

Frankfurt, 21. September 2011

Wertorientierung – nicht nur für agile Projekte

Jonathan Streit

itestra GmbH = informationstechnologie und –strategie

- itestra ist ein innovativer, unabhängiger Software-Dienstleister im Bereich unternehmenskritischer Prozesse, Systeme und Anwendungen
- 20+ Jahre ausgewiesene wissenschaftliche Arbeit in relevanten Forschungsgebieten:
 - Verteile Systeme, High Performance Computing
 - Software (Re-)Engineering
 - 80+ internationale Publikationen
 - Quamoco Initiator und Projektpartner
- Hochqualifizierte Mitarbeiter mit relevantem akademischen Werdegang und internationaler Ausrichtung
- Leistungen:
 - Software Analysen, Software Kennzahlen
 - Reengineering, Tuning, Modernisierung
 - Productivity Improvement
 - Solution Engineering



Projekt Quamoco



Ziel: Ein praktisch anwendbarer Software Qualitäts-Standard

- operationalisiert
- (ökonomisch) begründet

Quamoco-Forschungsprojekt:

- Drei Jahre Laufzeit (2009 - 2011)
- Unterstützt durch Bundesministerium für Bildung und Forschung (BMBF)
- Investition:
 - 3.7 Mio € durch BMBF
 - 2.2 Mio € durch Industrie

Quamoco

Der Qualitätsstandard für Software



SIEMENS



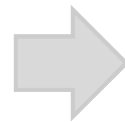
Kennzeichen: 01IS08023B



Investition



**SW-Entwicklung
und -Wartung**

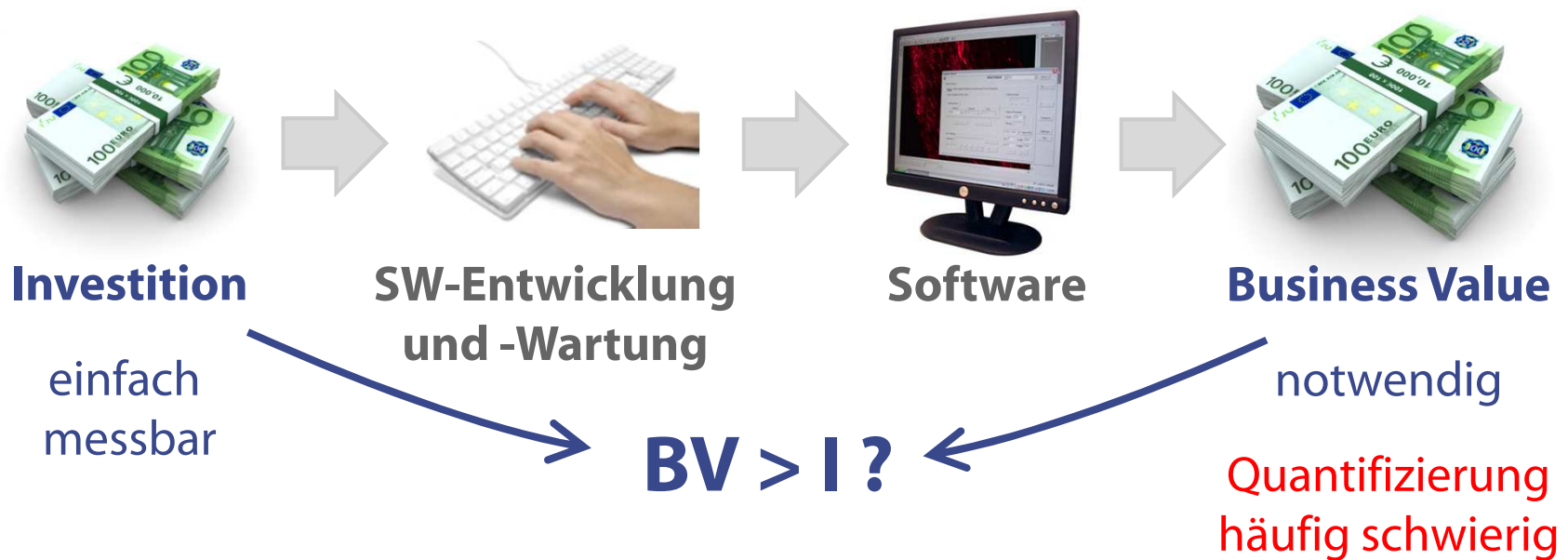


Software



Business Value

Wertorientierung





These:

In „klassischer“ Softwareentwicklung, -wartung und -controlling fehlt es an einem **Wertbegriff** (und in Folge auch Leistungsbegriff)



Investition



**SW-Entwicklung
und -Wartung**



Software



Business Value

Risiken:

1. „Sparschäden“ unerkannt / provoziert.
2. Ursächlichkeit bleibt unklar.
3. Grobgranulare, keine gezielte Steuerung einzelner Systeme.



Investition

Status Quo:
Fokus ausschließlich
auf **Kosten**,
ohne adäquate
Betrachtung d. **Leistung**

Übliche IT-Kennzahlen:

- IT-Kostenquote
- Kosten AE, Lizenzen, Hardware etc.
- Projekt: Plan/Ist-Vergleich
- Veränderung des Budgets ggü. Vorjahr
- Dienstleister: Stundensätze

Aussage zu Termin-/
Budgettreue,
jedoch nicht über
Wirtschaftlichkeit.

