# Requirements Engineering im agilen Umfeld

Verzichtbarer Luxus oder Schlüssel zum Erfolg?

**Johannes Bergsmann**
Software Quality Lab



# RE im Agilen Umfeld

Einleitung

## Agiles Manifest

„**Individuals and interactions** *over* processes and tools
**Working software** *over* comprehensive documentation
**Customer collaboration** *over* contract negotiation
**Responding to change** *over* following a plan"

Wobei der folgende Zusatz von großer Bedeutung ist, da er die Gewichtung relativiert:

*„That is, while **there is value in the items on the right**, we value the items on the left more."*





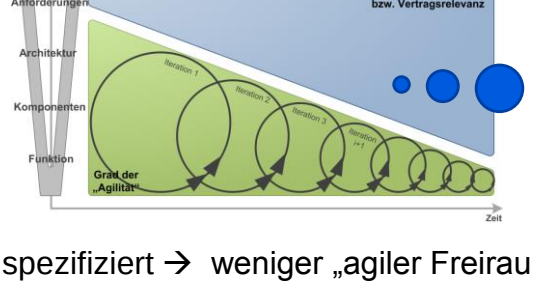
RE in agilen Methoden
Vorgehen in agilen Methoden

## Requirements-Engineering

- Braucht es im agilen Umfeld überhaupt Requirements-Engineering?
  - **„Ja", ohne RE sind auch agile Techniken nicht effizient!**
- Requirements-Engineering im agilen Umfeld ist ?
  - Das Erstellen von User-Stories,
  - Das Ausarbeiten von Feature- oder Use-Case-Beschreibungen
  - User-Interface-Prototypen (Mock-Up),
  - Das Klären von unklaren Anforderungen zwischen Auftraggeber (Product-Owner) und Entwicklern, Testern, etc. und
  - Die Feinspezifikation von Features z.B. im Rahmen des Sprint-Plannings
  - Das Erstellen einer Testspezifikation im Rahmen von test-getriebener Entwicklung
  - etc.

## Grad der Spezifikation bzw. Agilität

**Spezifikation bestimmt den Grad der Agilität**



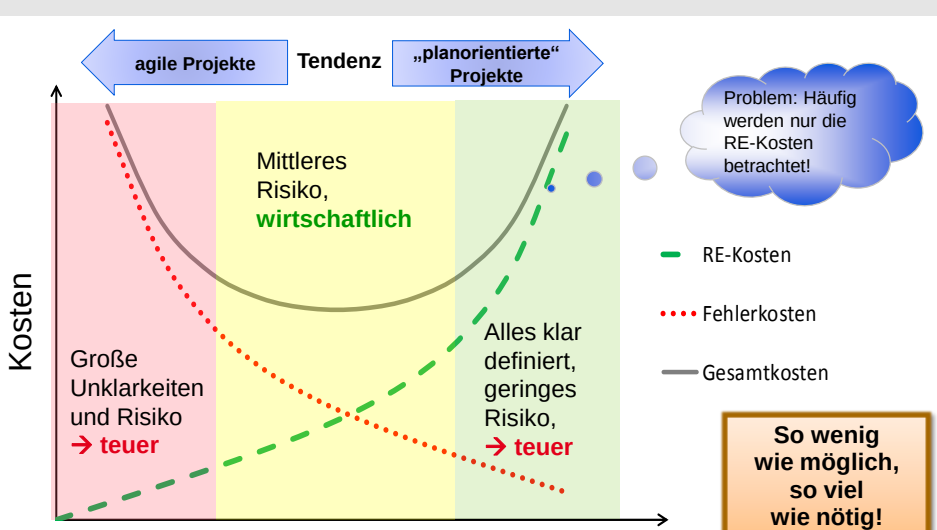

Achtung: Mehr Spezifikation ist nicht automatisch schlecht; mehr Agilität ist nicht automatisch gut!

