

# Kontinuierliche Integration in verteilten Projekten mit Offshoring

---

Scrum-Day 2011 – Darmstadt

Version 0.9  
13-Sep-2011  
Gero Seifert

Experience certainty.

IT Services  
Business Solutions  
Outsourcing

# Vortragender: Gero Seifert – Tata Consultancy Services (TCS)

- Programmleiter bei TCS D'dorf seit 2008
- Seit 20 Jahren in der SW-Entwicklung tätig
- Über 14 Jahre Erfahrung mit indischem Offshoring
- 6 Jahre Management agiler Projekte
- Maßgeblich beteiligt bei der Einführung agiler Praktiken bei Nokia, NSN und TCS.
- Aktuelles agiles Projekt:  
VW Car Configuration Services
- Diplom-Informatiker,  
Universität Erlangen-Nürnberg



**Kontinuierliche Integration (CI) in Verteilten Projekten mit Offshoring**

# **Dieser Vortrag gibt Hinweise, wie Sie...**

- I. ...den Begriff „Kontinuierliche Integration“ (CI) verstehen können.
- II. ...überschauen, wie komplex die Einführung von CI in Ihren Projekten wäre.
- III. ...einschätzen, in wie weit Ihre Projekte von CI profitieren würden.

# Übersicht

12:35 – 13:15

- 1** Begriffsklärung
- 2 Vor- und Nachteile
- 3 Fünf Schritte zur Einführung
- 4 Zusammenfassung
- 5 Diskussion

# CI betrifft IT-Projekte...

...die SW-Komponenten und -Module *entwickeln* oder warten, die zusammen ein Produkt bilden sollen.

---

CI betrifft **nicht** IT-Projekte, die primär dazu dienen, fertige SW-Produkte bei einzelnen Kunden zu integrieren oder zu konfigurieren (z.B. SAP, Siebel).

# CI ist nicht nur Versionsverwaltung...

...die *SW-Entwickler* eines Teams verwenden, um ihren *Quellcode* bei Bedarf schnell, konsistent und einheitlich *abzugleichen*.

# CI ist eine Praktik...

...von *SW-Entwicklungsteams*, die *ihre Artefakte\** regelmäßig, häufig und vollständig untereinander *abgleichen* und ihr Produkt nach jedem *Commit\*\** *automatisch* bauen und *testen*.

\* Artefakte zum Bauen und Testen des lauffähigen SW-Produktes (Quellcode, Build-Skripte, Testdaten, Testfälle, ...)

\*\* Commit ist die Freigabe von eigenen Artefakten für alle Teammitglieder über die Versionsverwaltung

# CI ist eine Praktik...

...und erfordert das *Praktizieren* aller Teammitglieder.  
Das bedeutet fast immer einen Kulturwandel.

SW-Werkzeuge sind dabei notwendig,  
aber nicht hinreichend.

# Übersicht

- 1 Begriffsklärung
- 2 Vor- und Nachteile**
- 3 Fünf Schritte zur Einführung
- 4 Zusammenfassung
- 5 Diskussion

# Vorteile

## 1. Geringeres Risiko

Keine großen, unkalkulierbaren Integrationsschritte

## 2. Geringere Fehlerkosten

Schnelle, frühzeitige Fehlerentdeckung und -behebung

## 3. Höhere Mitarbeitermotivation

Sichtbar wachsendes System und eine beherrschbare Anzahl gleichzeitiger Fehler

## 4. Verfügbarkeit des aktuellsten Produktes

Verwendung für Demo-, Test- oder Vertriebszwecke

# Nachteile / Hürden

## 1. Kulturwandel

...um Entwicklungsteams für kleine Integrationsschritte und hohe Testautomation zu gewinnen