Keine Aussage über Performance

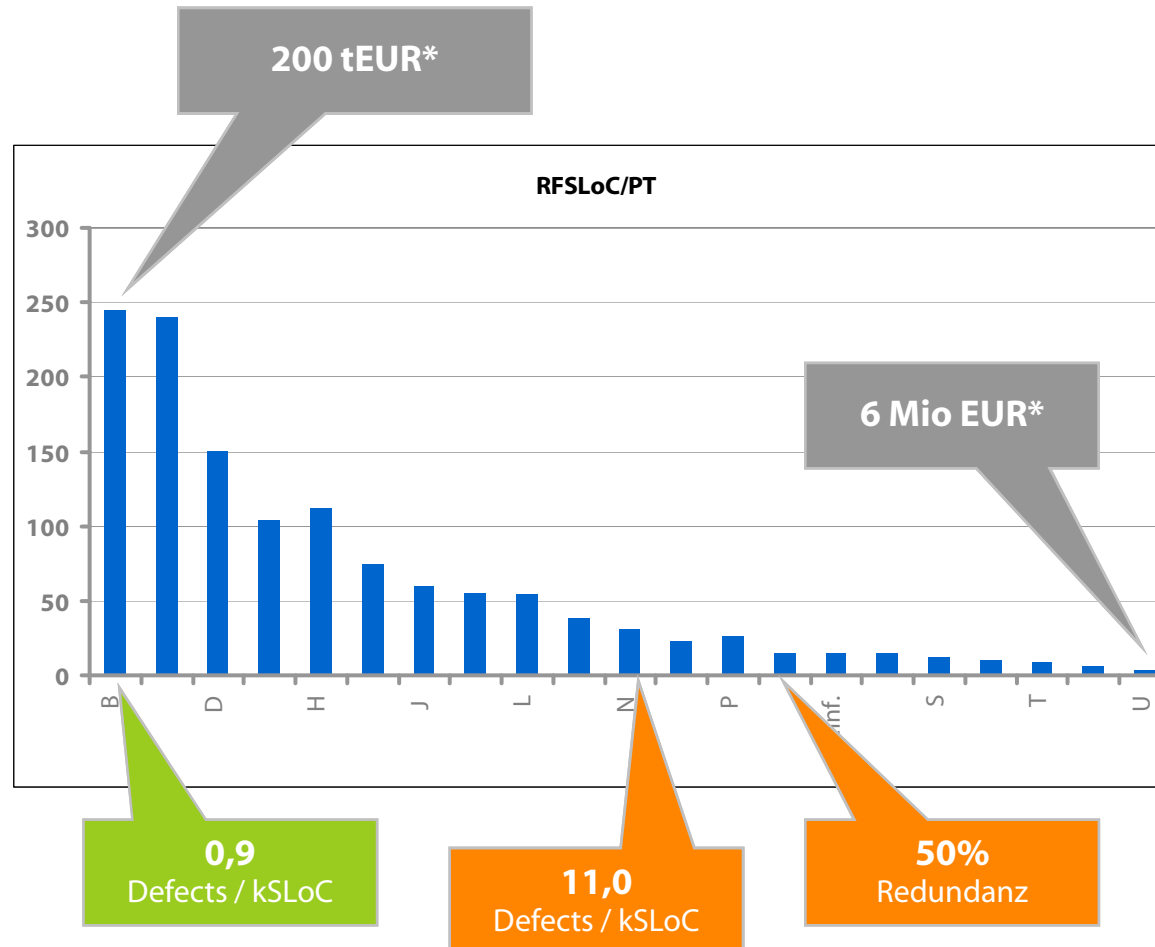
Vergleich: agile Software-Entwicklung

- Erstes Prinzip agiler Softwareentwicklung aus dem Agile Manifesto:
*Our highest priority is to satisfy the customer through early and continuous delivery of **valuable software**.*
- Fortschrittsmaß in XP: **Project velocity**
(Tatsächlich umgesetzte = einsatzbereite Features / Iteration)
- Lean Production:
 - Prinzip **Wertschöpfung** und **Wertstrom**
 - Alles andere = **Waste**



Erfahrungen (1)

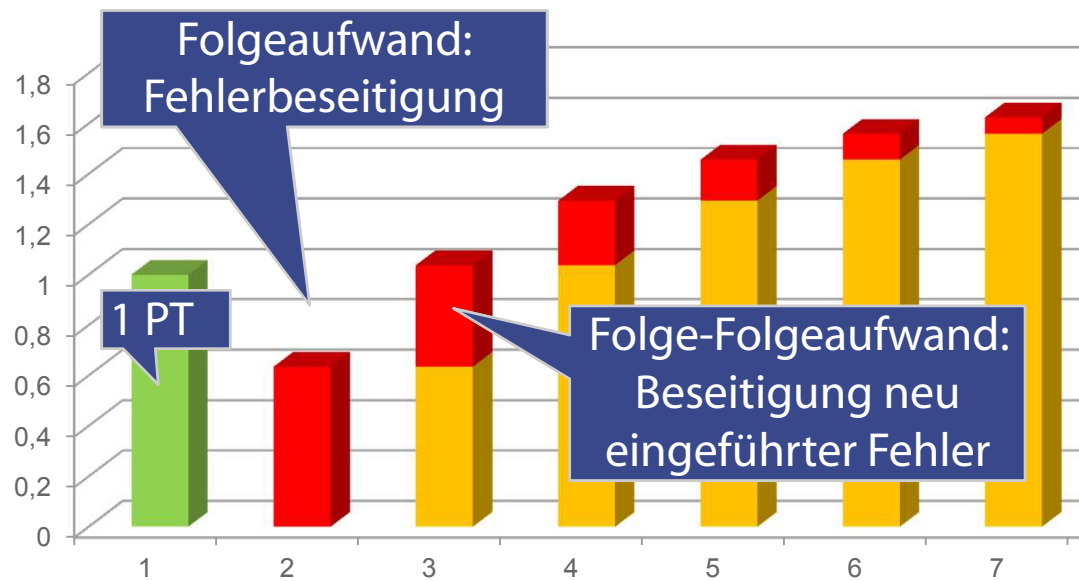
Enorme Unterschiede im Entwicklungs-Output:



*Beispielrechnung für System mit 100 kRFSLoC

Erfahrungen (2)

Fehlerbehebung = Waste!
1 PT investiert = 1 PT Leistung?