- mehr spezifiziert → weniger „agiler Freiraum"
- Spezifikation auf unterschiedlichen Ebenen
- Agilität auf unterschiedlichen Ebenen



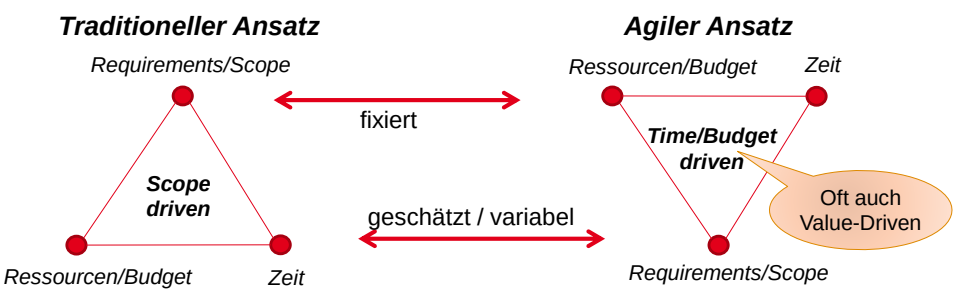


## Ausreichende Anforderungsqualität



## Wandel in der Projektbetrachtung

- Agiles Vorgehen fixiert **Datum** und **Ressourcen** und variiert die **Anforderungen bzw. Projekt-Scope**.




- Paramater **Qualität** (Tests, NFA, Nachhaltigkeit, …) nicht explizit angegeben! Fixiert oder variabel, je nach Einstellung von Team bzw. Organisation ! Hängt z.B. stark von der Qualität der Definition of Done ab.




# Techniken zur Requirements-Spezifikation

im agilen Umfeld

## Techniken zur Requirements-Spezifikation im agilen Umfeld

- User-Stories
- Use-Case Description
- Test-getriebene Entwicklung
- Specification by Example
- Behaviour-driven Development





# User-Stories

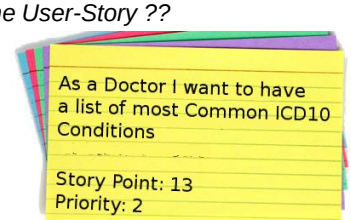
RE im Agilen Umfeld

## User-Stories - Schablone

- Als <**Benutzerrolle/Systemrolle**> will ich <das **Ziel / Story**> [, so dass <**Grund für das Ziel / Nutzen**>]

  Hinweis: Anwendung ist nicht unbedingt genau in dieser Syntax erforderlich!

- Beispiele
  - Als Projektmitglied will ich monatlich meine geleisteten Projektstunden abfragen.
  - Das System muss in Spitzenzeiten bis zu 50 User gleichzeitig unterstützen, da das System auch bei einer weiteren Expansion in den kommenden Jahren funktionieren muss. *?? Expertenstreit: Ist NFA eine User-Story ??*
  - Ein User kann die in seinem Konto gespeicherten Kreditkartendaten bearbeiten.




*Schlechtes Beispiel (typ. keine User-Anforderung):*

- Das System wird mit der Datenbank über einen Konnektor-Pool verbunden.





RE im Agilen Umfeld

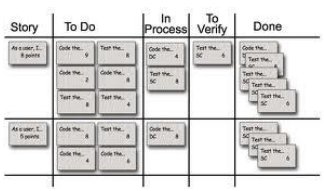
## User-Stories – RE-Hinweise

- **Story-Cards sind nur Darstellungs- und Diskussions-Hilfsmittel und KEINE angemessene Dokumentation von vertraglich vereinbarten Requirements!**
- Übergeordnete, grobe Requirements werden als *Epic* niedergeschrieben.
- User Stories sollten *übergeordnete Prozesse als Basis* haben.
- *Detailliertere Requirements* werden durch Teilung von User Stories spezifiziert oder
  können in zugeordneten *Use-Case-Descriptions* auch als Prozessablauf spezifiziert werden.