## 2. Hardwarekosten

...um einen schnellen Build (mit Tests) sicherzustellen

## 3. Anfängliche Entwicklungskosten

...um die erforderliche Infrastruktur aufzubauen

## 4. Wartungskosten für Legacy Code

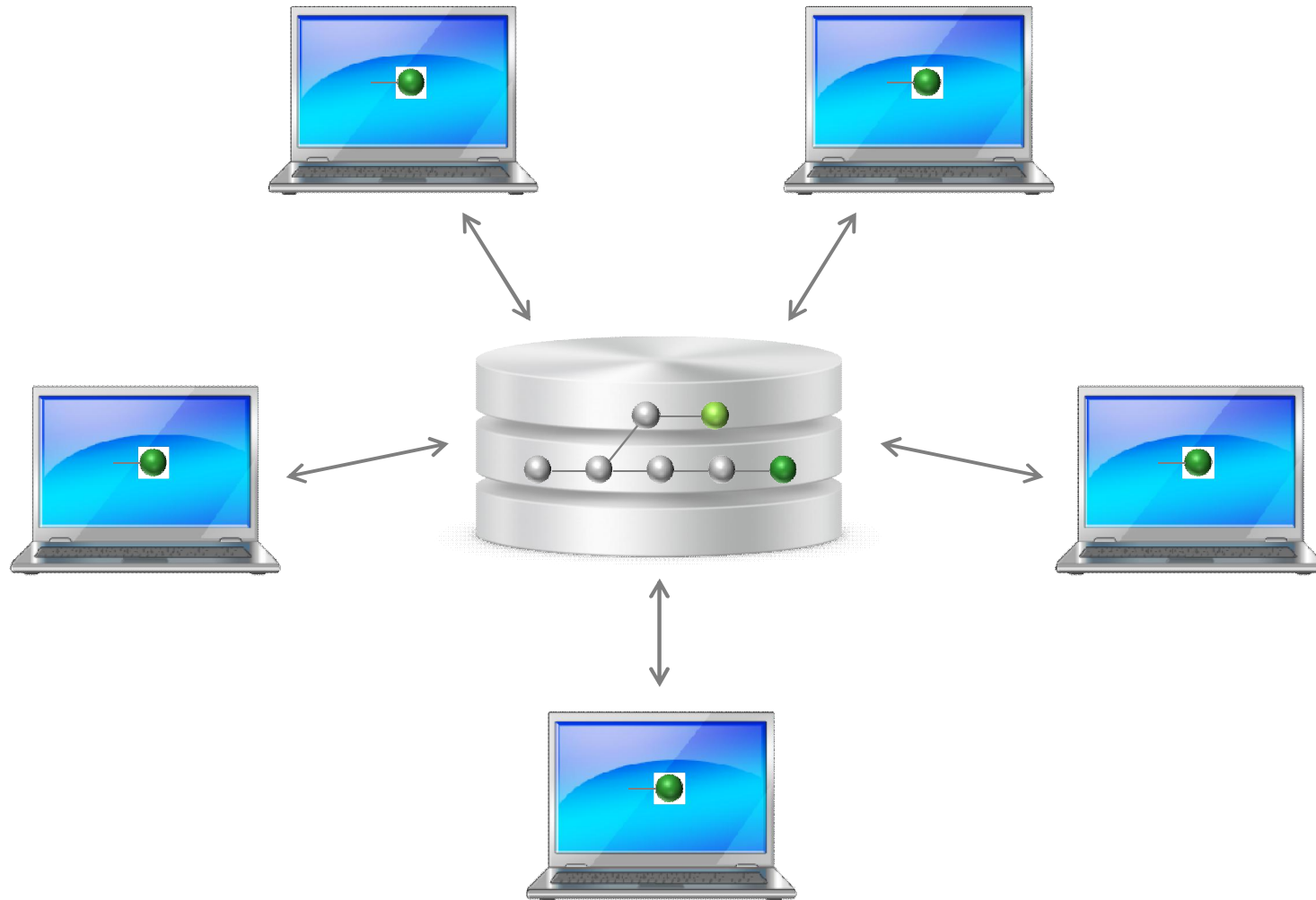
...die durch CI deutlich schneller sichtbar werden

# Übersicht

1	<b>Begriffsklärung</b>
2	<b>Vor- und Nachteile</b>
<b>3</b>	<b>Fünf Schritte zur Einführung</b> <ol style="list-style-type: none"><li>1. Versionsverwaltung</li><li>2. Gleiche Umgebungen</li><li>3. Automatischer Build</li><li>4. Weitere Testautomation</li><li>5. Systematische SITs*</li></ol>
4	<b>Zusammenfassung</b>
5	<b>Diskussion</b>

