XCEPTANCE
Software Technologies

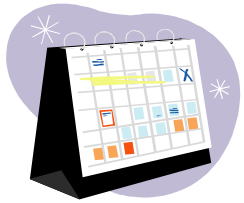
Software-Qualitätssicherung, Teil II: Test in Agilen Prozessen

Friedrich-Schiller-Universität Jena, Wintersemester 2017/2018
Ronny Vogel, Xceptance GmbH

Agenda

- Agile Vorgehensmodelle und Scrum
- Agile QS – Erster Ansatz
- Agile QS – Heute bewährt: Tester im Team
- QS in Scrum

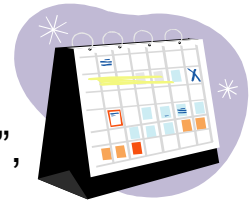
Agile Vorgehensmodelle und Scrum (1)



Historie

- viele Vorgänger, kein einzelner Ursprung
- 1957: erste inkrementelle SW-Entwicklungsmethoden bei IBM
- 1970er: Veröffentlichungen zu Konzepten inkrementeller SW-Entwicklung
- 1986: Begriff “Scrum” erstmals im Artikel “New New Product Development Game” von H. Takeuchi und I. Nonaka in einer Analogie benutzt
- 1988: Tom Gilb beschreibt “Evolutionary Development (EVO)” in seinem Buch “Principles of Software Engineering Management”
- 1990er: diverse Ansätze zu “leichtgewichtigen SW-Entwicklungsmethoden”

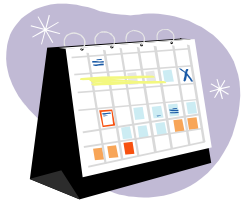
Agile Vorgehensmodelle und Scrum (2)



Historie (fortgesetzt)

- 1993: Ken Schwaber nutzt “Advanced Development Methods (ADM)”, die sich später zu “Scrum” weiterentwickeln
- 1993: erstes “Scrum”-Team um Jeff Sutherland bei der Easel Corporation
- 1995: erste öffentliche Präsentation von “Scrum” durch Ken Schwaber und Jeff Sutherland auf der OOPSLA '95 in Austin, Texas
- 1995: “Adaptive Software Development (ASD)”, “Feature Driven Development (FDD)”, “Dynamic Systems Dev. Method (DSDM)”
- 1996: “Crystal Clear”, “Extreme Programming (XP)”
- 1999: erste wirkliche Popularität Agiler Methoden durch das Buch “Extreme Programming Explained” von Kent Beck

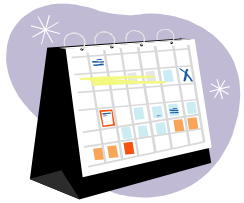
Agile Vorgehensmodelle und Scrum (3)



Historie (fortgesetzt)

- 2001: auf einem Treffen diskutieren 17 Softwareentwickler leichtgewichtige Vorgehensmodelle
  - die erstellen das "*Manifest für Agile Softwareentwicklung*" basierend auf *"Zwölf Prinzipien Agiler Softwareentwicklung"*
  - sie ersetzen den Begriff "*leichtgewichtig*" durch "*agil*"
  - sie prägen den Begriff "*Agile Vorgehensmodelle*"
- 2001: erstes Buch "Agile Software Development with Scrum" von Ken Schwaber und Mike Beedle
- 2003: erste zertifizierte Scrum Master
- November 2017: Aktualisierung von "The Scrum Guide" von Ken Schwaber und Jeff Sutherland, "Scrum Values" zugefügt, siehe www.scrumguides.org

Agile Vorgehensmodelle und Scrum (4)



Heute

- andauernde Evolution agiler Vorgehensmodelle
- noch immer gewisse Vielfalt von Methoden, Werkzeugen, Techniken und sich ändernden Meinungen - aber alle basieren auf ähnlichen Ideen
- Scrum mit Abstand am häufigsten eingesetzt
- agile Vorgehensmodelle weithin akzeptiert und genutzt
- VersionOne,”10th Annual State of Agile Development Survey” 2016: 95% aller Unternehmen nutzen agile Prozesse
- meist keine lehrbuchhafte Anwendung - agile Projekte wählen ein agiles Vorgehensmodell aus und passen es für ihren Zweck an

Agiles Manifest – Agile Werte

"Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:"

Individuen und Interaktionen	*mehr als*	*Prozesse und Werkzeuge.*
Funktionierende Software	*mehr als*	*umfassende Dokumentation.*
Zusammenarbeit mit dem Kunden	*mehr als*	*Vertragsverhandlung.*
Reagieren auf Veränderung	*mehr als*	*das Befolgen eines Plans.*

"Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein."

- in 2001 von 17 Softwareentwicklern erstellt
- agilemanifesto.org (en), agilemanifesto.org/iso/de/manifesto.html (de)

Agiles Manifest – Agile Prinzipien

Zwölf Prinzipien agiler Softwareentwicklung

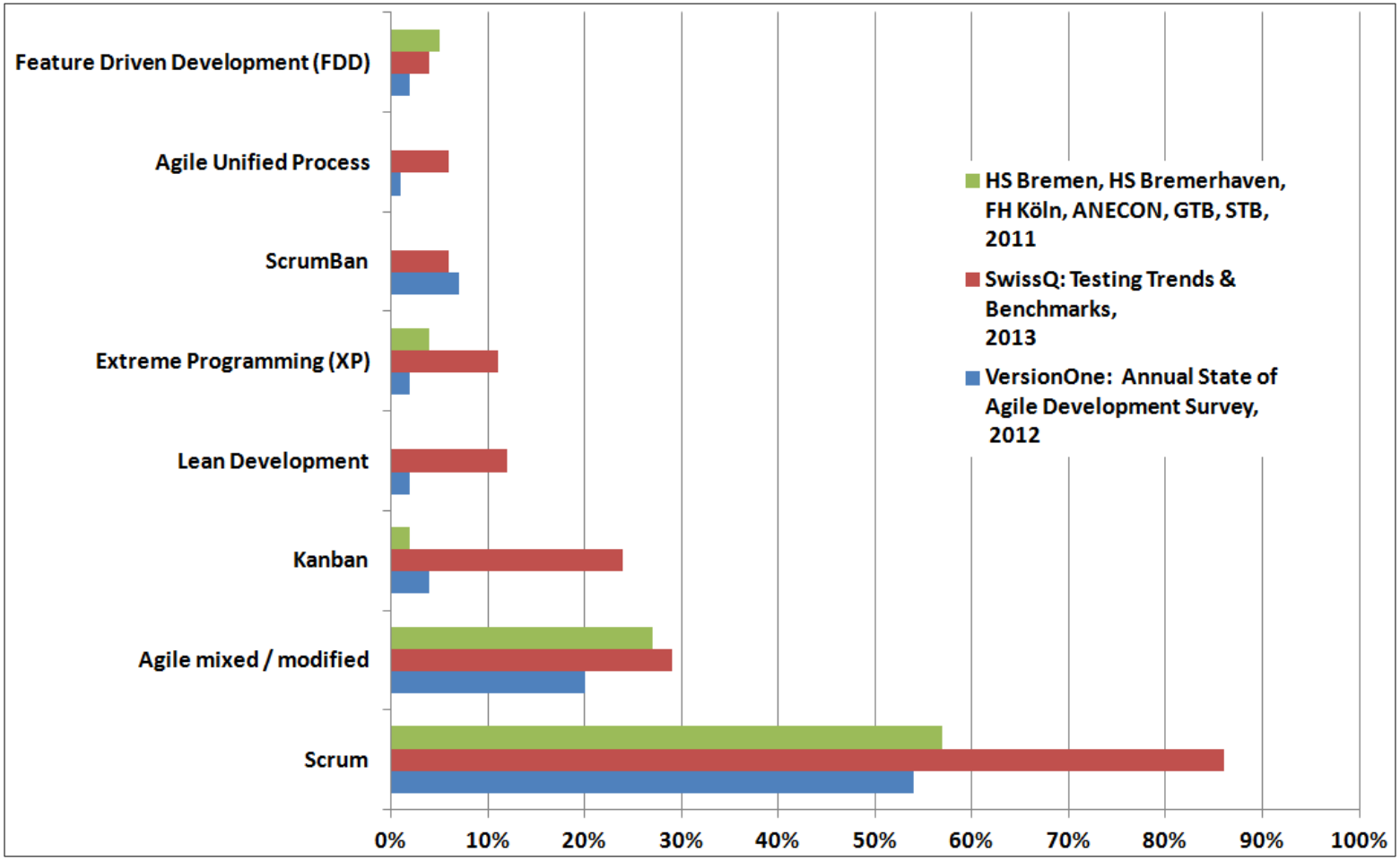
- Kundenzufriedenheit durch kontinuierliche Auslieferung nutzbarer Software
- heiße Anforderungsänderungen selbst spät in der Entwicklung willkommen; agile Prozesse nutzen Veränderungen zum Vorteil des Kunden
- liefere funktionierende Software regelmäßig aus (in Wochen statt Monaten), bevorzuge dabei die kürzere Zeitspanne
- funktionierende Software ist das wichtigste Fortschrittsmaß
- nachhaltige Entwicklung mit gleichmäßigem Tempo auf unbegrenzte Zeit
- enge, tägliche Zusammenarbeit von Fachexperten und Entwicklern