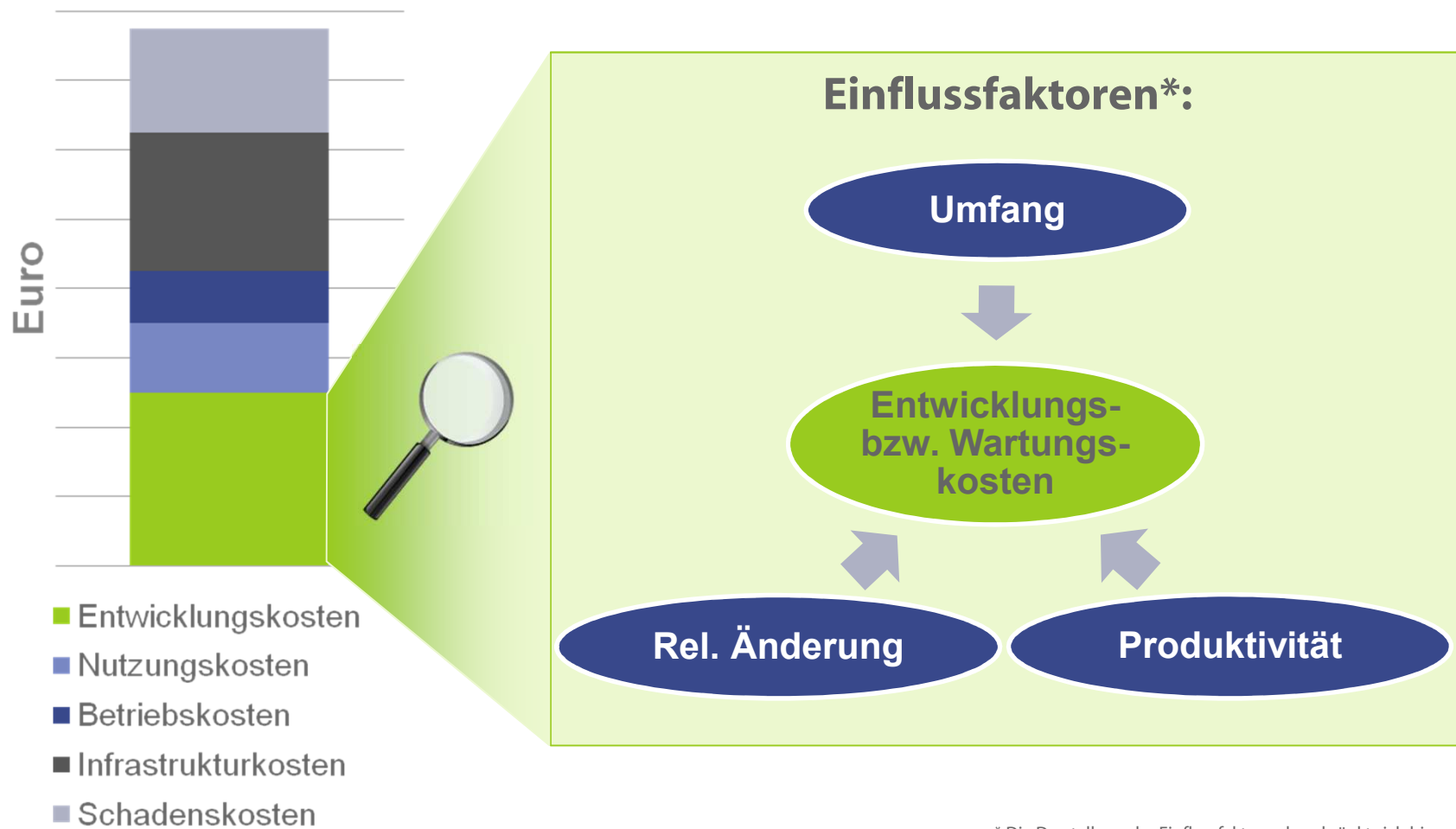
Erfahrungen (3)

OmsSecurePasswordModule.java	PasswordDeEncoder.java
<pre>149 150 public static String decode(String secret) throws NoSuchPaddingException, NoSuchAlgorithmException, 151 InvalidKeyException, BadPaddingException, IllegalBlockSizeException { 152 byte[] kbytes = "jaas is the way".getBytes(); 153 SecretKeySpec key = new SecretKeySpec(kbytes, BLOWFISH); 154 155 BigInteger n = new BigInteger(secret, SIXTEEN); 156 byte[] encoding = n.toByteArray(); 157 158 //SECURITY-344: fix leading zeros 159 if (encoding.length % EIGHT != 0) { 160 int length = encoding.length; 161 int newLength = ((length / EIGHT) + 1) * EIGHT; 162 int pad = newLength - length; //number of leading zeros 163 byte[] old = encoding; 164 encoding = new byte[newLength]; 165 for (int i = old.length - 1; i >= 0; i--) { 166 encoding[i + pad] = old[i]; 167 } 168 } 169 170 Cipher cipher = Cipher.getInstance(BLOWFISH); 171 cipher.init(Cipher.DECRYPT_MODE, key);</pre>	<pre>68 } 69 70 public static String decode(String secret) throws NoSuchPaddingException, NoSuchAlc 71 InvalidKeyException, BadPaddingException, IllegalBlockSizeException { 72 byte[] kbytes = JAAS_IS_THE_WAY.getBytes(); 73 SecretKeySpec key = new SecretKeySpec(kbytes, BLOWFISH); 74 BigInteger n = new BigInteger(secret, SECRET_RADIX); 75 byte[] encoding = n.toByteArray(); 76 if (encoding.length % EIGHT != 0) { 77 int length = encoding.length; 78 int newLength = (length / EIGHT + 1) * EIGHT; 79 int pad = newLength - length; 80 byte[] old = encoding; 81 encoding = new byte[newLength]; 82 for (int i = old.length - 1; i >= 0; i--) { 83 encoding[i + pad] = old[i]; 84 } 85 } 86 87 Cipher cipher = Cipher.getInstance(BLOWFISH); 88 cipher.init(2, key); 89 byte[] decode = cipher.doFinal(encoding); 90 return new String(decode);</pre>