RE im Agilen Umfeld

## User-Stories – RE-Hinweise

- User-Stories repräsentieren nur eine bestimmte Ebene in der Spezifikationshierarchie
- User-Stories sind gute Hilfsmittel für die Grobspezifikation
- User-Stories sind gut als Strukturierungs- und Planungsinstrument geeignet




- Achtung:
  - Detailspezifikation wird oft vernachlässigt!
  - Übergeordnete Spezifikation & Zusammenhang wird oft vernachlässigt (vor allem Prozesse!)
  - Es werden nicht alle Aspekte einer guten Spezifikation abgedeckt (z.B. Architektur, Kontext, grafische Prozessmodellierung, Visualisierung, etc.)
  - → **Diesen Ansatz alleine ermöglicht keine vollständige Spezifikation!**





## Use-Case Description

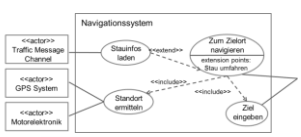
RE im Agilen Umfeld

## Use-Case Description

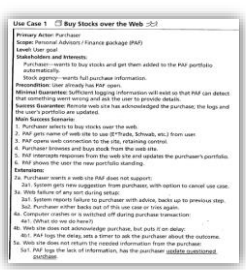
- Use-Case = Anwendungsfall =
  Verhalten des Systems (ggf. unter verschiedenen Bedingungen)

Unterscheidung:

- **Use-Case-Diagramm / Model**
  (sehr einfache grafische Notation)




- **Use-Case Description**
  (Textuelle Notation, die teilweise sehr komplex werden kann)



Use Case 1: Buy Stocks over the Web

Primary Actor: Purchaser

Scope: Personal Advisors / Finance package (PAF)

Level: User goal

Stakeholders and Interests:
- Purchaser—wants to buy stocks and get them added to the PAF portfolio automatically.
- Stock agency—wants full purchase information.

Precondition: User already has PAF open.

Minimal Guarantee: Sufficient logging information will exist so that PAF can detect that something went wrong and ask the user to provide details.

Success Guarantee: Remote web site has acknowledged the purchase; the logs and the user's portfolio are updated.

Main Success Scenario:
1. Purchaser selects to buy stocks over the web.
2. PAF gets name of web site to use (ETrade, Schwab, etc.) from user.
3. PAF opens web connection to the site, retaining control.
4. Purchaser browses and buys stock from the web site.
5. PAF intercepts responses from the web site and updates the purchaser's portfolio.
6. PAF shows the user the new portfolio standing.

Extensions:
2a. Purchaser wants a web site PAF does not support:
2a1. System gets new suggestion from purchaser, with option to cancel use case.
3a. Web failure of any sort during setup:
3a1. System reports failure to purchaser with advice, backs up to previous step.
3a2. Purchaser either backs out of this use case or tries again.
4a. Computer crashes or is switched off during purchase transaction:
4a1. (What do we do here?)
4b. Web site does not acknowledge purchase, but puts it on delay:
4b1. PAF logs the delay, sets a timer to ask the purchaser about the outcome.
5a. Web site does not return the needed information from the purchase:
5a1. PAF logs the lack of information, has the purchaser update questionable purchase.




RE im Agilen Umfeld

## Use-Case Description - Hinweise

- Use-Case Beschreibungen (textuell)
  - stellen schon eine relativ detaillierte Spezifikationsebene dar.
  - sind primär für prozessorientierte Abläufe gedacht

- Achtung:
  - Use-Case Beschreibungen sind in der reinen Textform nicht geeignet, um umfangreiche Zusammenhänge und Prozesse übersichtlich darzustellen!
  - Es werden nicht alle Aspekte einer Spezifikation abgedeckt (z.B. Architektur, Kontext, grafische Prozessmodellierung, Visualisierung, etc.) fehlen komplett.
  - → **Diesen Ansatz alleine ermöglicht keine vollständige Spezifikation!**

**Use-Case Beschreibungen sollten immer zusätzlich mit Prozessdiagrammen ,Interaktionsdiagrammen, Maskenprototypen, etc. visualisiert werden.**