Agiles Manifest – Agile Prinzipien

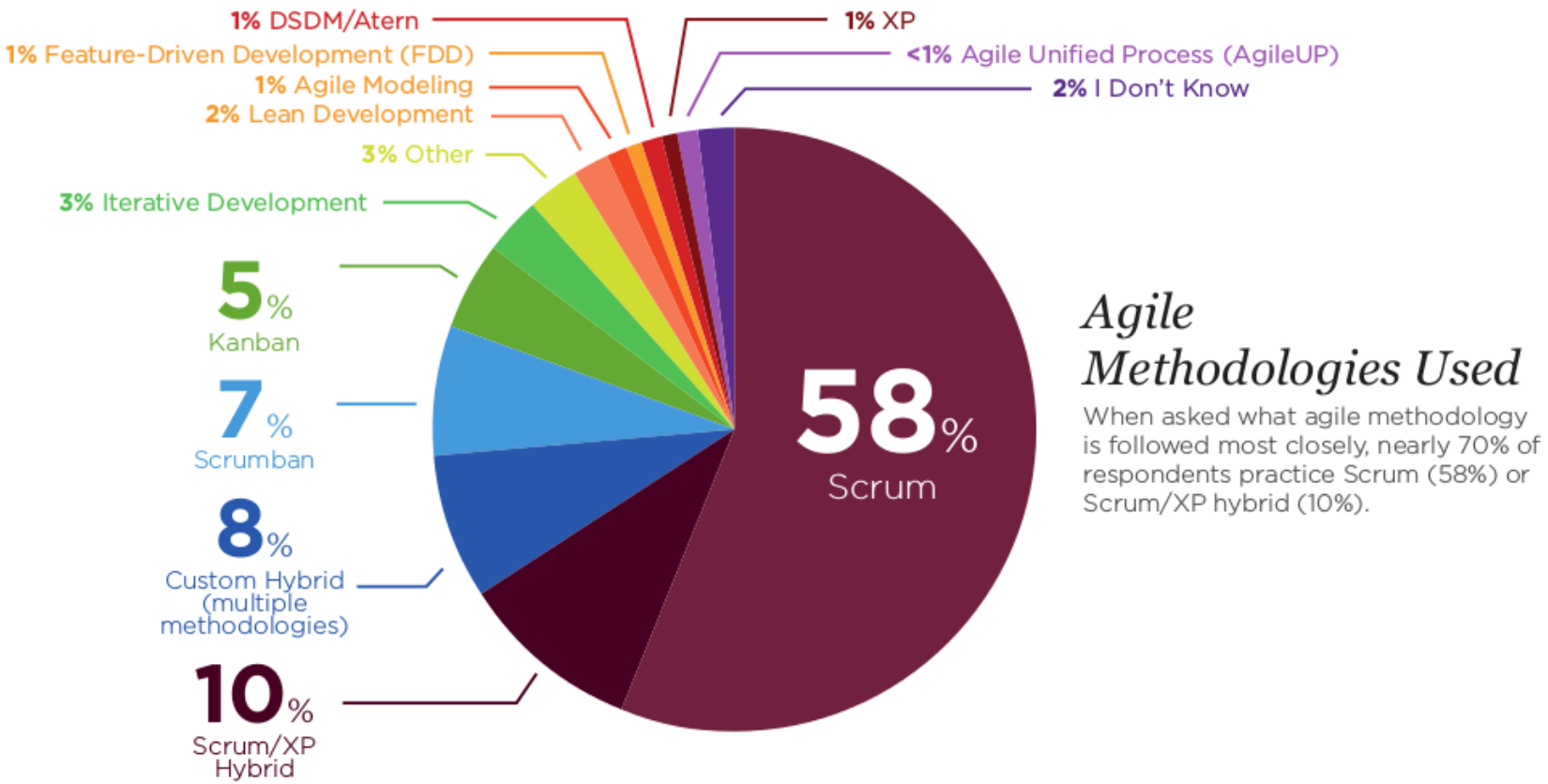
Zwölf Prinzipien agiler Softwareentwicklung (fortgesetzt)

- direkte Gespräche sind die effizienteste und effektivste Form der Informationsübermittlung (➔ Co-Location)
- errichte Projekte rund um motivierte Individuen, gib ihnen das benötigte Umfeld und vertraue darauf, dass sie die Aufgabe erledigen
- ständiges Augenmerk auf technische Exzellenz und gutes Design
- Einfachheit - die Kunst, die Menge nicht getaner Arbeit zu maximieren - ist essenziell
- die besten Ergebnisse entstehen durch selbstorganisierte Teams
- regelmäßige Reflexion der Teams, wie sie effektiver werden können, und Anpassung ihres Verhaltens

Nutzung diverser Agiler Methoden (1)



Nutzung diverser Agiler Methoden (2)



Quelle: VersionOne.com, 10th Annual State of Agile Report, 2016

Scrum

Was ist Scrum?

- Rahmenwerk und Vorgehensmodell zur Entwicklung und Erhaltung komplexer Produkte
- weder ein Prozess noch eine Technik zur Erstellung von Produkten
- schreibt keine konkreten Entwicklungspraktiken vor, sondern erlaubt den Einsatz verschiedener Prozesse und Techniken

Laut „The Scrum Guide“ ist Scrum:

- leichtgewichtig
- einfach zu verstehen
- schwierig zu meistern

Scrum

Basiert auf der Theorie empirischer Prozesssteuerung

- „Empirie“
  - Wissen wird aus Erfahrung gewonnen
  - Entscheidungen werden auf Basis des Bekannten getroffen
- Annahmen
  - die meisten modernen Entwicklungsprojekte sind zu komplex, um durchgängig planbar zu sein
  - wichtige Faktoren sind unberechenbar
  - der perfekte Plan existiert nicht

    ➔ Unsicherheiten und Veränderungen werden akzeptiert

    ➔ Verbesserung von Schätzungen und Verringerung von Risiken durch iterativen, inkrementellen Ansatz

Scrum

Drei Prinzipien der empirischen Prozesssteuerung

Transparenz:
- gemeinsame Sprache
- gemeinsames Verständnis
- wahrheitsgetreue Sichtbarkeit von Fortschritt und Hindernissen

Überprüfung/Inspektion:
- regelmäßige Prüfung der Scrum-Artefakte und des Fortschritts
- Erkennung ungewollter Abweichungen

Anpassung:
- Aspekte des Produktes und des Prozesses werden regelmäßig neu bewertet und bei Bedarf angepasst

Scrum

Ziel

- schnelle, kostengünstige und qualitativ hochwertige Fertigstellung eines Produktes, das einer zu Beginn formulierten Vision entsprechen soll

- Umsetzung der Vision in mehreren Iterationen, je maximal vier Wochen, heute typisch zwei Wochen
  ➔ *Sprints*

- Lieferung einer fertigen Software-Funktionalität am Ende jedes Sprints
  ➔ *Produkt-Inkrement*

- die neu entwickelte Funktionalität sollte in einem Zustand sein, dass sie an den Kunden ausgeliefert werden könnte
  ➔ *Potentially Shippable Code* oder *Usable Software*

Scrum

Das Scrum-Rahmenwerk beschreibt

- drei interne Rollen + drei externe Rollen (später zugefügt)
- Ereignisse/Aktivitäten
  - Sprint Planning
  - Daily Scrum
  - Sprint Review
  - Sprint Retrospective
- Artefakte
  - Product Backlog
  - Sprint Backlog
  - Burn-Down Chart
- Regeln, die Ereignisse und Artefakte miteinander verbinden
- Aktivitäten dienen der unmittelbaren Vorbereitung der nächsten Aktivität
- alle Aktivitäten sind zeitlich beschränkt (*timeboxed*)

Scrum-Rollen im Überblick

Drei interne Kernrollen, das “Scrum Team”:

- Product Owner
- Scrum Master
- Entwicklungsteam

Drei externe Rollen:

- Management
- Kunden
- Nutzer

Kein traditioneller Projektmanager!

➔ Team bestimmt Aufgabenverteilung und Zusammenarbeit selbst

➔ verbleibende Managementaufgaben auf diverse Rollen verteilt

Scrum-Rollen

Rolle „Product Owner“ (PO)

- verantwortet Projekterfolg und Erreichung der wirtschaftlichen Ziele
- kontinuierliche Festlegung und Priorisierung der Anforderungen mit Hilfe des Product Backlog
- darin pflegt er in Zusammenarbeit mit dem Entwicklungsteam die User Stories
- User Stories beschreiben Funktionalitäten aus der Sicht des Benutzers
- der PO versucht, Kundenbedürfnisse und Wünsche optimal in die Entwicklung einfließen zu lassen, ist aber kein Vertreter des Kunden

Scrum-Rollen

Rolle „Product Owner“ (fortgesetzt)

- die Festlegungen des Product Owners sind verbindlich
- nimmt beim Sprint-Review das Entwicklungsergebnis ab
- Entscheidung darüber, ob die vom Entwicklungsteam am Ende jedes Sprints gelieferte Funktionalität akzeptiert wird
- entscheidet über Auslieferungszeitpunkt, Funktionalität und Kosten

Häufige praktische Probleme des Product Owners

- oft nicht bevollmächtigt, Entscheidungen verbindlich zu treffen
- Überlastung mit fremden Aufgaben

Scrum-Rollen

Rolle „Scrum Master“

- unterstützt das Team, aber steuert es möglichst nicht
- treibt den täglichen Prozess, fokussiert das Team auf Scrum
- überwacht den Fortschritt, führt “Sprint Burn-Down Chart”
- stellt optimale Arbeitsbedingungen sicher, beseitigt Hindernisse
- Coach, Moderator und Change Agent; fördert Lernprozesse
- moderiert Besprechungen, z.B. das *Daily Scrum*
- moderiert bei Konflikten
- unterstützt bei der Einführung von testgetriebener Entwicklung, Continuous Integration und Refactoring
- hält Sprint Planning am Beginn und Retrospektive am Ende jedes Sprints
- sollte keine Personalverantwortung für die Teammitglieder haben und keine Leistungsbeurteilungen zu den Teammitgliedern abgeben

Scrum-Rollen

Rolle „Entwicklungsteam“

- setzt User Stories in auslieferbare Produktinkremente um
- arbeitet eigenverantwortlich; organisiert sich weitgehend selbst
- verantwortlich für die Lieferung des Potentially Shippable Code
- reines Scrum erlaubt keine unterschiedlich benannten Rollen im Entwicklungsteam; per Definition sind alle Mitglieder Entwickler (!)
- besteht in der Praxis oft aus Menschen mit cross-funktionalen Fähigkeiten, zum Beispiel:
  - Architekt
  - UI-Designer
  - Entwickler
  - Tester
  - Dokumentierer
- etwa 5 bis 10 Mitglieder

Scrum-Rollen

Drei externe Rollen

- nicht formell eingebunden, aber wichtig für den Erfolg
- auch als “Stakeholder” bezeichnet = vom Projektergebnis betroffen
- Management:
  - verantwortet Rahmenbedingungen (materielle Ressourcen, Unterstützung für den eingeschlagenen Kurs)
  - schützt Scrum-Team vor externen Arbeitsanforderungen
  - unterstützt bei Beseitigung von Hindernissen

- Kunde:
  - Auftraggeber, ermöglicht Projekt und profitiert vom Ergebnis
  - sollte eng mit Product Owner zusammenarbeiten
- Nutzer:
  - kann Produkt aus Nutzersicht beurteilen
  - sollte an Sprint Planning Punkt 1 und Sprint Review teilnehmen