\* System- und Integrationstests

# Schritt 1: Versionsverwaltung



# Schritt 1: Versionsverwaltung

Technik und  
Organisation,  
lokale Projekte

- **Eine Versionsverwaltung einsetzen**  
Notwendige Features sind Atomare Commits, Rollbacks und gleichzeitiges Edieren
- **Alle nicht-generierten Artefakte** versionsverwalten  
Quell-Code, IDE-Konfigurationen, DB-Skripte, Testsuiten, ...
- **Generierte & externe Artefakte** separat archivieren  
Compilierter Code, Build-Tools, ext. Bibliotheken, ...
- **Fein-granulare Operationen** unterstützen  
Idealerweise: parallele Änderung zweier Methoden einer Klasse ohne anschließenden Merge

# Schritt 1: Versionsverwaltung

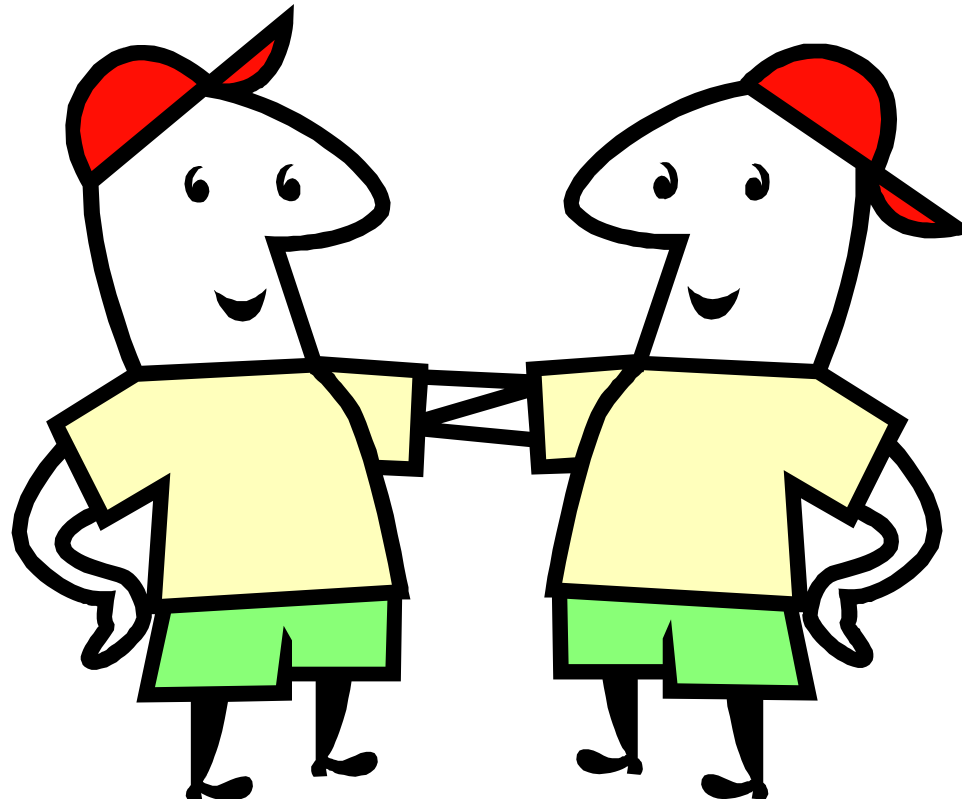
Technik und  
Organisation,  
verteilte Projekte

- Ausreichend **Bandbreite** zur Verfügung stellen  
Breitbandiger Internetzugang ist z.B. in Indien sehr teuer,  
aber für einen schnellen Abgleich mit dem Archiv unerlässlich

# Schritt 1: Versionsverwaltung

- Nur aus dem Archiv bauen  
Keine Dateien nehmen, die nur lokal beim Entwickler liegen
- Änderungen in **kleinen Schritten** committen  
Entwicklungsstände sichern, Zugriffskonflikte vermeiden, ...
- Durch Architektur Abhängigkeiten minimieren  
Versionierte Schnittstellen, fein-granulare Objekte, ...
- **Nicht branchen!**  
Versions-Verzweigungen nur aus triftigen Gründen erlauben  
Konfigurierbarkeit statt kundenspezifischem Code realisieren
- Täglich **Commitfrequenz** und **-größe** messen!