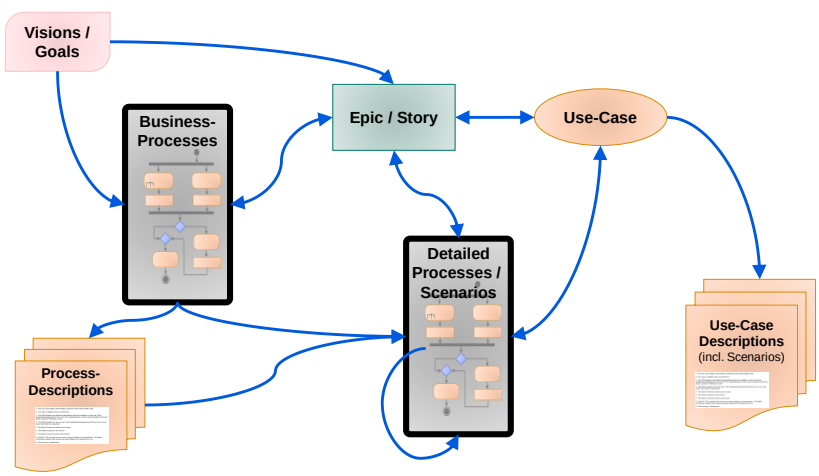



**Software Quality Lab GmbH**
www.software-quality-lab.com
info@software-quality-lab.com

**Linz** | A-4041 Linz | Gewerbepark Urfahr 30
**Wien** | A-1150 Wien | Mariahilfer Str. 136
**Graz** | A-8053 Graz, Wiener Straße 253
**München** | D-80992 München, Agnes-Pockels-Bogen 1

RE im agilen Umfeld

## Zusammenhänge








# Test-getriebene Entwicklung

RE im Agilen Umfeld

## Test-Driven - Motivation

- Situation: Die gewünschte und erforderliche Testabdeckung wird oft nicht erreicht.
- Testgetriebene Entwicklung versucht entgegenzuwirken.
- 2 Ebenen:
  - Testen im Kleinen (Unit-Tests) und
  - Testen im Großen (Systemtests, Akzeptanztests)





RE im Agilen Umfeld

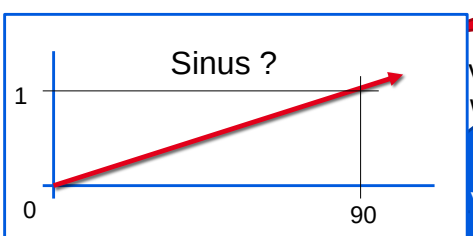
## Test-Driven - Eigenschaften

- Triviale Metrik für Erfüllung der Anf… Tests werden bestanden oder eben …

Aber Achtung: triviale Metrik birgt auch Gefahr der „Betriebsblindheit“!

- Weniger Risiko bei Änderungen, weil kleine Schritte entlang bestandener Tests.
- Führt tendenziell zu besser wartbarem und erweiterbarem Programmcode
- Testautomatisierung erhält einen großen Stellenwert
- **Unit-Tests und Systemtests dokumentieren das System**
  - → was das Softwaresystem leisten soll, liegt in Form lesbarer wie auch jederzeit lauffähiger Tests vor.
  - → Besser als schlecht gewartete Quellcode-Dokumentation
  - → **„ausführbare Spezifikation“**

## Test-Driven - Spezifikation




v. nicht automatisiert getestet und in
werden (z.

SPEC (im Kopf): Eine Sinus-Funktion soll implementiert werden.
Wieviel Tests sind nötig, um dasselbe Verständnis zu erreichen wie mit der Spec.?
TEST1: sin(90) = 1 → OK
TEST2: sin(0) = 0 → OK

etc.)

- Wie wird sichergestellt, dass die Tests / Spec. auch ausreichend sind?
- **Sich nur auf automatische Tests zu verlassen, ist nicht ausreichend!**

→ **Diesen Ansatz ermöglicht keine vollständige Spezifikation!**





## Specification by Example

RE im Agilen Umfeld

## Specification by Example

- Set an einfachen Prozess-Templates von Gojko Adzic
- Ziele / Vorteile:
  - Das für den Auftraggeber richtige Produkt möglichst effizient zu erstellen!
  - Immer eine aktuelle und gültige Spezifikation / Dokumentation haben.

**1. Deriving Scope From Goals**

- Nur wenn etwas erstellt wird, was auch einen Nutzen hat und ein Ziel unterstützt ist es effektiv!

**2. Specifying Collaboratively**

- Zusammenarbeit ALLER Betroffenen – Einbindung von Fachleuten, Entwickle
- Kollektive Beteiligung an der Spezifikationserstellung.