Scrum

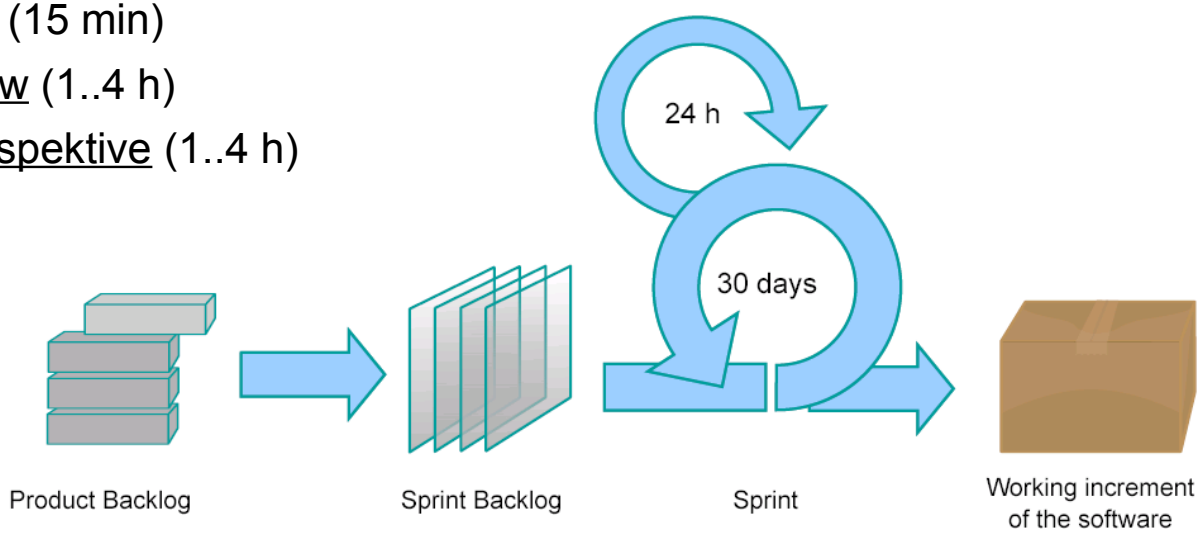
Timeboxing

- Timebox: fester Zeitrahmen für das Projekt und einzelne Aktivitäten
- Idee: jede Aktivität ist auf die festgelegte Dauer beschränkt, auch wenn nicht alle geplanten Inhalte abgeschlossen werden konnten
- noch offene Teile werden in eine folgende Timebox verschoben oder gestrichen
- Zweck: Erhöhung der Effizienz
- „timeboxed“ in Scrum sind alle planmäßigen Meetings, z.B. Daily Scrums, und die Sprints selbst

Scrum

Zyklen

- initiale Erstellung von Produktkonzept („Vision“) und Releaseplan
- kontinuierliche Pflege des Product Backlogs durch den Product Owner
- Sprints dauern 1..4 Wochen; je Sprint durchgeführt:
  - Sprint Planning => Sprint Backlog (2 x 1..4 h)
  - Daily Scrum (15 min)
  - Sprint Review (1..4 h)
  - Sprint-Retrospektive (1..4 h)



Scrum

Produktvision (vom „Scrum Guide“ inzwischen entfernt)

- vor Projektstart erstellt und regelmäßig angepasst
- kann beispielsweise enthalten:
  - Vision
  - Produktidee
  - Funktion
  - Produkteigenschaften
  - Zielgrößen
  - Marktforschungsergebnisse
  - Befragung von Kunden
  - Roadmap
- hilft bei der Kommunikation mit Stakeholdern und bei der Erstellung des Product Backlog

Scrum

Releaseplan (vom „Scrum Guide“ inzwischen entfernt)

- verantwortet vom Product Owner
- wird zu jedem Sprint aktualisiert
- Übersicht zum Zeit- und Kostenrahmen, zu Terminen für Zwischenergebnisse und Fertigstellung
- enthält grobe Reihenfolge der Umsetzung der Anforderungen und die erwartete Anzahl Sprints
- im Projektverlauf iterativ präzisiert
- dabei Risiken berücksichtigt

Scrum

Definition of Done („DoD“)

- gibt genau an, wann eine Anforderung/User Story als fertiggestellt gilt
- vor Projektbeginn definiert von Entwicklungsteam und Product Owner
- enthält auch alle notwendigen Testaktivitäten
- ist verbindlich

➔ Die Umsetzung einer User Story, die nicht alle Kriterien der DoD erfüllt, ist nicht fertig!

Scrum

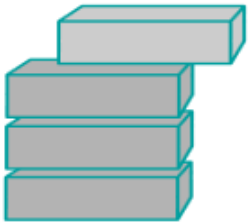
Beispiel für eine DoD

- Code ist fertiggestellt, committed und erfolgreich auf dem CI-System gebaut
- Unit-Tests und Integrationstests sind erstellt und erfolgreich ausgeführt
- Code ist dokumentiert
- Code Review im Team durchgeführt, Coding Guidelines eingehalten
- Release Notes sind erstellt
- QS-Maßnahmen sind beendet
- alle bekannten, noch nicht vollständig behandelten Fehler sind dokumentiert
- es gibt keine kritischen Fehler, die die User Story betreffen
- an der User Story dokumentierte Akzeptanzkriterien sind erfüllt
- Lizenzinformationen verwendeter Module sind dokumentiert
- der Product Owner hat die User Story akzeptiert

Scrum

Product Backlog

- ein zentrales Product Backlog für das Produkt
- Ablage für funktionale und nicht-funktionale Anforderungen
- zu Beginn grobgranular und unvollständig; regelmäßig präzisiert („Refinement“, füher „Grooming“)
- vom Product Owner gepflegt
- umfasst je Anforderung
  - Priorität
  - User Story / Beschreibung
  - Akzeptanzkriterien
  - Risiken
  - Schätzung für Entwicklungsaufwand (vom Team ermittelt)
- liefert den Input für die Sprint Backlogs



Scrum

User Stories

- spezifizieren Anforderungen; eine User Story je Anforderung
- in Alltagssprache bewusst kurz formuliert
- bestehen aus Anforderung und Akzeptanzkriterien
- oft auf Story Cards geschrieben (physisch oder virtuell)

Anforderung

- Typische Form:
  Als <Rolle> **möchte ich** <Wunsch>, **um** <Nutzen/Ziel>
- Beispiel:
  Als registrierter Nutzer **möchte ich** nach dem Login meinen Namen sehen, **um** eine Rückmeldung über das korrekte Login zu erhalten.

Akzeptanzkriterien

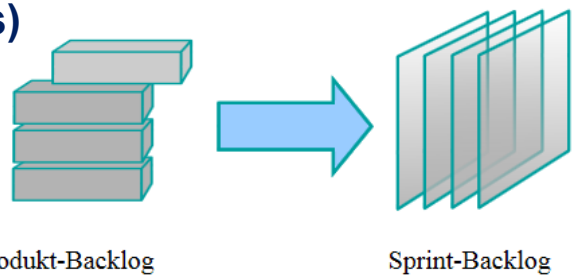
- Liste zu überprüfender Voraussetzungen für die Akzeptanz
- praktisch sind das oft die Anforderungsdetails

Scrum

Sprint Planning Meeting

- Kick-Off zu Beginn jedes Sprints
- Ziel: Erstellung des Sprint Backlog
- Timebox: 2..8 h (2h je Sprint-Woche)
- Input: Product Backlog
- Output: Sprint Backlog
- geteilt in zwei Punkte:

1. Auswahl von Anforderungen (Product Backlog Items)



2. Ermittlung der zur Umsetzung nötigen Aufgaben (Tasks)

Scrum

Sprint Planning Meeting, Punkt 1

- Was kann in diesem Sprint fertiggestellt werden?
- Wie: Verhandlung zwischen PO und Entwicklungsteam
  - PO erläutert Sprint-Ziel
  - Sprint-Ziel vom Team bestätigt
  - PO präsentiert höchstpriorisierte, ausgewählte Product Backlog Items
  - PO und Entwicklungsteam diskutieren diese und priorisieren eventuell neu
  - eventuell auch neue Product Backlog Items erstellt
  - Entwicklungsteam schätzt gemeinsam, welche Product Backlog Items voraussichtlich implementiert werden können (*Forecast*)
  - (Begriff „Commitment“ 2011 aus Scrum Guide entfernt)
- *Tester berücksichtigen den Testaufwand bei Schätzungen*

Scrum

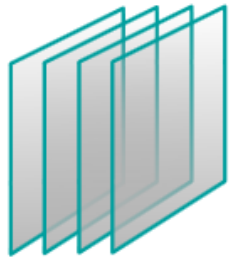
Sprint Planning Meeting, Punkt 2

- Wie wird die ausgewählte Arbeit erledigt?
- Entwicklungsteam (ohne Product Owner) ermittelt die für die Implementierung der Product Backlog Items nötigen Aufgaben (Tasks)
- die Tasks bilden das Sprint Backlog
- Granularität/Aufwand je Task etwa 1-8 h
- *Tester sorgen dafür, dass die Tasks auch Zeit für Unit Tests und andere Entwicklertests enthalten*
- *Tester schreiben separate Karten für Test Tasks ODER*
- *Tester fügen Test Tasks zu Entwicklungs-Task-Karten zu*
  - ➔ *Task ist nicht erledigt, wenn der Test nicht beendet ist*
  - ➔ *versucht, Mini-Wasserfall zu vermeiden*
  - ➔ *funktioniert nur, wenn keine entfernten Kollegen involviert sind*