## Schritt 2: Gleiche Umgebungen



# Schritt 2: Gleiche Umgebungen

Technik und  
Organisation,  
lokale Projekte

- Gleiche **Entwicklungstools** verwenden  
Z.B. Compiler- und IDE-Versionen
- Einheitliche **Testumgebungen** pflegen  
Z.B. Patchlevel der Betriebssysteme, Datenbankversionen
- Spezifische Umgebungen im Projekt **einplanen**  
Auch wegen der Budgetbewilligung zu Projektbeginn nötig

## Schritt 2: Gleiche Umgebungen

Technik und  
Organisation,  
verteilte Projekte

- **Verfügbarkeit** und **Kosten** von HW & SW prüfen  
Sie können je nach Standort stark variieren
- **Vorlaufzeiten** für Genehmigungen einplanen  
Je nach Standort muss die Verwendung von HW oder SW genehmigt werden. Dies dauert sehr unterschiedlich lang
- **Sicherheitsrichtlinien** berücksichtigen  
Dazu müssen oft Umgebungen speziell konfiguriert werden

# Schritt 3: Automatischer Build

**Jenkins** | search | hudson\_guest | log out | ENABLE AUTO REFRESH

This is the view for the new ccs4 trunk build projects.

All Detailed Dashboard Maintenance build 2.3 Quality Dashboard **ccs4 trunk** +

S	W	Job ↓	Last Success	Last Failure	Last Duration
🌐	☀️	trunk Adapters FCSEA	2 days 1 hr (#27)	1 mo 5 days (#17)	42 sec
🌐	☁️	trunk Adapters GIMAV3			
🌐	☀️	trunk Adapters GIPA			
🌐	☁️	trunk Backends MBVM			
🌐	☀️	trunk Backends MBVM			
🌐	☀️	trunk BuildAll			
🌐	☀️	trunk Clients AdminWe			
🌐	☀️	trunk CommonJavaUtil			
🌐	☀️	trunk Facades CCSIAF			
🌐	☀️	trunk Monitoring			
🌐	☀️	trunk ProductDataImp			
🌐	☀️	trunk ProductDataUpd			
🌐	☁️	trunk RESTservices MD			
🌐	☁️	trunk RESTservices MD			
🌐	☀️	trunk RESTservices Me			

**Jenkins** | search | hudson\_guest | log out | ENABLE AUTO REFRESH

**Jenkins > Quality Dashboard**

All Detailed Dashboard Maintenance build 2.3 **Quality Dashboard** ccs4 trunk +

Warnings per project

Job ↓	Checkstyle	FindBugs	PMD	Total
🌐 trunk Adapters FCSEA	41	1	8	50
🌐 trunk Adapters GIMAV3	0	0	0	0
🌐 trunk Adapters GIPA	45	6	30	81
🌐 trunk Clients AdminWebUIV3	39	54	18	111
🌐 trunk CommonJavaUtilities	46	1	0	47
🌐 trunk Facades CCSIAFFacade	2	2	9	13
🌐 trunk Monitoring	0	0	2	2
🌐 trunk ProductDataImport	266	2	11	279
🌐 trunk ProductDataUpdate	13	3	8	24
🌐 trunk RESTservices MDSCCBB2Media	5	2	6	13
🌐 trunk RESTservices MDSCCBB2Render	1	1	6	8
🌐 trunk RESTservices MediaDirectoryV3	18	0	4	22
🌐 trunk SAPIProductDataUpdate	93	13	9	115
🌐 trunk ScheduledProcesses AvailabilityController	6	2	4	12
🌐 trunk ScheduledProcesses MonitoringAnalyzer	0	6	5	11
🌐 trunk ScheduledProcesses VehicleConfigurationStorageServiceDelete	0	1	0	1
🌐 trunk Webservices MCSAudi	20	2	19	41
🌐 trunk Webservices MSCCBB2Media	30	4	6	40
🌐 trunk Webservices MSCCBB2Render	6	0	3	9
🌐 trunk Webservices MCSMBVMedia	1	1	7	9
🌐 trunk Webservices MCSMOM	9	0	10	19
🌐 trunk Webservices PCSMBV	73	3	5	81
🌐 trunk Webservices PCSSAPI	78	4	25	107
🌐 trunk Webservices PCSSAPI	0	0	0	0