Sehr gut, um „in die Gänge“ zu kommen und die Anwender ins Boot zu holen!

**3. Illustrating using examples**

- Identifizieren von passenden konkreten Beispiel-Fällen für Funktionen, Aussehen, Sonderverhalten, etc. und Abstimmen der Beispiele mit den Beteiligten





RE im Agilen Umfeld

## Specification by Example

**4. Refining the specification**

- Offene Kommunikation über die Spezifikation / Beispiele und Überarbeitung
- Durch die Abstimmung und Verfeinerung werden autom. auch Akzeptanz-Kriterien definiert.

**→ Spezifikation ist gleichzeitig Anforderungs-Spezifikation, Akzeptanz-Test-Spezifikation und Basis für einen künftigen autom. Regressionstest.**

**5. Automating validation without changing specifications**

Siehe Test-Driven

- Die definierten und abgestimmten Beispiele werden als autom. Tests implementiert
  **= “ausführbare Spezifikation”**

**6. Validating frequently**

- Automatisierte Spezifikationen (Tests) können leicht auf Ihre Gültigkeit geprüft werden

**7. Evolving a documentation system**

- Automatisierte Spezifikationen (Tests) stellen eine „lebende“ Dokumentation dar
- **Dies ist das Endprodukt von „Spezifkation by Example“**







**Software Quality Lab GmbH**
www.software-quality-lab.com
info@software-quality-lab.com



**Linz** | A-4041 Linz | Gewerbepark Urfahr 30
**Wien** | A-1150 Wien | Mariahilfer Str. 136
**Graz** | A-8053 Graz, Wiener Straße 253
**München** | D-80992 München, Agnes-Pockels-Bogen 1

RE im Agilen Umfeld

## Specification by Example

- **Automatisierte Regressionstests = Spezifikation und Dokumentation**
  - Dadurch versucht man die Beteiligten dazu zu „zwingen“, die Spezifikation / Dokumentation immer aktuell zu halten (lebende Spezifikation)
- Hinweise:
  - Für Beispiele (Examples) gilt dasselbe, wie für sonstige Requirements-Spezifikationen: richtig, vollständig, konsistent, ausreichend, verständlich, etc.
  - Nur weil Durchlauf der autom. Tests „grün“ ist, heißt das nicht, dass alles gut ist! Beispiele =ca.= Testfälle → gleiches Problem wir bei Test-Driven!
  - Spezielle Frameworks zur Automatisierung sind dafür Voraussetzung
  - **Dieser Ansatz ermöglicht nur eine reduzierte Spezifikation!**
    Es werden nicht alle Aspekte einer guten Spezifikation abgedeckt
    (z.B. Architektur, Kontext, grafische Prozessmodellierung, Visualisierung)

  → **der Ansatz eignet sich primär für stark funktionsorientierte Spezifikationen**





RE im Agilen Umfeld

## Specification by Example

Zitate von **Martin Fowler** zum Nachdenken:

- Specification By Example isn't the way most of us have been brought up to think of specifications.
- Specifications are supposed to be general, to cover all cases.
- **Examples only highlight a few points**,
  you have to infer the generalizations yourself.
- This does mean that Specification By Example **can't be the only requirements technique you use**.

Quelle: http://martinfowler.com/bliki/SpecificationByExample.html

# Behavior Driven Development (BDD)






## Behaviour Driven Development (BDD)

- 2003 von Dan North als Antwort auf Test-Driven Development
  (Unit-Tests sind für Anwender schlecht lesbar)
- erstellt User-Stories und erfasst dazu auch das Verhalten (Behaviour) mit Vor-&Nachbedingungen
- Beschreibung erfolgt in „natürlicher Sprache“:

User-Story

Verhalten

→ **Nicht-Programmierer schreiben „Code“ vor der eigentlichen Entwicklung**

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

Scenario: Add two numbers
  Given I have entered 50 into the calculator
  And I have entered 70 into the calculator
  When I press add
  Then the result should be 120 on the screen
```

Vorbedingung

Verhalten

Nachbedingung

RE im Agilen Umfeld

## Behaviour Driven Development (BDD)

Beispiel Cucumber (Tool für BDD, Codebasis: Ruby)

**1: Describe behaviour in plain text**

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario: Add two numbers
    Given I have entered 50 into the calculator
    And I have entered 70 into the calculator
    When I press add
    Then the result should be 120 on the screen
```