Scrum

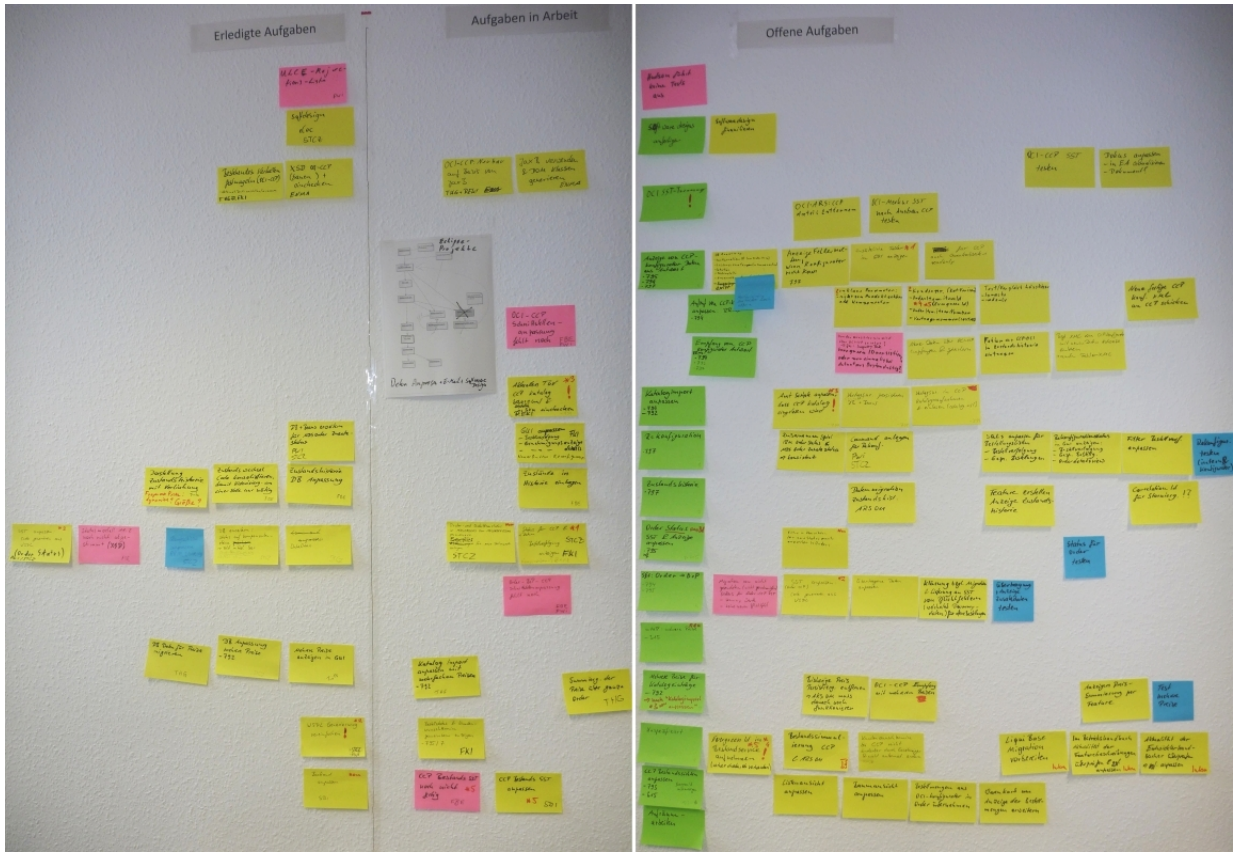
Sprint Backlog

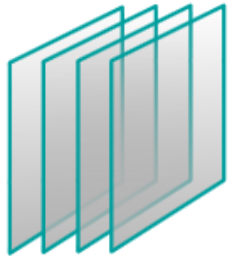
- Zweck: Verwaltung der Tasks im Sprint
- Ergebnis des Sprint-Planning-Meetings, Punkt 2
- umfasst im Detail
  - Sprint-Ziel
  - umzusetzende Anforderungen (Product Backlog Items)
  - dafür zu erledigende Tasks, sortiert nach
    - TO DO
    - IN PROGRESS
    - DONE
  - eventuell bereits Zuordnung zu Personen
  - Restaufwandsschätzung
- täglich aktualisiert
- Kärtchen an einer Wand (*Taskboard*), oft auch elektronisch



Scrum

Sprint Backlog, Beispiel:

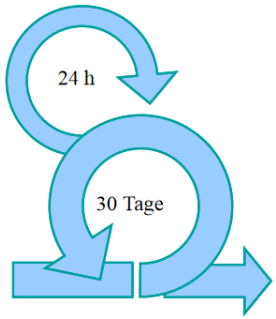




Scrum

Daily Scrum Meeting

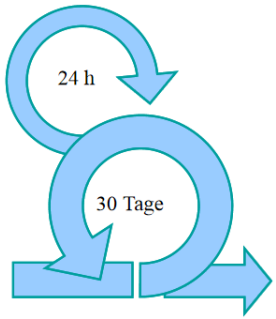
- täglich gleiche Uhrzeit und gleicher Ort
- oft als Stand-Up
- Timebox: 15 Minuten
- jedes Mitglied des Entwicklungsteams berichtet:
  - Was habe ich gestern erledigt, das dem Entwicklungsteam hilft, das Sprint-Ziel zu erreichen?
  - Was werde ich heute erledigen, um dem Entwicklungsteam bei der Erreichung des Sprint-Ziels zu helfen?
  - Sehe ich Hindernisse (*Impediments*), die mich oder das Entwicklungsteam vom Erreichen des Ziels abhalten?
- für andere Themen spezifische Meetings nach Bedarf planen; am besten gleich im Anschluss



Scrum

Daily Scrum Meeting (fortgesetzt)

- Ziele:
  - Synchronisation der Aktivitäten
  - Planung für die nächsten 24h
  - Erkennung und Beseitigung von Hindernissen
- Rederecht haben nur Entwicklungsteam und Scrum Master
- Product Owner darf zuhören
- *Tester nehmen als gleichberechtigte Mitglieder des Entwicklungsteams teil*



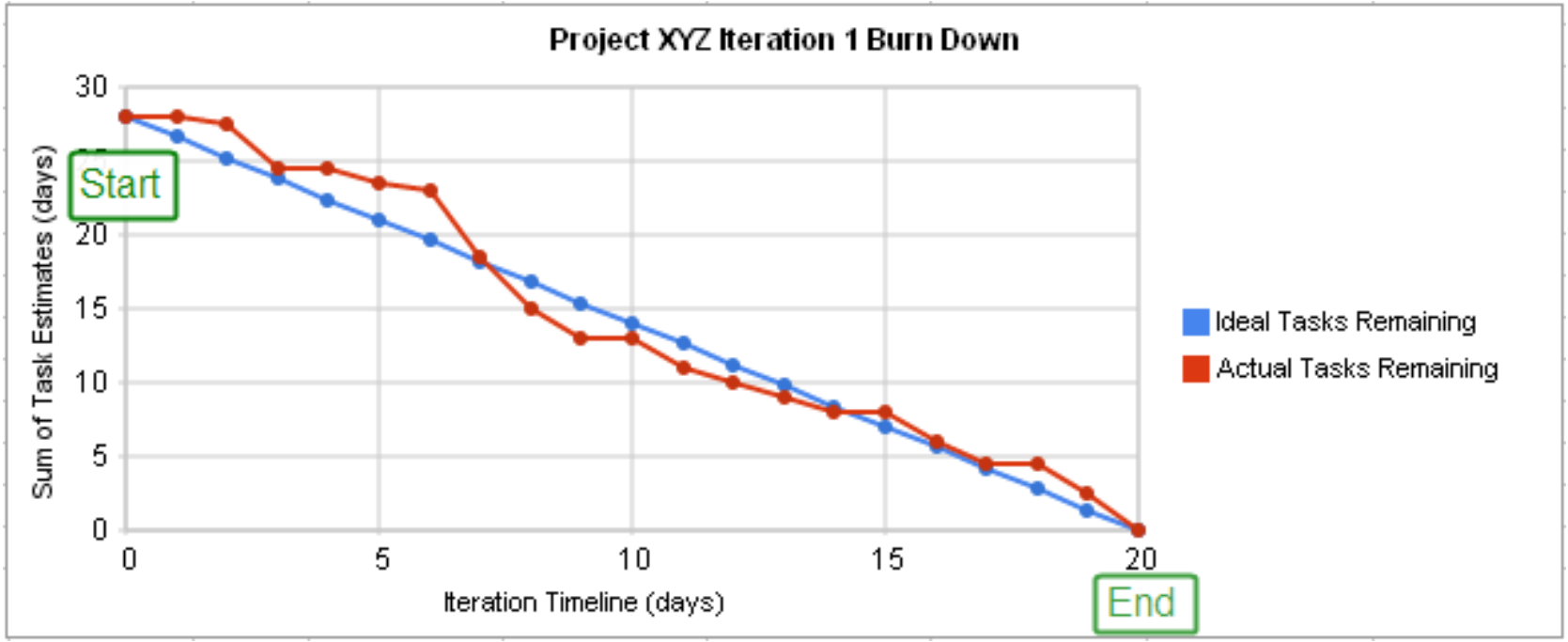
Scrum

Sprint Burn-Down Chart

- Ziel: Vergleich Soll - Ist
- grafische Darstellung des Restaufwands über die Sprint-Dauer
  - X-Achse: Tage im Sprint
  - Y-Achse: Restaufwand in h (nicht mehr empfohlen), Tasks oder Story Points
- zwei Linien, ideal und real
- ideal: fallende Linie, am Sprintende Restaufwand = 0
- real: täglich vom Scrum Master oder Entwicklungsteam aktualisiert
- auch als Burn-Up Chart mit Vorteilen bei Änderungen des Restaufwands
- (*Product/Release Burn-Down Chart* vom Scrum Guide 2011 entfernt)

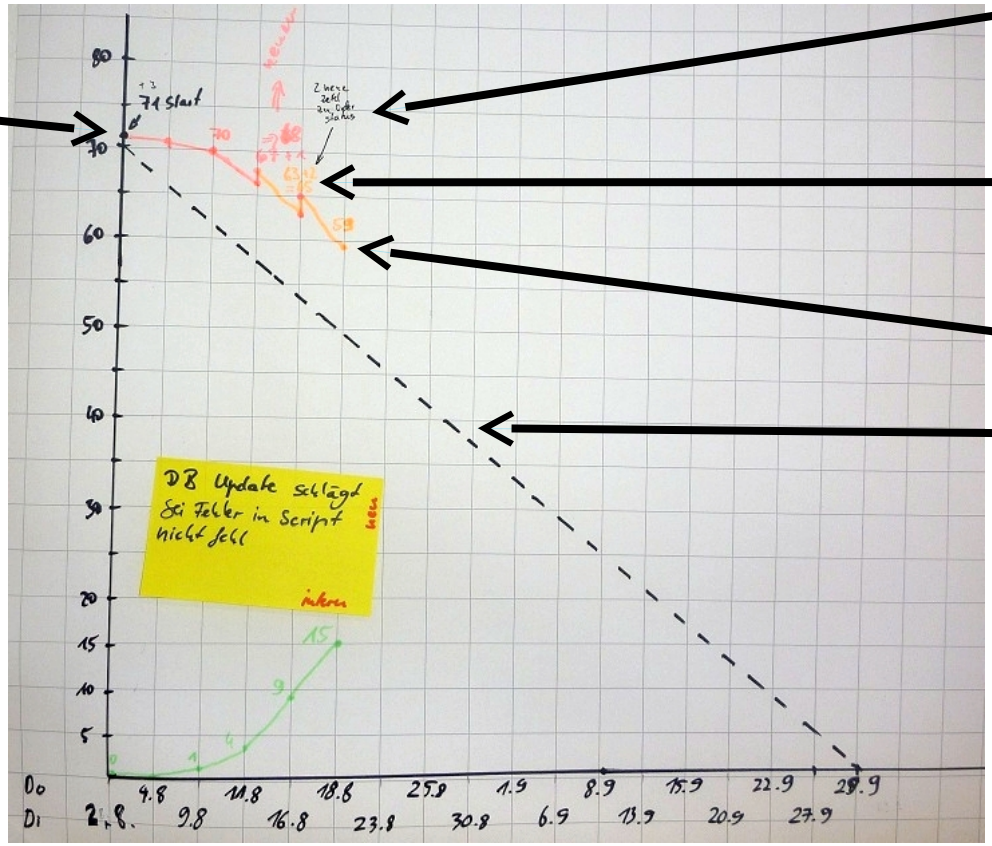
Scrum

Sprint Burn-Down Chart, fortgesetzt



Scrum

Sprint Burn-Down Chart, Beispiel:



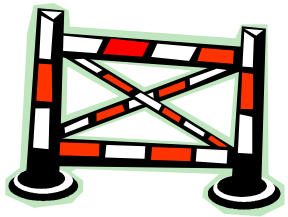
Scrum

Impediment Backlog (Hindernisliste)

- sammelt die in den Daily-Scrums kommunizierten Hindernisse
- vom Scrum Master verwaltet
- der Scrum Master versucht, die Hindernisse zu beseitigen
- Erwähnung im Sprint-Endbericht durch den Product Owner

Was ist ein Impediment (Hindernis, Blocker)?

- alles, was ein Mitglied des Entwicklungsteams daran hindert, seine Arbeit effizient auszuführen
- Beispiele:
  - fehlende Rechner, Zugänge oder Lizenzen
  - zu wenig Arbeitsspeicher
  - langsame Build-Maschine
  - mangelnder Input von anderen Teams oder Kundenvertretern



Scrum

Sprint Review

- Meeting am Ende jedes Sprints
- Timebox: max. 4 h bei 4 Wochen Sprintdauer
- Scrum Team und Stakeholder; ev. Support, Service, Sales, Marketing...
- Ziele:
  - Inspektion des Produktinkrements, Sammlung von Feedback
  - Anpassung des Product Backlogs
- Durchführung:
  - PO erklärt, was „Done“ ist und was nicht
  - Vorstellung des Sprint-Ergebnisses durch das Entwicklungsteam, oft als Demo
  - PO prüft und akzeptiert das Ergebnis aus Sicht der Anforderungen
- informell, keine Slides, maximal 2 h Vorbereitung

Scrum

Sprint-Retrospektive

- Meeting nach dem Sprint Review, vor dem nächsten Sprint Planning
- Timebox: max. 3 h bei 4 Wochen Sprintdauer
- Ziel: kontinuierliche Verbesserung des Entwicklungsprozesses
- Teilnehmer:
  - Entwicklungsteam, Scrum Master (moderiert)
  - oft ganz oder teilweise ohne Product Owner
- möglichst ehrlich, aber respektvoll, ohne persönliche Anschuldigungen
  - Was war gut?
  - Was war schlecht?
  - Was können wir verbessern?
- Ergebnis: einige wenige, konkrete Maßnahmen

Agenda

- Agile Vorgehensmodelle und Scrum
- Agile QS – Erster Ansatz
- Agile QS – Heute bewährt: Tester im Team
- QS in Scrum

Agile QS – erster Ansatz (1)

Gute allgemeine Grundsätze zur Verbesserung der Qualität…

- Verschiebung von Risiken und Unsicherheit vom Ende weiter nach vorn
- inkrementelle Schritte mit jeweils qualitätsgesicherten Ergebnissen
- Forderung nach konstruktiven QS-Maßnahmen („eingebaute Qualität“)
- frühestmöglicher Test, frühes und regelmäßiges Feedback
- frühestmögliche Einbeziehung der Stakeholder
- Förderung von Eigenverantwortung und Motivation des Teams
- Prinzipien schlanker Prozesse; „just enough, just in time“

Agile QS – erster Ansatz (2)

…aber in der Praxis oft zu schmaler Fokus hinsichtlich der QS-Maßnahmen; meist nur zwei Teststufen beschrieben:

1. Primärer Fokus auf automatisierten Unit-Tests (xUnit, Continuous Integration, Test Driven Development (TDD))

2. Akzeptanz von User Stories durch den Product Owner (und Kunden)

Agile QS – erster Ansatz (3)

Problem

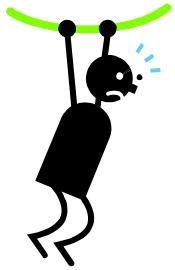
➔ Oft nur obige QS-Maßnahmen beschrieben (voriges Slide)!

➔ Alte, Entwickler-zentrierte Denkweise agiler Vorgehensmodelle:
„Tests sind notwendig, Tester jedoch nicht…“

➔ Die obigen Maßnahmen sind sinnvoll und wichtig, aber nicht ausreichend!

➔ Viele andere, aus traditionellen Vorgehensmodellen bekannte QS-Maßnahmen, wären auch hier wichtig!

➔ Hohes Risiko!



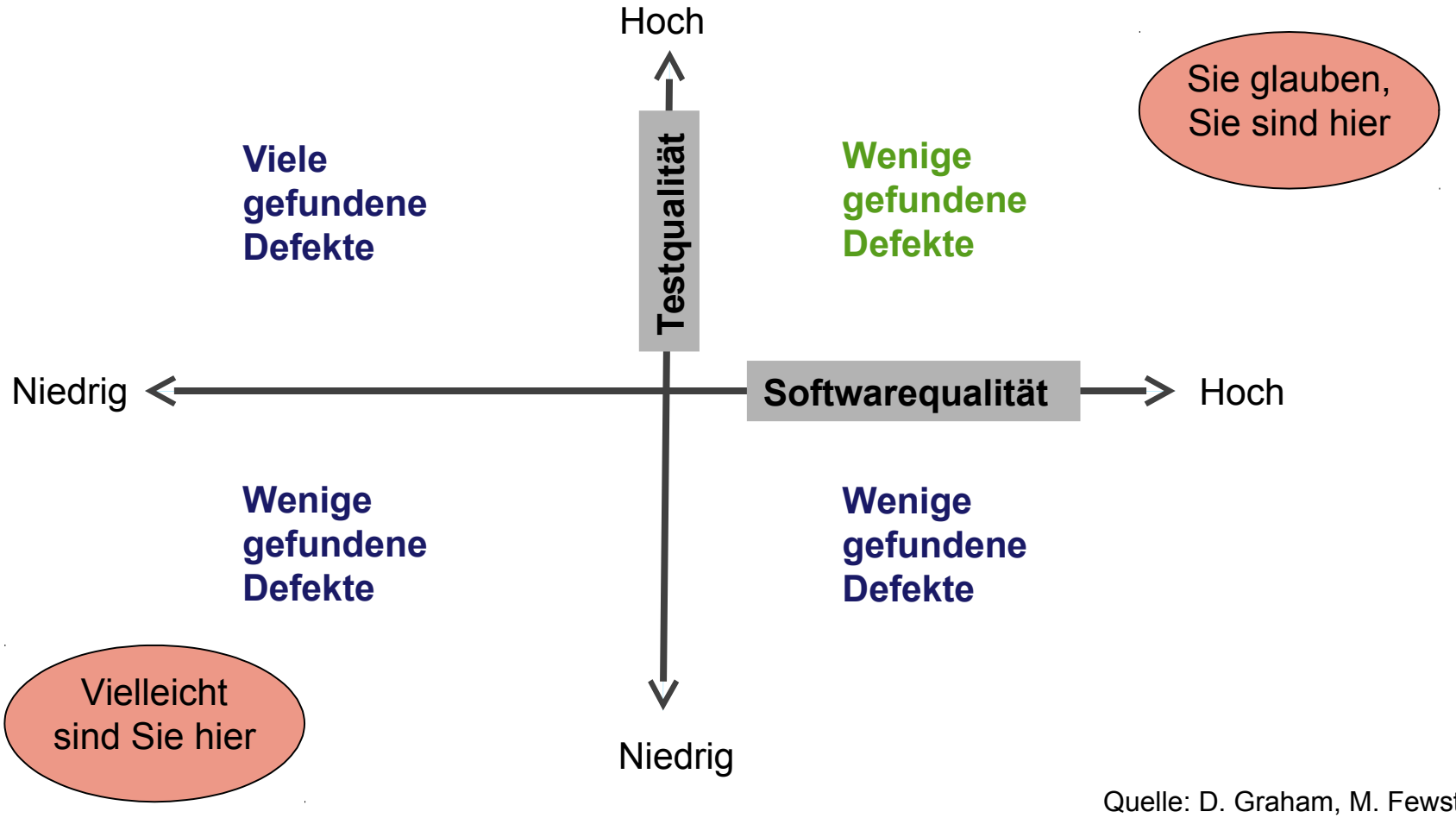
Agile QS – erster Ansatz (4)

“The Scrum Guide”, Ken Schwaber, Jeff Sutherland, 2017:

- “Scrum recognizes no titles for Development Team members, regardless of the work being performed by the person...
- ...Scrum recognizes no sub-teams in the Development Team, regardless of domains that need to be addressed like testing, architecture, operations, or business analysis; and,...
- ...Individual Development Team members may have specialized skills and areas of focus, but accountability belongs to the Development Team as a whole…
- ...Scrum’s roles, events, artifacts, and rules are immutable and although implementing only parts of Scrum is possible, the result is not Scrum.”

➔ Die Rolle “Tester” existiert in Scrum nicht (jedoch in XP).

Agile QS – erster Ansatz (5)



Quelle: D. Graham, M. Fewster

Was fehlt?

Integrationstests

umfassende Regressionstests

Tiefere Testabdeckung (Sonderfälle, …)

Wartbarkeit

Layout

Benutzerfreundlichkeit

Kompatibilität

Test auf bestimmte implizite Anforderungen

Testexpertise und Zeit bei Product Owner und Kunden

Lasttests

Übertragbarkeit

…

Ein definierter, systematischer Testprozess!

Warum werden dazu Tester benötigt?