# Schritt 3: Automatischer Build

Technik und  
Organisation,  
lokale Projekte

- **One-button Build** entwickeln  
Das lauffähig SW-Produktes durch ein einziges Kommando vollständig generieren
- **Automatische Modultests** aufnehmen  
Es gibt sie natürlich schon ;-)
- Build-Ergebnis **allgemein sichtbar** machen  
CI-Tool (CruiseControl, Hudson, Jenkins, ...) einsetzen, das die Ergebnisse aufbereitet und darstellt bzw. per Mail verschickt

# Schritt 3: Automatischer Build

Technik und  
Organisation,  
verteilte Projekte

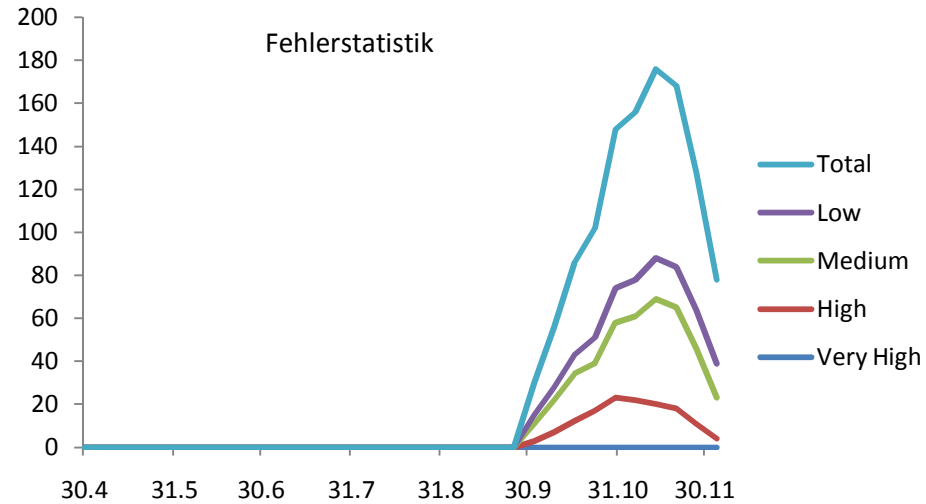
- **Dauer minimieren → Schnelles Feedback!**  
Build-Dauer sollte inklusive Modultests unter zwei Minuten bleiben: **inkrementell bauen**, Modultests optimieren, ...
- **Standort-Builds aufsetzen**  
...auf **einer** Versionsverwaltung; spart viel Zeit & Bandbreite
- **Kompetenz verteilen**  
Der automatische Build ist eine kritische Resource und muss unabhängig vom Standort hoch verfügbar sein
- **Build-Manager und Proxys benennen**  
Sie dürfen Entwickler anweisen, den Build zu reparieren