(Strukturell) identische Verarbeitung

- **5 - 90% Cloning** in produktiven Anwendungen bei Großunternehmen
 - Bis zu **30% statisch unerreichbarer Code**
 - Standish Group: **45%** der Features einer Software **nie verwendet**
- ⇒ **Erhöht die laufenden Kosten für Entwicklung und Wartung**
- ⇒ **Unused: Entwicklungskosten ohne Wertbeitrag**

Softwarekosten und Einflussfaktoren



* Die Darstellung der Einflussfaktoren beschränkt sich hier auf Entwicklungskosten; für andere Kostenpositionen (z.B. Infrastruktur) existieren analoge Modelle.

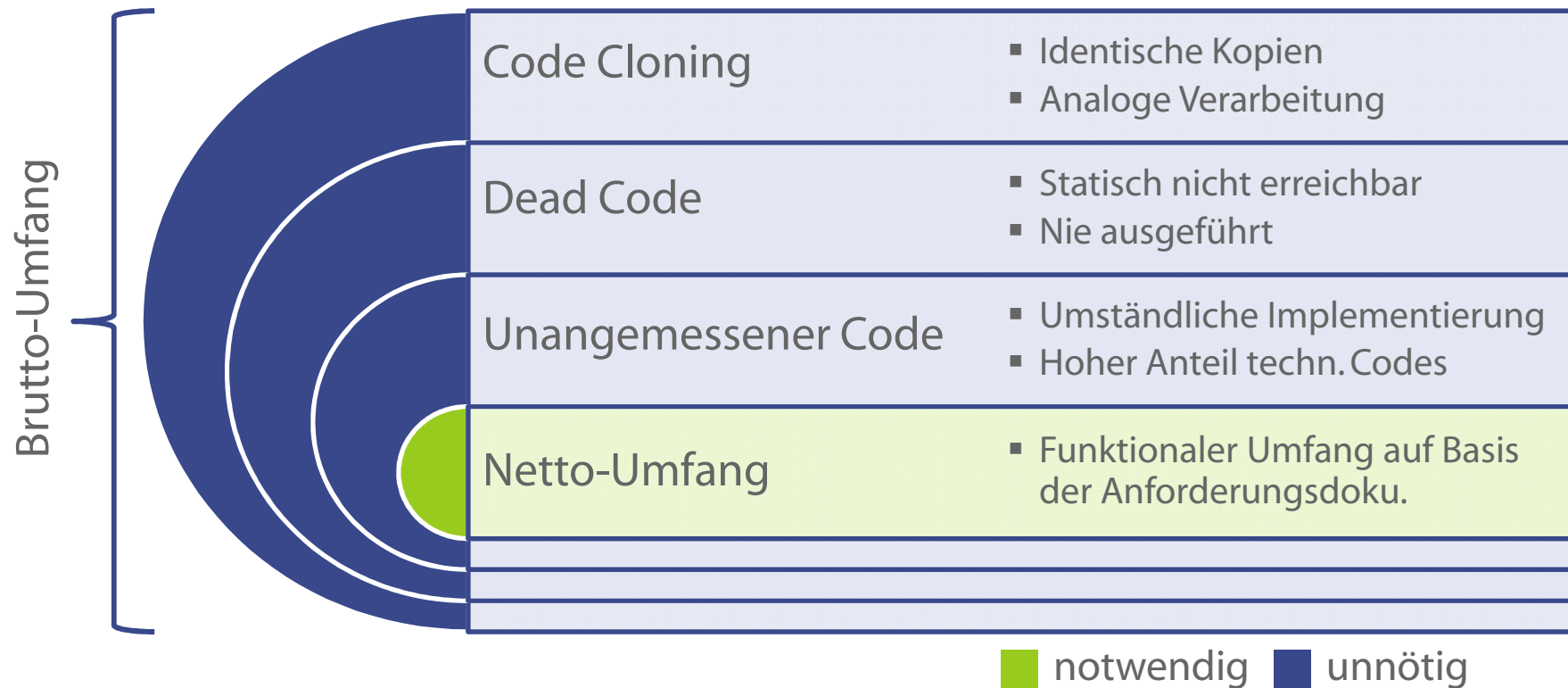
Einflussfaktor 1: Umfang (Messansätze)



Metrik	Aufwand	Aussagekraft
LOC	Niedrig	Gering
SLOC	Niedrig	Gering
Function Points (manuell)	Sehr hoch	Sehr hoch
Function Points (backfired)	Niedrig	Gering
Redundanzfreie SLOC (RFSLOC) bzw. FP auf Basis RFSLOC	Niedrig	Hoch