Anspruch der eingebauten Qualität oft nicht haltbar

- Interessenkonflikt der Entwickler zwischen Entwicklungs- und QS-Aufgaben
- Unit-Tests am schnellsten und effizientesten durch Entwickler selbst geschrieben, aber bekannte Grenzen beim Test durch Entwickler selbst:
  - ➔ man begegnet eigenem Code zu optimistisch, kritischer Abstand fehlt
  - ➔ unbewusst möchte man die korrekte Funktion zeigen, keine Fehler finden
  - ➔ Blindheit gegenüber eigenen Fehlern
  - ➔ Entwickler haben ein anderes Mindset, als unabhängige Tester
  - ➔ Andere Testaktivitäten sind zusätzlich notwendig.

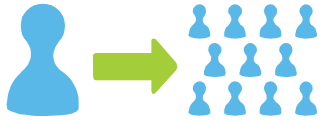
Jede Testautomatisierung hat Grenzen

Product Owner mit Interessenkonflikten und wenig Testexpertise

Agenda

- Agile Vorgehensmodelle und Scrum
- Agile QS – Erster Ansatz
- Agile QS – Heute bewährt: Tester im Team
- QS in Scrum

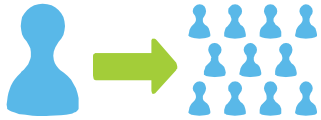
Tester im Entwicklungsteam (1)



Dedizierte Tester *sind* heute in der agilen Praxis Teil des Entwicklungsteams:

- Verhältnis Entwickler zu Testern je Entwicklungsteam etwa 3:1 bis 1:1
- Tester sind im Idealfall Spezialisten mit
  - Kenntnissen in analytischen QS-Maßnahmen und
  - Kenntnissen in konstruktiven QS-Maßnahmen
- Vorteile gegenüber den ersten agilen QS-Ansätzen
  - Verbesserung der QS-Maßnahmen durch Spezialwissen
  - keine Interessenkonflikte zwischen Entwicklungs- und Testaufgaben
  - keine reine Entwicklersicht
  - Tester stellen Fragen, vermitteln, bringen Menschen zusammen

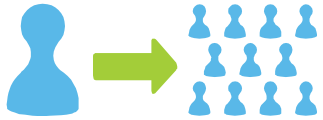
Tester im Entwicklungsteam (2)



Warum *müssen* dedizierte Tester in agilen Prozessen Teil des Entwicklungsteams sein?

- ➔ sonst nicht alle Tests Teil der Definition-of-Done
- ➔ gleichberechtigte und frühe Einbindung in alle Aktivitäten nötig
- ➔ Tester kommunizieren intensiv mit Entwicklern, technischen Autoren usw.
- ➔ nur so ist ein optimaler wechselseitiger Wissensaustausch möglich
- ➔ nur so können Engpässe rechtzeitig bemerkt und behoben werden
- ➔ nur so kann die gemeinsame Verantwortung von Entwicklern und Testern für die Qualität („Whole Team Approach“) umgesetzt werden

Whole Team Approach



Das gesamte Entwicklungsteam ist verantwortlich für Qualität und Qualitätssicherung, nicht nur die Tester!

- Entwickler sind nicht fertig, wenn sie Code an Tester übergeben
- bei offenen Testaufgaben gegen Ende einer Iteration bleiben alle länger im Büro, nicht nur die Tester
- Tester als „Qualitätsgewissen“ des Teams nehmen Einfluss auf Team-Entscheidungen und -Arbeitsweisen (konstruktive QS)
- Tester haben gleiche Rechte im multi-funktionalen Entwicklungsteam

Herausforderungen für agile Tester (1)

Neue Rolle der Tester im Vergeich zu klassischer QS

- ungewohnte, zunächst oft unklare Rolle und Aufgaben der Tester im Entwicklungsteam
- Verantwortlichkeit der Tester verschiebt sich
  - von einer nachgelagerten, unabhängigen Kontrollinstanz
  - zu einer Service-Rolle / zum Qualitätstreiber im Entwicklungsteam

  ➔ Kooperation statt Unabhängigkeit
- mehr Eigeninitiative nötig
- Entwickler sind eventuell skeptisch gegenüber Testern und umgekehrt
- kulturelle Unterschiede für frühere unabhängige QS-Teammitglieder schwer zu akzeptieren

➔ Die neue soziale Rolle ist oft die größte Herausforderung.

Herausforderungen für agile Tester (2)

Erweiterte technische Kenntnisse erforderlich

- Kommunikation auf Augenhöhe mit Entwicklern nötig
  - zur Erfüllung der Arbeitsaufgaben
  - zur Akzeptanz der Tester durch die Entwickler
- mehr Testautomatisierung erforderlich, z.B. für Regressionstests
- Beteiligung an konstruktiven QS-Maßnahmen
  - Erstellung und Prüfung von Coding Style Guides
  - Code Review
  - eventuell Hilfe bei Continuous Integration
  - …

Tipps für neue agile Tester (1)

Neue Rolle akzeptieren und leben!

- ➔ Ablegen der „Qualitätspolizei”-Denkweise bei Testern nötig
- ➔ „Talk First“ – immer erst miteinander reden!
- ➔ Machen Sie sich nützlich, unterstützen Sie die Entwickler!
- ➔ Liefern Sie Feedback!
- ➔ Gehen Sie aktiv auf die Entwickler zu, seien Sie nicht schüchtern!
- ➔ Fordern Sie benötigte Dinge ein!

Technische Skills ausbauen!

- ➔ Testautomatisierung
- ➔ Buildprozess
- ➔ Verständnis für Code…

Tipps für neue agile Tester (2)

Soft Skills ausbauen!

- ➔ Zusammenarbeit mit den Entwicklern noch wesentlich enger, als traditionell üblich
- ➔ Der Erfolg eines Testers hängt noch stärker von seiner Kommunikations- und Teamfähigkeit ab.

Beginn der Zusammenarbeit - Vorschlag

Beginn der Tester-Entwickler-Zusammenarbeit, Aufbau von Vertrauen:

1. Tester sprechen mit Entwicklern über jede Anforderung, bevor sie implementiert wird; beide einigen sich auf eine vereinbarte Lösung; (Entwickler lernen Tester als hilfreiche Diskussionspartner schätzen)
2. Tester erstellen Testfälle oder Testideen für jede Anforderung und bitten Entwickler um ein Review, bevor sie mit dem Test beginnen
3. Tester bitten Entwickler um Übergabe, wenn eine Anforderung testbereit ist:
   - Entwickler erläutert, was er implementiert oder geändert hat
   - Entwickler gibt Testhinweise; was zu beachten/was kritisch
   - Tester stellt Fragen zur Implementierung

➔ Am einfachsten beginnt man mit 3., später dann 2. und 1. hinzunehmen.

Zusätzlich zentrales QS-Team?

Mögliche Aufgaben eines zentralen QS-Teams:

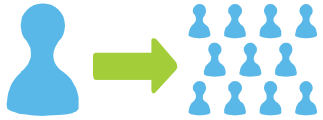
- Bereitstellung von zentralem QS-Know-How (Methodik, Tools, …)
- Betreuung zentraler QS-Infrastruktur (Werkzeuge, Testumgebungen)
- Erstellen und Verwalten von Testdaten
- fortlaufende Verbesserung der QS-Maßnahmen, Richtlinien und Style-Guides
- Testautomatisierung
  - Erstellung von Vorgaben
  - Pflege von Testframeworks, z.B. von „Page-Objekten“ bei Web-Tests
  - Erstellung automatisierter Testfälle
- Planung, Durchführung und Auswertung übergreifender Lasttests

➔ Bei mehreren agilen Entwicklungsteams ist ein zusätzliches zentrales QS-Team neben den Testern in den Entwicklungsteams oft sinnvoll.

Agenda

- Agile Vorgehensmodelle und Scrum
- Agile QS – Erster Ansatz
- Agile QS – Heute bewährt: Tester im Team
- QS in Scrum

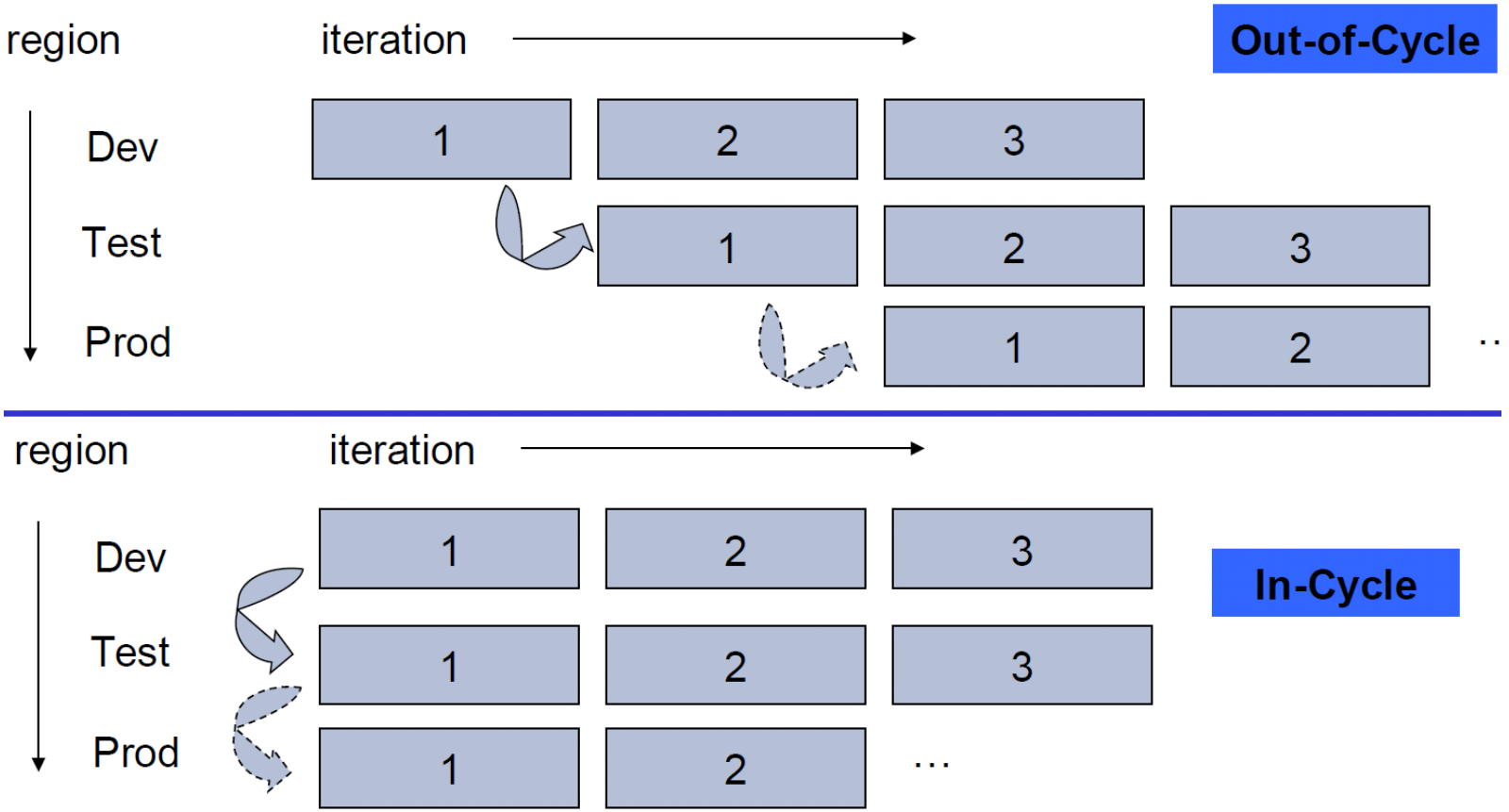
Tester im Entwicklungsteam



Gleichberechtigte Teilnahme an allen Aktivitäten

- Sprint Planning
  - ➔ Tester schätzen und berücksichtigen Testaufwand
  - ➔ Tester erstellen Test-Tasks
- Daily Scrum
  - ➔ Tester berichten
  - ➔ Tester nennen ihre Hindernisse
- Sprint Review
  - ➔ Teilnahme an Ergebnispräsentation, präsentieren eventuell selbst
  - ➔ liefern Feedback, ...
- Sprint Retrospektive
  - ➔ Tester schlagen Verbesserungen aus QS-Sicht vor