## Schritt 3: Automatischer Build

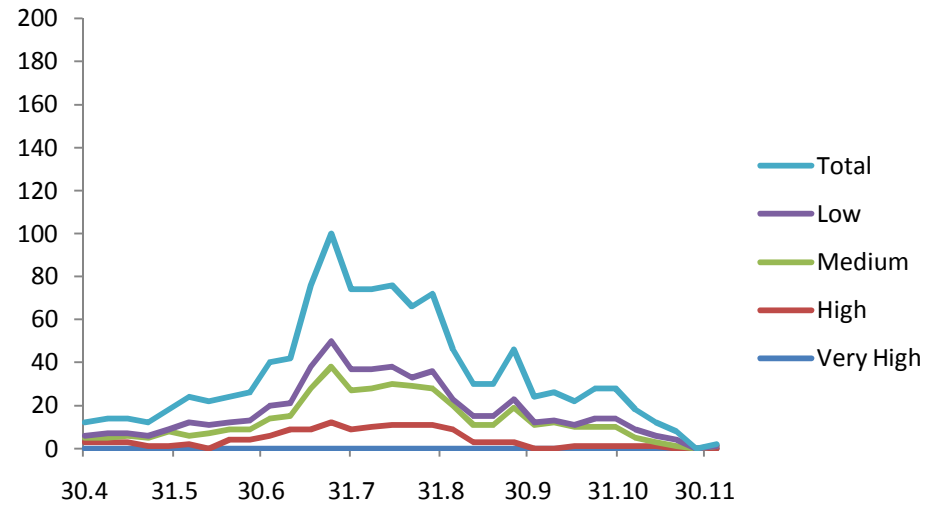
- **Lasst den Build heile!**  
Es bleibt dabei: häufig kleine Änderungen committen
- **Build-Reparatur hat höchste Priorität**  
Im Zweifelsfall macht der Build-Manager einen Rollback
- **Build-Manager und Proxys agieren als Coaches**  
In dieser Rolle erfährt man gut die Bedeutung des Builds
- **Bei jedem Build die Dauer, Testabdeckung, und Anzahl der Abbrüche messen!**

# Schritt 4: Weitere Testautomation

Einmaliger SIT\* zum  
Entwicklungsende



Kontinuierlicher SIT\*



\* System- und Integrationstest

# Schritt 4: Weitere Testautomation

Technik und  
Organisation,  
lokale Projekte

- **System- und Integrationstests (SIT) automatisieren**  
Evtl. Test-Frameworks wie FitNesse oder Twist verwenden
- **Build um Smoke-Tests erweitern**  
Dazu installiert der One-button Build das SW-Produkt  
zunächst automatisch in einer Referenzumgebung
- **Vor jedem Commit testen!**  
Jeder Entwickler testet seine Änderungen vor dem Commit  
mit den zusätzlichen automatischen Tests im Sandkasten
- **Testinfrastruktur erweitern**  
Haben die Entwickler genügend Sandkästen?

# Schritt 4: Weitere Testautomation

Technik und  
Organisation,  
verteilte Projekte

- **Testleiter** plant und koordiniert  
Lücken in der Testabdeckung schließen, Wiederverwendung von Tests fördern, ...
- **Build-Dauer minimieren**  
Mit Smoke-Tests dauert der Build erheblich länger: stärkere HW einsetzen, **inkrementell bauen**, Smoke-Tests parallel ausführen, ...

## Schritt 4: Weitere Testautomation

- Entwickler lernen SITs zu automatisieren  
Dazu müssen viele Entwickler erst Testgrundlagen erlernen
- Entwickler schreiben **automatisierte SITs**  
Was wird aus den traditionellen Testern?  
Welcher Grad an Testautomation läßt sich durchsetzen?
- Bei jedem Build die **Dauer, Testabdeckung** (für SIT)  
und **Anzahl der Abbrüche** messen!

## Schritt 5: Systematische SITs



# Schritt 5: Systematische SITs

Technik und  
Organisation,  
lokale Projekte

- SITs nach Aufgabe und Dauer klassifizieren  
Z.B. je eine Testklasse für Stabilitäts- und Performancetests
- Builds für Testklassen aufsetzen  
Beinhaltet die Planung, wann wo welche SITs laufen sollen;  
evtl. Testinfrastruktur erweitern

# Schritt 5: Systematische SITs

Technik und  
Organisation,  
verteilte Projekte

- Teams sind für **Featuregruppen** verantwortlich  
Im Agilen sind “cross-functional teams” üblich
- Tests nach Featuregruppen klassifizieren  
Dies bedeutet fast immer redundantes Testen
- In den Tools Featuregruppen unterstützen  
Von der Anforderungserfassung bis zum vollautomatisch generierten Testbericht