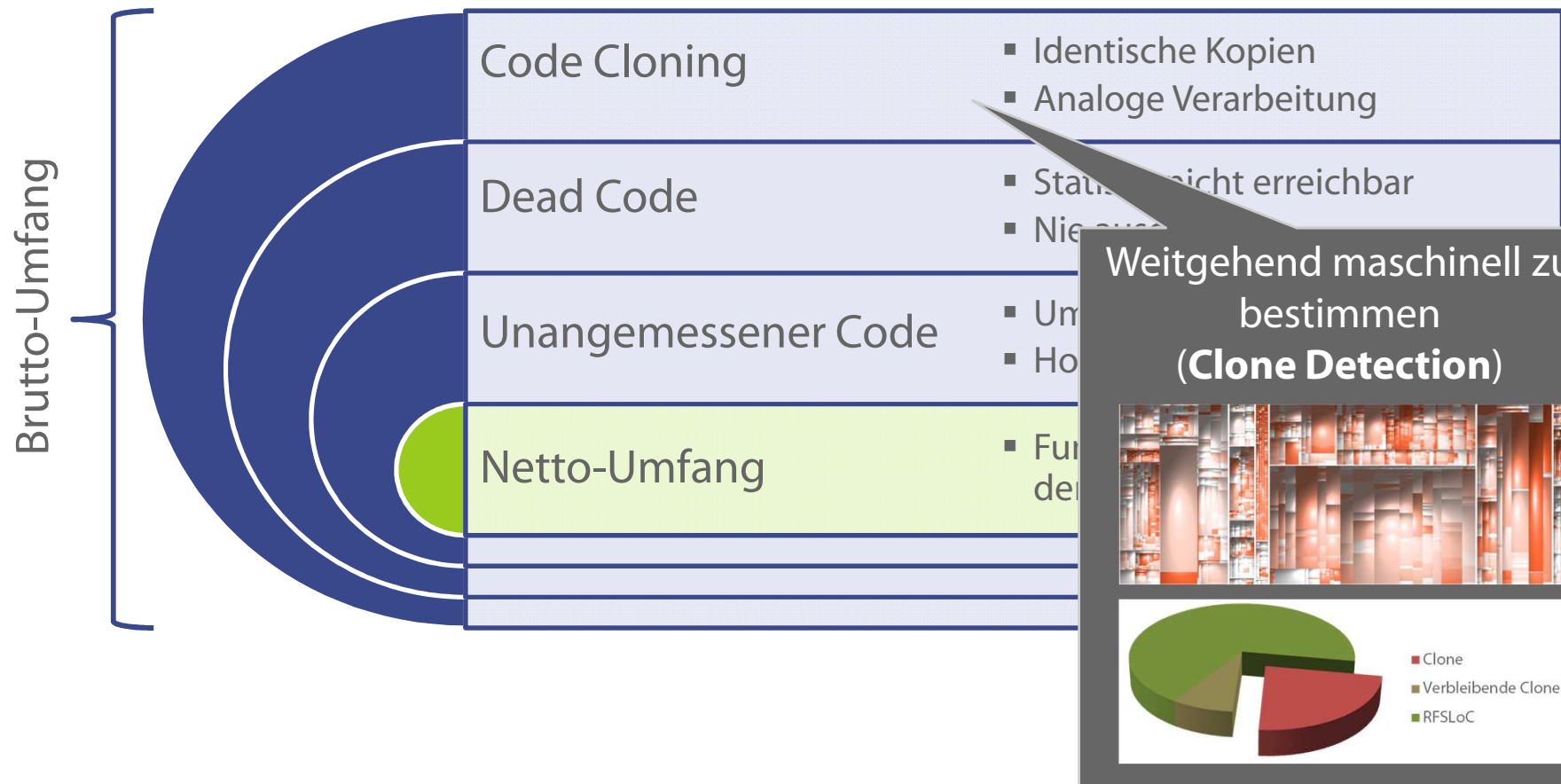
- Manuelle Function Point Messung liefert das beste Ergebnis, die Durchführung ist jedoch langwierig und teuer
- Durch die maschinelle RFSLOC-Messung kann eine gute Näherung mit minimalem Aufwand erzielt werden

Einflussfaktor 1: Umfang (Brutto-Netto)