Test In-Cycle vs. Out-Of-Cycle (1)?



Quelle: Syed Rayhan, Code71, Inc.

Test In-Cycle vs. Out-Of-Cycle (2)?

Je Sprint eine potentiell auslieferbare Version erfordert Test In-Cycle!

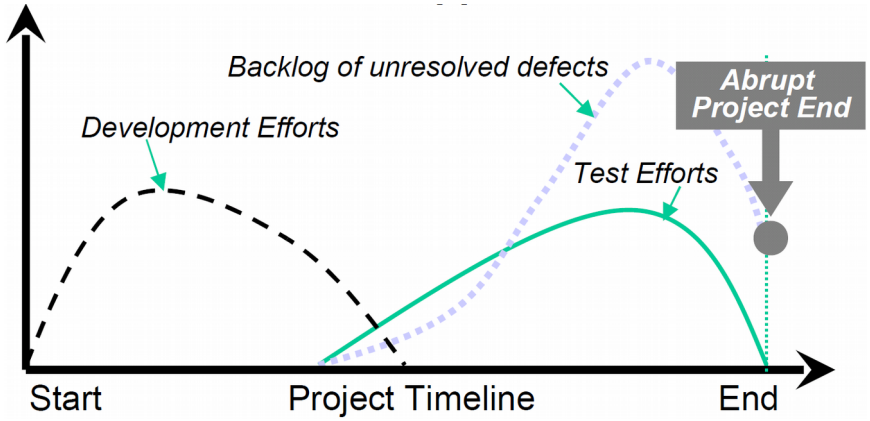
- eine Anforderung ist erst dann fertig umgesetzt, wenn alle zugehörigen Entwicklungs- und Testaufgaben abgeschlossen sind; siehe DoD
  - auch Integration mit Schnittstellenpartnern
  - auch Regressionstest, Lasttest, ...
  - auch Fehlerbehebung
- Keinen Aufwand in andere Sprints verschieben, 90%-Syndrom vermeiden!
- “Hardening Sprint”/”End Game” vor Release dennoch oft sinnvoll, auch wenn teilweise von Scrum-Verfechtern abgelehnt

Test In-Cycle vs. Out-Of-Cycle (3)?

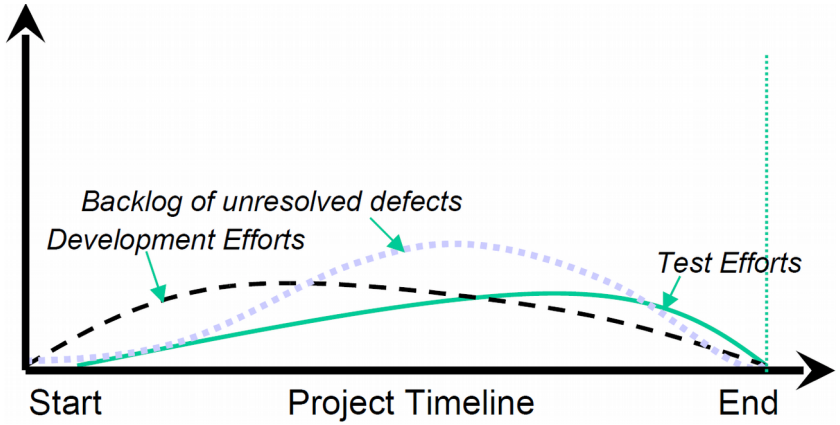
Je Sprint eine potentiell auslieferbare Version erfordert Test In-Cycle!

- ➔ Test innerhalb jedes Sprints ist kritisch für den Projekterfolg!
- ➔ Test nicht mehr nur auf fertigen Artefakten/Komponenten möglich
- ➔ Testaktivitäten nach vorn verlagern; frühestmögliche Vorbereitung
- ➔ Entwickler sind Testern im Fortschritt nicht voraus
- ➔ Ziel: Verlagerung der Schwerpunkte wie im nächsten Slide

Schwerpunkte im Releasezyklus



Wasserfallmodell (idealisiert)



Scrum (idealisiert)

Quelle: IBM Deutschland Research & Development GmbH, 2011

Agile QS – Zeitplanung

Traditionelle Zeitplanung nicht anwendbar

- keine dedizierte Testphase
- keine Zeit für detaillierte Testplanung
- keine zeitlich sequentiellen Teststufen
- potentiell auslieferbares Produkt am Ende jedes Sprints gefordert; häufige Änderungen; aber ohne dedizierten Zeitrahmen für Regressionstests
- Test darf nicht hinter Entwicklung zurückbleiben, damit Anforderungen “DONE”

➔ Testvorbereitung startet mit Beginn eines Sprints
➔ Erstellung logischer Testfälle vor Verfügbarkeit einer ausführbaren SW-Version
➔ frühestmöglicher Testbeginn, schnelles Feedback bei Verfügbarkeit einer testbaren Implementierung an Entwickler
➔ Test einzelner Entwicklungsschritte
➔ Automatisierung der Regressionstests

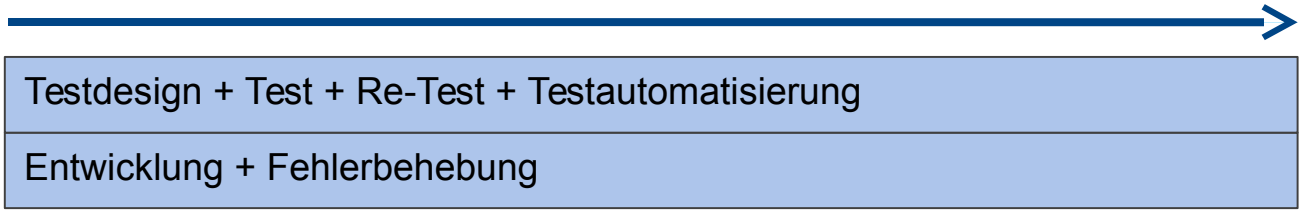
Mini-Wasserfall im Sprint vermeiden (1)

Gefahr eines Mini-Wasserfallmodells zwischen Entwicklung und Test:

Iteration

Parallele Durchführung von Entwicklung und Test anstreben:

Iteration



Mini-Wasserfall im Sprint vermeiden (2)

Auch daher gilt wieder:

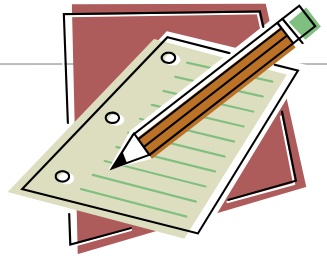
- Testaktivitäten möglichst früh im Sprint beginnen
- Test einzelner Entwicklungsschritte
- frühen Eindruck gewinnen, frühes Feedback an Entwickler
- sequentielle Übergabe implementierter Anforderungen; nicht alle gegen Ende
- spätesten Übergabetermin für implementierte Anforderungen im Sprint vereinbaren
- eventuell QS-Endgame innerhalb jedes Sprints einführen (letzte 1-2 Tage)

Mini-Wasserfall im Sprint vermeiden (3)

“Whole Team Approach” leben

- Gesamtes Team verantwortet Fertigstellung, das gilt auch für die QS!
- Separate Stufen “Development Done” versus “Testing Done” vermeiden!
- Solange die QS am Ende eines Sprints nicht fertig ist, arbeitet das gesamte Entwicklungsteam daran, nicht nur die Tester!

Agile QS – Soll-Vorgaben



Leichtgewichtige Soll-Vorgaben

- keine volle Anforderungsdokumentation und Spezifikation
- oft nicht detailliert genug für detaillierte Testfälle
- QS basiert auf weniger Dokumentation und mehr Kommunikation
- viele Änderungen, aber wenig Änderungsmanagement/-verfolgung

➔ Tester vor Fertigstellung der Anforderungen involviert
➔ Testfallerstellung schließt verstärkt Anforderungsdiskussion ein
➔ Tester klären Anforderungsdetails mit PO und Entwicklungsteam
➔ Tester erweitern Akzeptanzkriterien zusammen mit PO und Entwicklern
➔ Tester schreiben Testfälle, die die Anforderungen illustrieren, statt Testfälle aus vordefinierten Anforderungen zu erstellen
➔ Tester finden versteckte Annahmen, fragen “Warum?”, “Was, wenn...?”

Definition of Test (DoT)

Problem

- systematischer Testprozess in agilen Vorgehensmodellen nicht definiert, auch kaum in Artikeln und Büchern

Idee

- “Definition of Test (DoT)” als zentrales Dokument erstellen
- enthält alle Festlegungen zu QS-Maßnahmen
- Teststrategie, Regeln für Testinhalte, Häufigkeit, Ausführung von QS-Maßnahmen, Testfallentwurf, Werkzeugnutzung, …
- vergleichbar mit Rahmentestkonzept in traditionellen Vorgehensmodellen
- Regeln zur Integration der QS-Maßnahmen in den agilen Prozess
- wer erledigt welche QS-Aufgaben
- oft gegliedert nach Code-, Story-, Integrations- und Release-Ebene

Beispiel DoT (1) – Code-Ebene

Was, wer, wie?