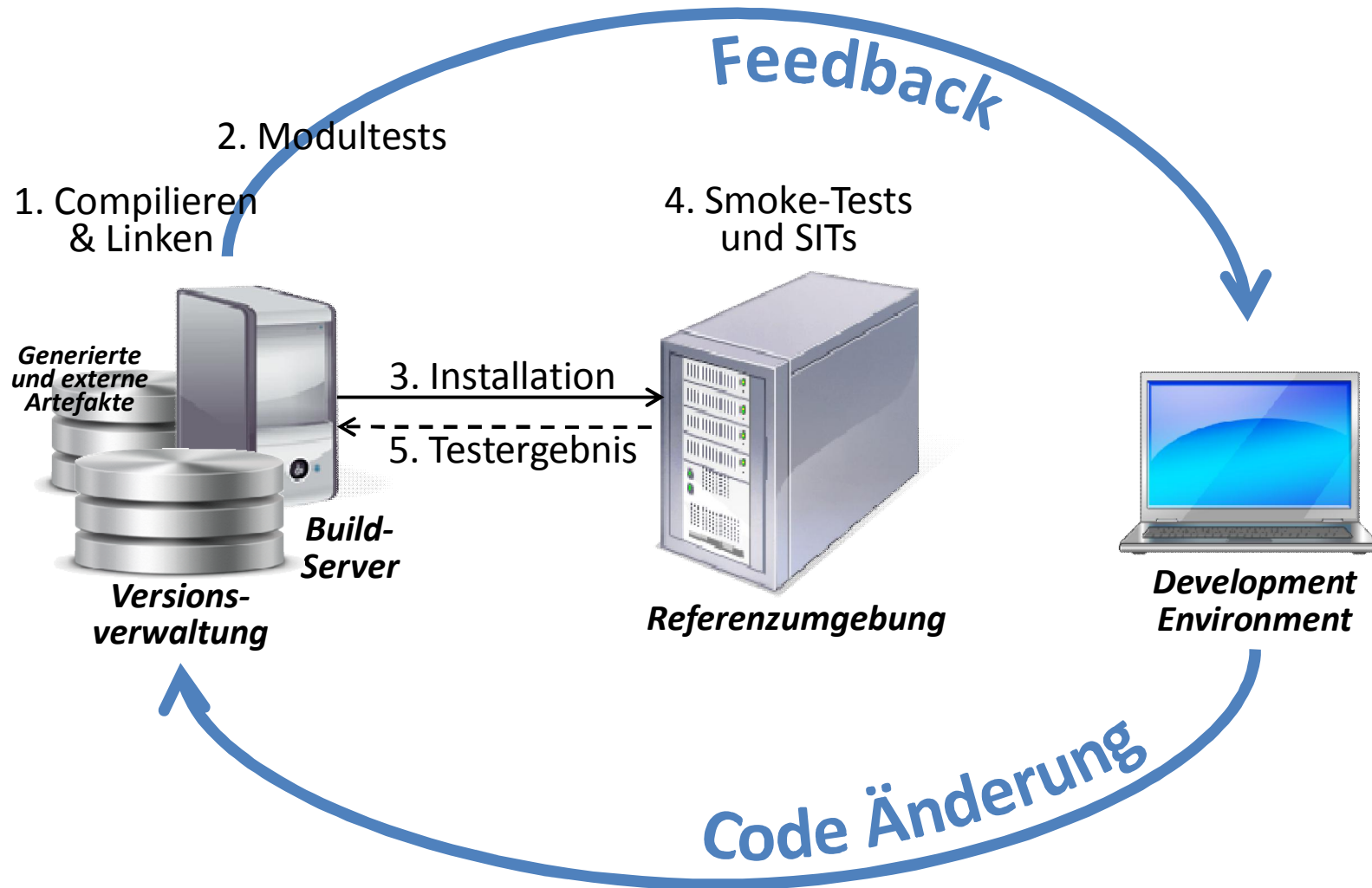
## Schritt 5: Systematische SITs

- Teammitglieder pflegen **Featuregruppen**  
Angabe der zugehörigen Featuregruppen in User Stories, in Commits, in SITs, ...
- Bei jedem Build messen (Dauer, Abdeckung, Anzahl Abbrüche) **nach Featuregruppe**