**2: Write a step definition in Ruby**

```
Given /I have entered (.*) into the calculator/ do |n|
  calculator = Calculator.new
  calculator.push(n.to_i)
end
```

**3: Run and watch it fail (Class Calculator missing)**

```
$ cucumber features/addition.feature
Feature: Addition  # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers                    # features/additi
    Given I have entered 50 into the calculator  # features/step_d
      uninitialized constant Calculator (NameError)
      ./features/step_definitons/calculator_steps.rb:2:in `Given /
      features/addition.feature:7:in `Given I have entered 50 into
    And I have entered 70 into the calculator    # features/step_d
    When I press add                             # features/additi
    Then the result should be 120 on the screen  # features/additi
```

**4. Write code to make the step pass**

```
class Calculator
  def push(n)
    @args ||= []
    @args << n
  end
end
```

**5. Run again and see the step pass**

```
$ cucumber features/addition.feature
Feature: Addition  # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers                    # features/additi
    Given I have entered 50 into the calculator  # features/step_d
    And I have entered 70 into the calculator    # features/step_d
    When I press add                             # features/additi
    Then the result should be 120 on the screen  # features/additi
```

**6. Repeat 2-5 until green like a cuke**

Quelle: http://www.cukes.info/

**7. Repeat 1-6 until the money runs out**







RE im Agilen Umfeld

## Behaviour Driven Development (BDD)

- Automatisierte Regressionstests geben schnelles Feedback = **„lebende“ Spezifikation und Dokumentation**

- BDD: interessanter Ansatz, um enge Kopplung zwischen den Beteiligten zu erreichen.
- Hinweise:
  - Für BDD gilt dasselbe wie für andere „textlastige Spezifikationstechniken“: richtig, vollständig, konsistent, ausreichend, verständlich, etc.
  - Spezielle interpretierbare Spezifikations-Frameworks sind dafür Voraussetzung (z.B. Cucumber, JBehave, etc.)
  - **Diesen Ansatz ermöglicht keine vollständige Spezifikation**
    Es fehlen Aspekte einer guten Spezifikation wie z.B. Architektur, Kontext, grafische Prozessmodellierung, Visualisierung, Usability, etc.

  → **eignet sich primär für stark funktionsorientierte Spezifikationen**







**Software Quality Lab GmbH**
www.software-quality-lab.com
info@software-quality-lab.com

**Linz** | A-4041 Linz | Gewerbepark Urfahr 30
**Wien** | A-1150 Wien | Mariahilfer Str. 136
**Graz** | A-8053 Graz, Wiener Straße 253
**München** | D-80992 München, Agnes-Pockels-Bogen 1

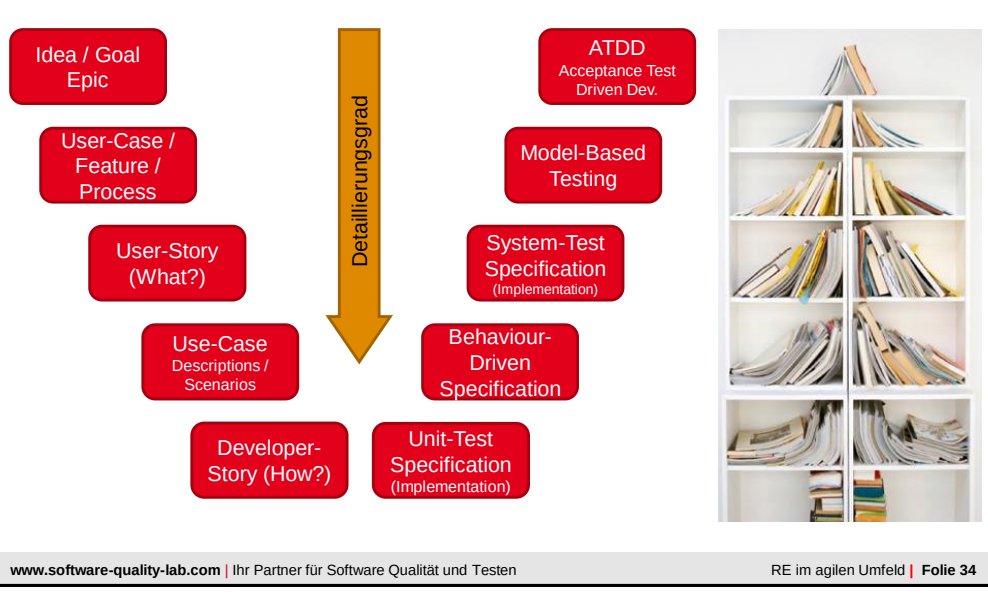
# Wie passt das alles zusammen?