- Code-Ebene (Klassen, Packages)
- kontinuierlich in jedem Sprint
- Erstellung von Unit Tests direkt nach Code-Erstellung oder davor (TDD)
- jeder Entwickler in seiner Entwicklungsumgebung
- Entwickler committen Code erst nach erfolgreichem lokalen Unit Test
- danach Ausführung automatisch in Continuous-Integration-Umgebung
- Code Review
- Fehlerbehebung sofort, keine Fehlerverwaltung

Inhalte

- Automatisierung mit xUnit
- Qualität auf Klassen- und Package-Ebene
- Einhaltung von Style Guides

Beispiel DoT (2) – Story-Ebene

Was, wer, wie?

- User-Story-Ebene, teilweise bis hinab auf Task-Ebene
- kontinuierlich für jede User Story
- Tester, Testautomatisierer, Product Owner (PO)
- Integrationsumgebung
- Fehlerverwaltung auf Task Board und in Bugtracker

Inhalte

- Verfeinerung Akzeptanzkriterien, Abstimmung mit PO
- Test auf Akzeptanzkriterien, weitere Qualitätsmerkmale, UAT
- Entwurf und Erstellung von Story-Testfällen, oft in einem Werkzeug
- Automatisierung von Testfällen oder Auswahl zur Automatisierung
- Ausführung automatisierter Regressionstestfälle der vorherigen Sprints regelmäßig tagsüber oder nachts auf separater Umgebung
- eventuelle Ausführung manueller Regressionstestfälle

Beispiel DoT (3) – Integrations-Ebene

Was, wer, wie?

- System-Integrations-Ebene (je Sprint nach Ende der Story-QS aller Teams)
- Tester, Testautomatisierer, Product Owner (PO), Nutzer in Sprint Review
- System-Integrationsumgebung, dort nur Stories mit erfolgter Story-QS
- Fehlerverwaltung im Bugtracker und eventuell auf Task Board
- Ausführung automatisierter Integrationstests mindestens täglich/nächtlich
- je Sprint; eventuell in einem QS-Endgame (1-2d)

Inhalte

- Erstellung teamspezifischer Integrationstestfälle in Testmanagementwerkzeug
- Erstellung von Integrationstestfällen zwischen Teams und zu Drittsystemen
- Automatisierung von Integrationstestfällen
- auch Ausführung der automatisierten Integrationstestfälle vorheriger Sprints
- manuelle End-Zu-End-Tests über alle Systeme, explorativer Test
- Test nichtfunktionaler Anforderungen, z.B. Lasttest

Beispiel DoT (4) – Release-Ebene

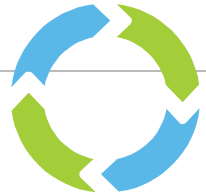
Was, wer, wie?

- Release-/Projekt-/Abnahme-Ebene
- teamübergreifend
- Tester aller Teams, Testautomatisierer, Testmanager, Product Owner
- je Release
- eventuell in einem speziellen Release-/Hardening-Sprint
- Fehlerverwaltung im Bugtracker und eventuell auf Task Board

Inhalte

- Abnahmetest auf Release-Ebene
- Ausführung sprint-spezifischer Testfälle
- Ausführung der automatisierten Regressionstestfälle vorheriger Sprints
- manuelle Regressionstests
- manuelle End-Zu-End-Test über alle Systeme, explorativer Test
- Test nichtfunktionaler Anforderungen, z.B. Lasttest

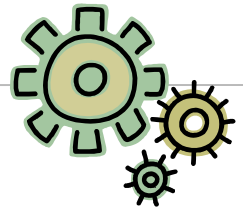
Agile QS – Mini-Testzyklus



Mini-Testzyklus, z.B. innerhalb von 24 h

- automatisierte Unit-Tests nach jeder Code-Änderung lokal bei Entwicklern
- automatisierte Builds nach jedem Commit in ein zentrales Repository mit anschließenden automatisierten Unit-Tests
- automatisierte Integrationstests mindestens einmal pro Nacht oder fortlaufend auf einem dedizierten Integrationssystem
- im Idealfall auch regelmäßige automatisierte Lasttests auf einer dedizierten Lasttestumgebung
- Live-Tests im Produktivsystem nach Deployments
- kontinuierliches End-Zu-End-Monitoring im Produktivsystem

Continuous Integration (1)



Was ist Continuous Integration?

- automatisiertes, fortlaufendes Zusammenfügen von Softwarekomponenten zu einer Anwendung
- anschließend automatisierte Ausführung bestimmter Tests
- ausgelöst durch Änderungen am Code

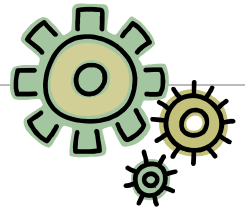
Vorteile

- laufende Erkennung
  - von Integrationsproblemen (Code passt nicht zusammen, …)
  - von Fehlern bei Ausführung des integrierten Builds
- Motivation der Entwickler zu mehr Disziplin, Qualität und kürzeren Commit-Intervallen durch sofortige Reaktion des Systems auf fehlerhaften Code

➔ Verbesserung der Softwarequalität

➔ ständige Verfügbarkeit eines lauffähigen, aktuellen Standes

Continuous Integration (2)



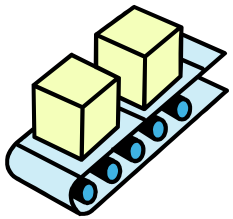
Ablauf

- regelmäßige Commits der Entwickler in zentrales Versionsverwaltungssystem
  - zu jeder Code-Änderung müssen Unit-Tests erstellt werden
  - vor dem Commit müssen die Unit-Tests erfolgreich auf der lokalen Entwicklungsumgebung des Entwicklers gelaufen sein
- automatische Ausführung des Buildprozesses nach Erkennung neuer Commits
- automatische Ausführung der Unit-Tests im integrierten Build
  - dabei oft automatische Ermittlung der Code-Überdeckung durch entsprechende Werkzeuge
  - oft auch automatische Code-”Vermessung” anhand bestimmter Metriken (Code Quality Management, CQM)
- maximal empfohlene Ausführungszeit: 10 min (oft schwer erreichbar)
- Ergebnis ist ein integrierter Softwarestand mit erfolgreichen Basistests
- bei Fehlern Rückmeldung an die Entwickler, z.B. per automatischer E-Mail

Continuous Delivery (1)

Was ist Continuous Delivery?

- Sammlung von Techniken, Prozessen und Werkzeugen zur Verbesserung des Software-Lieferungsprozesses auf die Entwicklungs-, Test-, Integrations- und sogar Produktivumgebungen
- schnelle und häufige Auslieferung neuer Softwareversionen in kleinen Schritten
- umfasst zuverlässigen, wiederholbaren, automatisierten Lieferprozesses
- Wenn etwas schwierig ist, mache es früh und oft ➔ Liefere oft aus!
- Das einzige, was wirklich “DONE” ist, ist live!



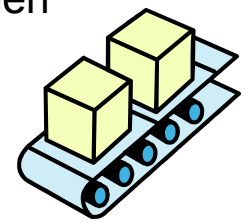
Eigenschaften

- Weiterentwicklung/Weiterführung der Continuous Integration
- nach erfolgreicher Continuous Integration automatische Fortsetzung mit automatisierten Tests höherer Ebenen (Integrationstests, eventuell Lasttests, …)
- anschließend meist auch Ausführung einiger manueller Tests
- Ergebnis ist eine getestete, automatisch auslieferbare Softwareversion

Continuous Delivery (2)

Eigenschaften/Vorteile

- Kleineres Risiko durch kleinere Änderungen!
  - ➔ weniger Abhängigkeiten zwischen unterschiedlichen Änderungen
  - ➔ Probleme schneller eingrenzbar
- schnelle Reaktion auf veränderte Anforderungen und bei Fixes möglich
- schnelles Feedback von realen Nutzern
- Entwicklung und Operations arbeiten enger mit Anforderern zusammen
- Operations ist nicht länger Engpass
- ermöglicht Ausrollen von Erweiterungen und Fehlerbehebungen mit minimalem Risiko und ohne großen manuellen Aufwand



- jede Version in der Versionsverwaltung zu jeder Zeit lieferbar
- letzter Auslieferungsschritt sollte dennoch immer manuell ausgelöst werden

Vielen Dank für Ihre Aufmerksamkeit!

Haben Sie Fragen?

Kontakt

Xceptance Software Technologies GmbH
Leutragraben 2-4
07743 Jena

Tel.: +49 (0) 3641 55944-0
Fax: +49 (0) 3641 376122

E-Mail: kontakt@xceptance.de
http://www.xceptance.de

Referenzen (1)

Bücher

- Lisa Crispin, Janet Gregory: „Agile Testing – A Practical Guide for Testers and Agile Teams“, Addison Wesley (2009)
- James A. Whittaker, Jason Arbon, Jeff Carollo: „How Google Tests Software“, Addison-Wesley (2012)
- Mike Cohn: „Succeeding with Agile: Software Development Using Scrum“, Addison-Wesley (2009)
- Kent Beck: „Test-Driven Development: By Example“, Addision Wesley (2003)
- Mary & Tom Poppendieck: „Lean Software Development – An Agile Toolkit“, Addison Wesley (2003)
- Whittaker, James, „Exploratory Software Testing“, Addison Wesley (2009)
- Andreas Spillner, Tilo Linz: „Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard“, dpunkt.verlag GmbH, 5. Auflage (2012)

Referenzen (2)

Websites zu Agilen Methoden & Scrum

- http://www.scrum.org
- http://www.agilemanifesto.org
- http://www.agilealliance.org
- http://www.controlchaos.com
- http://www.mountaingoatsoftware.com/scrum
- http://www.scrumalliance.org

Foren

- http://tech.groups.yahoo.com/group/agile-testing/
- http://tech.groups.yahoo.com/group/scrumdevelopment/

Blogs

- http://testobsessed.com/ (Elisabeth Hendrickson)
- http://www.kohl.ca/blog/ (Jonathan Kohl)

Referenzen (3)

Websites zum Softwaretest

- http://www.agiletester.ca/ (Lisa Crispin, Janet Gregory)
- http://www.istqb.org/downloads/glossary.html (ISTQB glossary)
- http://www.stickyminds.com
- http://www.satisfice.com (James Bach)
- http://kaner.com/?page_id=7 (Cem Kaner)
- http://testingeducation.org/wordpress/ (Cem Kaner)
- http://www.developsense.com (Michael Bolton)

Copyrights