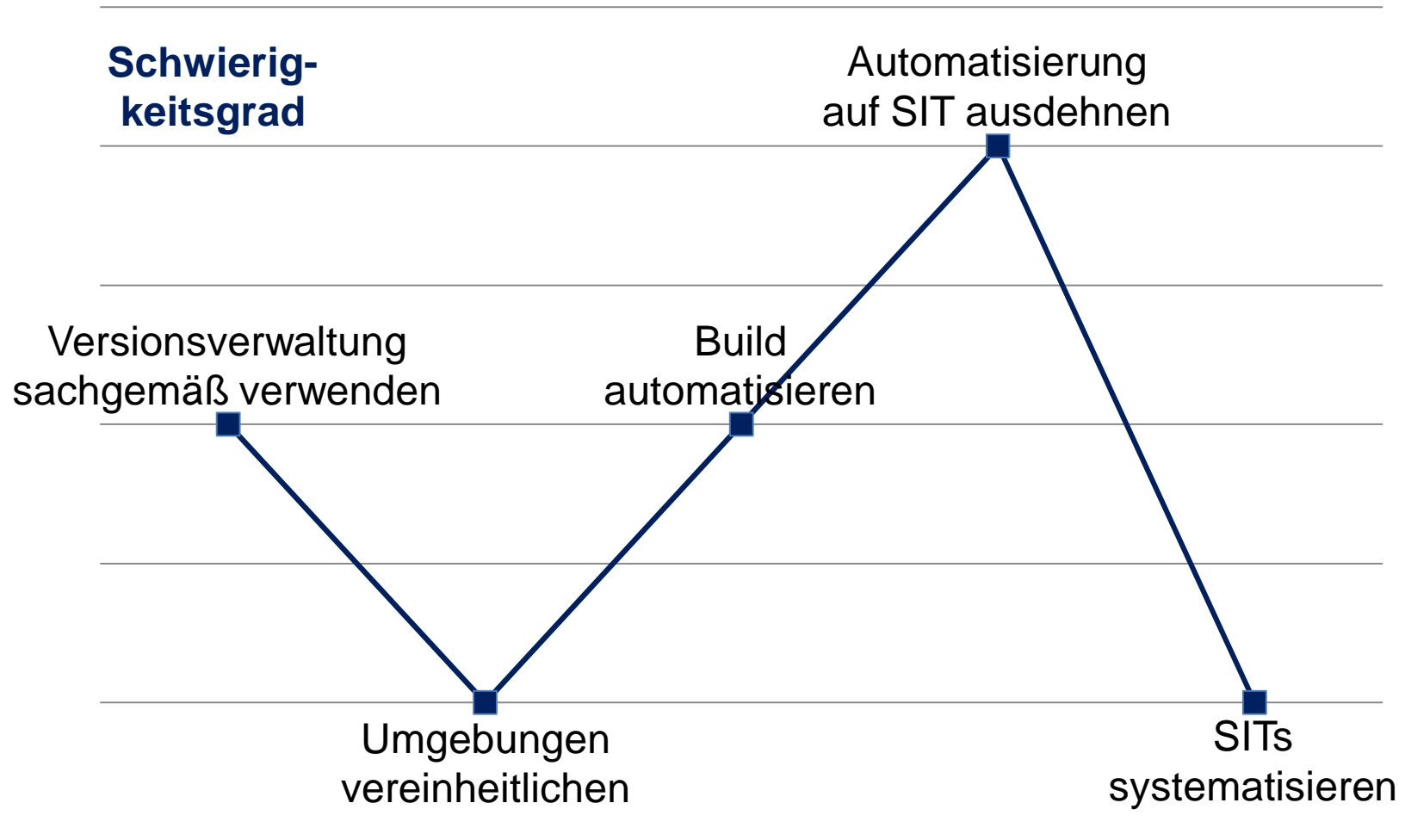
# Übersicht

- 1 **Begriffsklärung**
- 2 **Vor- und Nachteile**
- 3 **Fünf Schritte zur Einführung**
- 4 **Zusammenfassung****
- 5 **Diskussion**

# Zusammenfassung



# Zusammenfassung



## GERO SEIFERT

Program Manager  
TATA CONSULTANCY SERVICES

Tata Consultancy Services Deutschland GmbH  
Heltorfer Straße 1, D-40472 Düsseldorf  
Mobile +49 173-678 0618, Fax +49 211 91319-99  
Phone +49 211 91319-200  
gero.seifert@tcs.com      website: www.tcs.com

## Diskussion

---

Think Agile – Think Düsseldorf