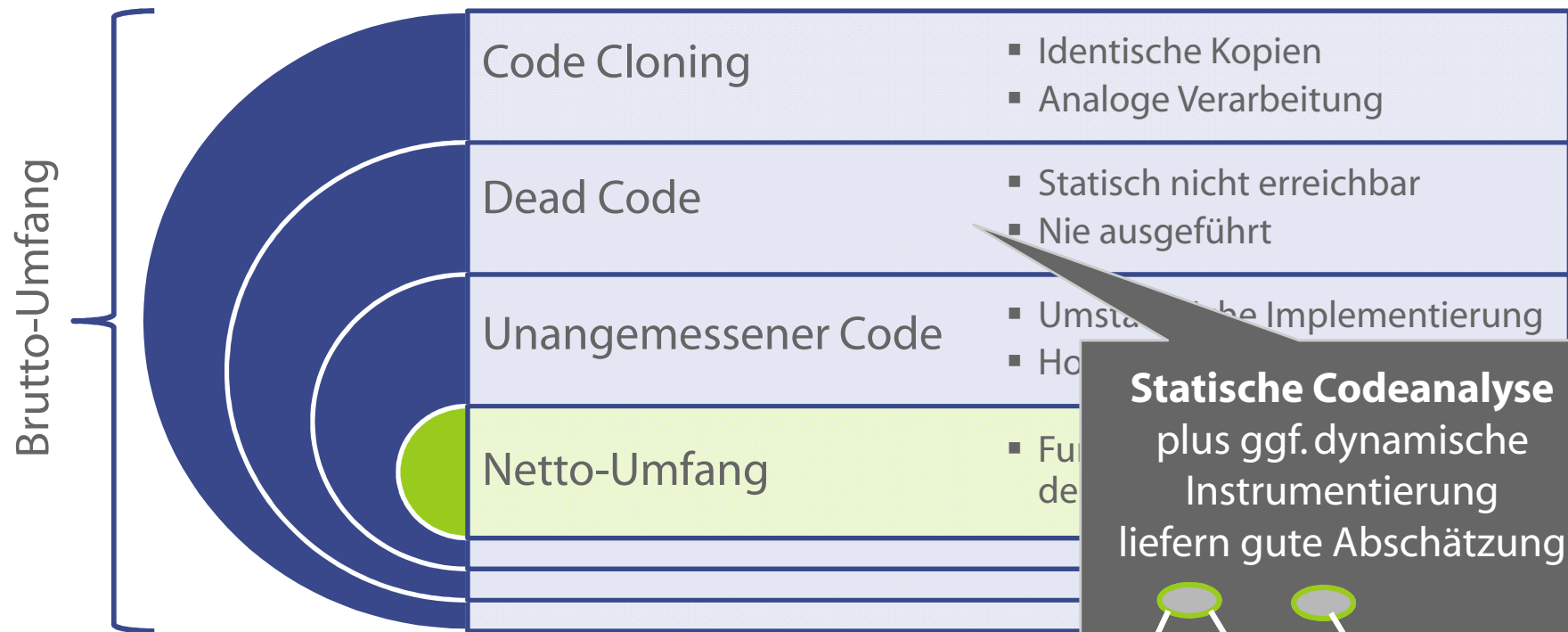


- **Angemessen:** Brutto = Netto

Einflussfaktor 1: Umfang (Brutto-Netto)



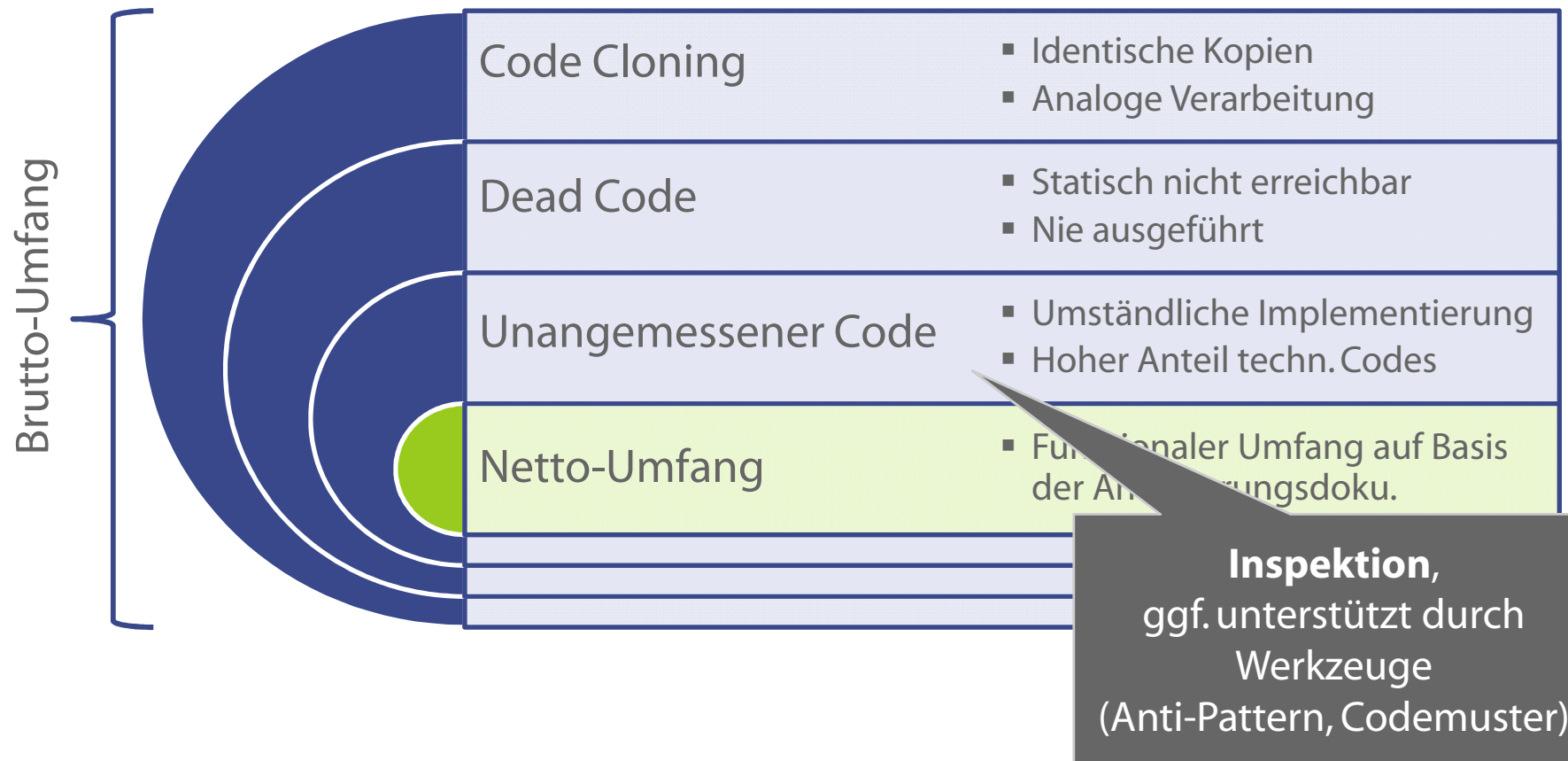
Einflussfaktor 1: Umfang (Brutto-Netto)



Statische Codeanalyse
plus ggf. dynamische
Instrumentierung
liefern gute Abschätzung