## Wie passt das alles zusammen?



# Worauf sollte man bei der Spezifikation im agilen Kontext noch aufpassen?

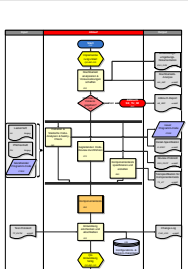


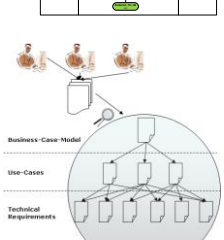


RE in agilen Vorgehensweisen

## Worauf sollte man noch aufpassen?

- Abläufe nicht vergessen!
  Geschäftsprozesse, Use-Cases, Aktivitätsdiagramme, Interaktionsdiagr., etc.
  Nur textuelle Beschreibungen sind meist zu wenig!
- System-Architektur sollte auch dokumentiert werden
- Schnittstellen zu anderen Systemen (möglichst bald)
- Visualisierung und Prototyping ist essentiell !!
  z.B. für Detailspezifikationen, Zusammenhänge, Strukturen, Design, User-Sicht, Screen-Mockups, Reports, Schnittstellen, Entscheidungen, etc.






- Top-Down-Überblick wird oft vernachlässigt!
- Gesamtüberblick geht oft verloren
  (Benutzer- & technische Gesamt-System-Dokumentation?)
- „agile“ Tools sind zur Spezifikation von Zusammenhängen oft nicht geeignet – Tracking-Systeme eigenen sich nur begrenzt zur Requirements-Spezifikation!

# weitere RE-Themen im agilen Umfeld

Überblick





RE in agilen Methoden

## RE-Risiken bei agilem Vorgehen

- Kunde (Kundenvertreter) nimmt sich oft zu wenig Zeit!
  - Es ist oft schwierig, Kundeninput genau dann zu bekommen, wenn ihn das (Entwickler-)Team benötigt. Kunden sind ausgelastet mit „Tagesgeschäft“.
- Testen & Testautomatisierung wird unterschätzt!
  - Testautomatisierung in großem Umfang ist Voraussetzung.
  - Wer gibt Teststrategie, Vorgehensweise, etc. übergreifend vor?
  - Wer testet auf Gesamtsystemebene?

- Die vertragliche Basis für die Arbeit ist unklar!
  - Vor allem, wenn nur stichwortartige Features und Story-Cards verwendet werden.
  - Gegen was nimmt der Kunden am Schluss das System ab?
  - Was ist, wenn am Ende ein anderer Kundenvertreter sagt, dass das System noch nicht passt / fertig ist?
  - Was passiert, wenn der „Entscheider“ beim Kunden wechselt?

RE in agilen Methoden

## RE-Risiken bei agilem Vorgehen

- Es ist oft nicht klar, wer die Änderungsaufwände zahlt!
- Dokumentation ist oft unzureichend!
  - Gesamt-System- und Architektur-Dokumentation/Überblick
  - Feature-Details – was kann mein System nun wirklich alles?
  - Change-Traceability – Warum hat wann wer was geändert?
- Ohne Spezifikationen ist es schwierig, Personen, die nicht ständig im Team sind, in den agilen Prozess einzubinden.
  - Wie schaut die Usability, Masken/Reports im Kopf der Beteiligten aus, wenn es nicht niedergeschrieben ist? Bei allen gleich?
  - Wie interpretieren 3 Entwickler an unterschiedlichen Standorten die Story „Als Benutzer wünsche ich mir, dass eine Monatsübersicht ausgedruckt werden kann.“?
  - Etc.



