIT, University of Madeira Portugal

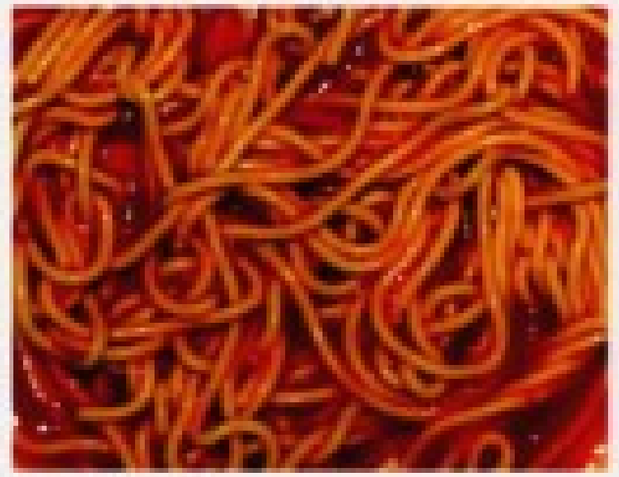
"Refactoring ist reine Zeitverschwendung!

Wie oft soll ich Ihnen noch sagen,
dass wir mehr Features brauchen!?"

Funktionale Qualität

vs.

Strukturelle Qualität



# Formen von technischen Schulden

- Spaghetti-Code
- Exzessive Komplexität
- Codeduplizierung
- Unsauberer Code jeglicher Form
- ...

# Einfluß technischer Schulden auf die Entwicklungsgeschwindigkeit

- Technische Schulden wachsen mit der Größe des Projekts, oft unmerklich.
- Kumulativer Effekt!
- Durch Eingehen von technischen Schulden sinkt die Entwicklungsgeschwindigkeit - sukzessive bis auf Null.
- Teams gehen oft technische Schulden ein, wenn der Druck, möglichst viele Features zu implementieren, hoch ist ("PUSH").
- Technische Schulden müssen zurückbezahlt werden, sonst droht Insolvenz!

## Refactoring

- Kontinuierliche Änderung der Struktur des Codes ohne eine Änderung des Verhaltens.

- Refactoring = Rückzahlung technischer Schulden

- Konsequentes, kontinuierliches Refactoring führt im Projektverlauf sogar zu einer Beschleunigung!

# Studien zu Refactoring?

Schwierig zu finden...

Aber:

- Refactoring ist elementarer Bestandteil z.B. von TDD
- Effekt von fehlendem Refactoring: "Rotten Code"
- Führt oft dazu, dass die Neuentwicklung eines Systems günstiger und sicherer ist als die Weiterentwicklung.
- Beispiel: Neuschreiben von 8 "monolithischen" GIS Web-Applikationen war günstiger als Implementierung von 2 großen Features
  - Neue Systeme haben jetzt > 85% gemeinsamen Code
  - Produktionsbetrieb über > 1 Jahr fehlerfrei
  - "Late Changes" angstfrei möglich

# Studie zu ThoughtWorks "Agile Development Approach"

Ein Kundenprojekt wurde von Forrester Consulting untersucht.

Elemente des "Thoughtworks Agile Development Approach":

- Short cycles
- Test first development
- Continuous integration
- Refactoring
- Empowered team
- Reduction of project risk

Quelle
http://www.thoughtworks.com/sites/www.thoughtworks.com/files/files/TEI-media.pdf
Forrester Consulting

# Studie zu ThoughtWorks "Agile Development Approach" (2)

1. Die Kosten für das untersuchte Kundenprojekt lagen **40% niedriger** als bei einem vergleichbaren Wasserfall-Projekt.

2. Höhere Qualität und effizientere Entwicklung führten zu einer Reduktion von Projektdauer, Fehlerzahl und Nacharbeit.

3. Repriorisierung und signifikante Änderungen am Leistungsumfang waren möglich.

4. Schlüsselanforderungen wurden früher geliefert und waren damit früher nutzbar.

## Meine bescheidene Meinung...

*Nicht der Einsatz, sondern der Verzicht auf agile Methoden und Entwicklungspraktiken ist meiner Meinung nach mit hoher Wahrscheinlichkeit ein kostspieliger Fehler!*

Use the force!

Für Feedback und Fragen:

E-Mail: eike.reinel@tngtech.com
Twitter: @eike_reinel