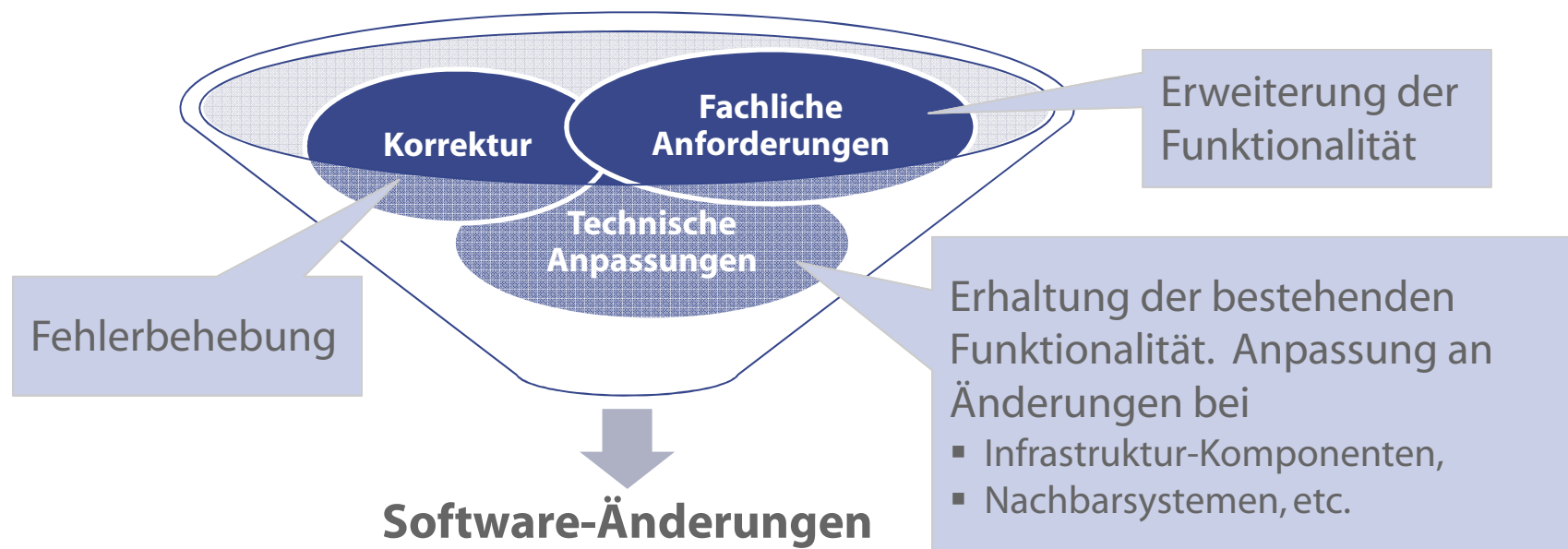


Einflussfaktor 1: Umfang (Brutto-Netto)



Einflussfaktor 2: Änderungsrate

- Software-Applikationen unterliegen stetigem Änderungsdruck



- Die Änderungsrate variiert zwischen Software-Applikationen
- **Angemessen:** Keine Korrekturen sondern primär Erweiterung

Einflussfaktor 3: Produktivität

- Der Aufwand pro durchzuführender Änderung ist von zahlreichen Faktoren abhängig:



- Produktivitätsmessungen zeigen: Die Produktivität variiert zwischen verschiedenen Teams enorm (Faktor 20)!
- Angemessen:** Minimale Störfaktoren, ~100 SLoC, >1 Function Point/Tag

Modell der Entwicklungskosten p.a.

Kosten	=	Umfang	×	Rel. Änderung	÷	Produktivität
Ist-Kosten		Brutto-Umfang		Korrekturen, Anpassungen und Erweiterungen		Neue und geänderte Umfänge pro Personentag
Soll (=Benchmark)		Netto-Umfang		Erweiterungen, Anpassungen		>1 Function Point oder 100 Zeilen pro Personentag

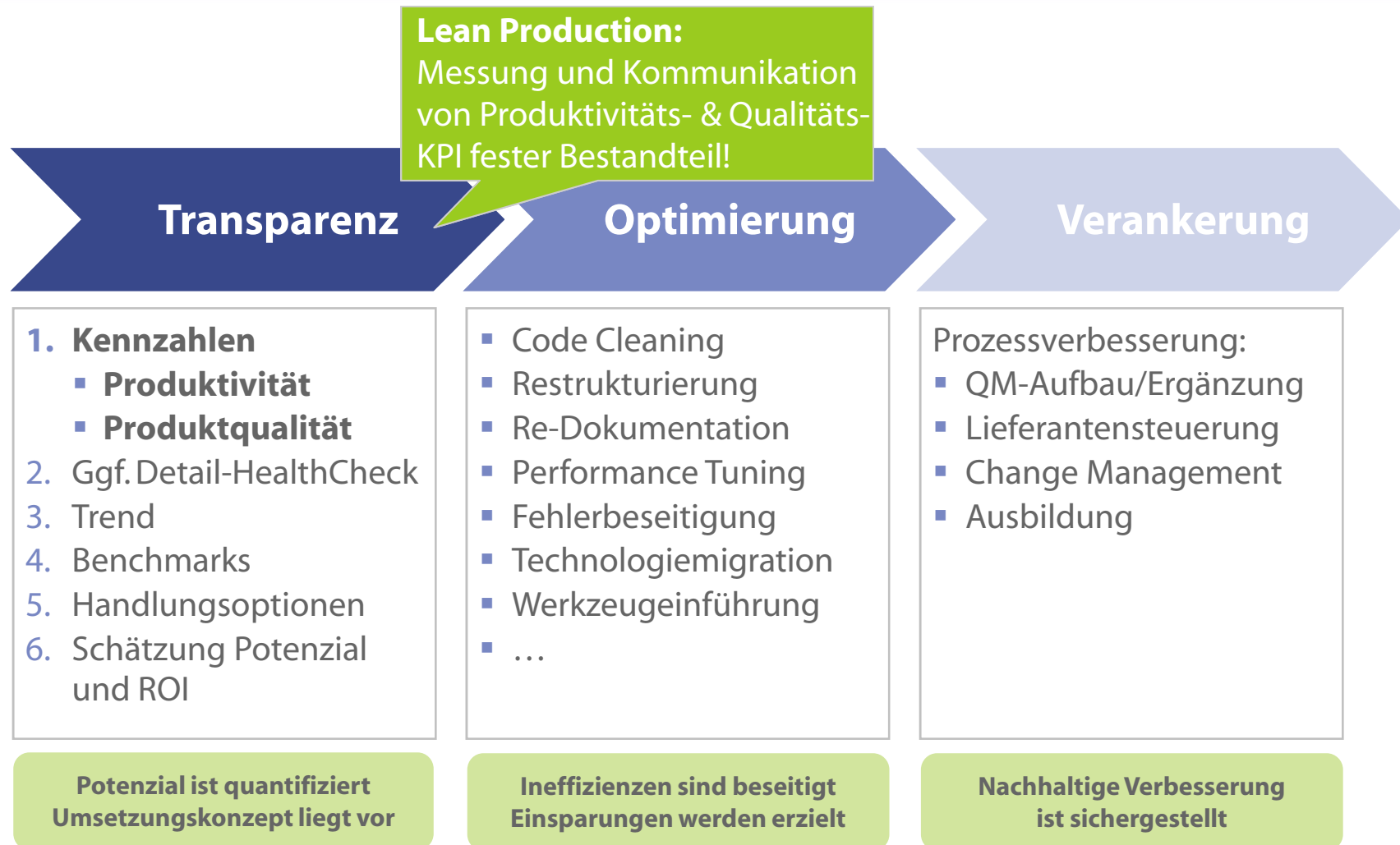
- **Differenz der Ist-/Soll-Kosten** entspricht dem **Optimierungspotential**
- Aktuelle Kosten **angemessen**, falls Ist/Soll Differenz minimal ist