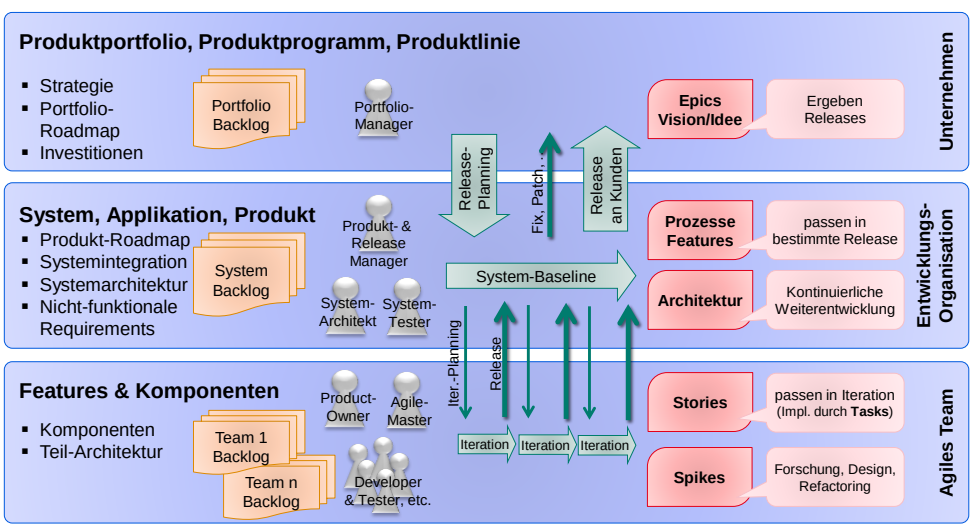


## Das „Big-Picture“

für RE im Agilen Bereich

RE im agilen Umfeld

## Big Picture




In Anlehnung an Dean Leffingwell





RE im agilen Umfeld

## Big Picture - Hinweise

- Komplexität des Unternehmens (Management-Ebenen) auch in agilen Vorgehensweisen wichtig und zu berücksichtigen!
- Übergeordnete Architektur, Integrations- & Test-Verantwortung ist notwendig auf Ebene des Produkt-Managements
- Nicht-Funktionale Requirements teamübergreifend behandeln
- System- & Integrationstest (zus.) übergeordnet zu agilen Teams
- Rolle des Product-Owners ist in größeren Entwicklungs-Organisationen zu relativieren:
  - Für Gesamt-Produkt verantwortlich ist der Produkt-Manager
  - Product-Owner im agilen Team ist „Requirements-Verantwortlicher“ für die jeweils ausgewählten Features → Abstimmung mit Produkt-Manager

RE in agilen Vorgehensweisen

## Zusammenfassung

- Requirements-Engineering nimmt in agilen Vorgehensweisen einen wichtigen Platz ein!
- Agiles RE ist mehr als „nur“ User Stories erstellen.
- Es gibt verschiedene RE-Techniken in agilen Vorgehensweisen
  - User-Stories, Use-Case Beschreibungen, Test-getriebene Entwicklung
  - Specification by Example, Behaviour-driven Development
- Die derzeit aktuellen „agilen“ Spezifikationstechniken sind primär für funktionsorientierte Beschreibungen geeignet.
- Es fehlen einige wesentliche Spezifikationselemente-/sichten
  - z.B. System-Kontext, grafische Prozessmodellierung, Benutzeroberflächen-Visualisierung, Schnittstellen-Kontext, nicht funktionale Anforderungen, Usability, etc.
- In größeren Entwicklungsorganisationen oder Projekten (> 1 agiles Team) ist die Komplexität auch im RE entsprechend zu berücksichtigen.





# Software Quality Lab GmbH

Ihr Partner für Software Qualität und Testen

Mariahilfer Straße 136
A-1150 Wien
**[T]** +43 732 890072-0
**[F]** +43 732 890072-411
**[W]** www.software-quality-lab.com