Zur Bestimmung der Parameter des Kostenmodells genügen
7 Kennzahlen (KPI):

TCO & grobgranular!

KPI	Beschreibung	Parameter des Kostenmodells
Jährliche Kosten	Gesamtkosten p.a., aufgeteilt auf einzelne Positionen (Erweiterung, Wartung, Fehler, etc.)	Kosten
Umfang (brutto)	Gesamtumfang	Brutto-Umfang
Redundanzfreier Ant.	Gesamtumfang abzüglich Cloning und Dead Code	Netto-Umfang
Angemessenheit	Redundanzfreier Anteil abzüglich unnötig „aufgeblähten“ Codes	
Wachstum/Änderung	Absoluter Zuwachs & geänderte Programmzeilen	Rel. Änderung
Fehleranteil	Anteil Fehlerkorrektur an Gesamtaufwand	
Geschwindigkeit	Neue und geänderte Umfänge (FP, RFSLoC) pro PT	Produktivität

FP: Function Points; SLoC: Source Lines of Code – Anzahl Programmzeilen ohne Leerzeilen und Kommentarzeilen



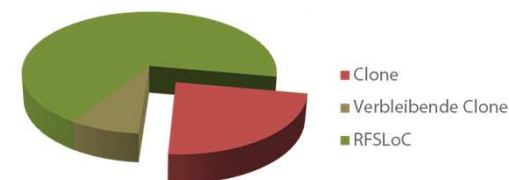
Symptome	Handlungsmöglichkeiten
Niedrige Geschwindigkeit (Änderung und Wachstum pro PT)	<ul style="list-style-type: none">▪ Optimierung Entwicklungsprozess (nicht nur Coding, z.B. auch Anforderungen) siehe Lean Software Development▪ Wechsel des Dienstleisters?
Hoher Fehleranteil, niedriges Wachstum (relativ p.a. und pro PT)	<ul style="list-style-type: none">▪ Beseitigung von Qualitätsdefiziten▪ Fehlervermeidung (Anforderungsprozess, Assertions, ...)
Niedriges Wachstum bei akzeptabler Geschwindigkeit	<ul style="list-style-type: none">▪ Keine Anforderungen? ⇒ Ende Lebenszyklus oder Nische?▪ Anforderungstau? ⇒ Budget nicht ausreichend
...	<ul style="list-style-type: none">▪ ...

Beispiel

	Kennzahl	Java-System	COBOL-System
Extern	Geschwindigkeit Wachstum (RFSLOC/PT)	35	6
	Geschwindigkeit Änderung (CRFSLOC/PT)	75	20
	Rel. Wachstum (% p.a.)	4%	2%
	Fehleranteil (PT _{Def} /PT)	30%	50%
Intern	Cloning	durchschnittlich	hoch (40%)
	Dead Code	n/a	10%

(Werte ähnlich realen Informationssystemen)

- Über 50 **strukturierte Analysen** geschäftskritischer Informationssysteme
 - Standortbestimmung Wachstum, Änderung, Produktivität, Fehlerraten
 - Technische und organisatorische Ursachen
 - In Folge: Umsetzung von Maßnahmen z.B.:
 - Entfernung Unused Code (bis zu 30%) und Clone-Entfernung
 - Inkrementelle Migrationsstrategie COBOL → Java
 - Erstellung Überblicksdokumentation, Wissensverteilung im Team
 - Verschlinkung Anforderungsprozess



■ Kennzahlenerhebung Software-Portfolio

- Umfeld: Große europäische Versicherungsgruppe (> 20 Mio LOC)
- Gegenüberstellung mit manuell erhobenen Zahlen (insb. Function Points)
- Ergebnis:
 - Hohe Korrelation mit manuell erhobenen Kennzahlen
 - Neue qualitative Ergebnisse, bislang nicht bekannt
 - Minimaler Aufwand ~ Bruchteil der manuellen Auszählung

	Kennzahl	Java-System	COBOL-System
Extern	Geschwindigkeit Wachstum	35	6
	Geschwindigkeit Änderung	75	20
	Rel. Wachstum (in p.p.)	4%	2%
	Fehleranteil (in p.p.)	30%	50%
Intern	Cloning	durchschnittlich	hoch (40%)
	Dead Code	n/a	10%



HABEN SIE FRAGEN?



Jonathan Streit | streit@itestra.de