**Qualitätssicherungskonzept**

**für die Softwareentwicklung**

**Teil 2 - Der Testprozess**

**Autor: Marion Grabenweger**

**12. November 2012**

**Version 0.3**

**Inhaltsverzeichnis**

[**1 Einleitung 1-1**](#_Toc340585487)

[**2 Der Testprozess 2-2**](#_Toc340585488)

[**2.1 Organisatorischer Testprozess 2-3**](#_Toc340585489)

[**2.2 Testmanagementprozess 2-3**](#_Toc340585490)

[**2.3 Statischer Testprozess 2-6**](#_Toc340585491)

[**2.4 Dynamischer Testprozess 2-6**](#_Toc340585492)

[**3 Testen im Softwarelebenszyklus 3-11**](#_Toc340585493)

[**3.1 Wasserfallmodell 3-11**](#_Toc340585494)

[**3.2 Klassisches V-Modell 3-11**](#_Toc340585495)

[**3.3 XP (Extreme Programming) 3-13**](#_Toc340585496)

[**3.4 Scrum 3-14**](#_Toc340585497)

[**3.5 Kanban 3-18**](#_Toc340585498)

# Einleitung

Das Qualitätssicherungskonzept soll die Grundsätze der Qualitätssicherungsmaßnahmen der Softwareentwicklung regeln.

Dieses Dokument gliedert sich in 4 Teile:

Teil 1 „Grundlagen des Softwaretestens“ beschreibt die Grundlagen des Softwaretestens und inkludiert Begriffsbestimmungen, um ein gemeinsames „Wording“ zu ermöglichen.

Im 2. Teil wird ein generischer Testprozess beschrieben. Durch die Gliederung in Teilprozesse soll sichergestellt werden, dass das Testvorgehen für alle Projekte anwendbar ist, wobei Teile des Prozesses von anderen Bereichen übernommen werden können.

Der 3. Teil beschäftigt sich mit der Dokumentation, die während der jeweiligen Testteilprozesse erstellt werden kann und liefert Templates und Beispiele.

Im 4. Teil werden die Testverfahren und Testtechniken erklärt, die zum Einsatz kommen können um eine systematische Testfallableitung zu gewährleisten.

Alle 4 Teile haben zum Ziel einen systematischen und strukturierten Testprozess einzuführen um die Qualität der zu liefernden Softwareprodukte zu steigern.

# Der Testprozess

Testen ist ein integraler Bestandteil unterschiedlicher Softwareentwicklungsmodelle. Die Testaktivitäten müssen daher an das ausgewählte Softwareentwicklungsmodell angepasst werden.

Der Testprozess besteht aus Testphasen, die – jeweils nach Erfordernis – für unterschiedliche Ebenen angewendet werden können. Nämlich für das Gesamtprojekt, für jede Teststufe und auch je Testobjekt und Testfall.

Obwohl in den Abbildungen die einzelnen Phasen in sequentieller Reihenfolge angegeben sind, können sie sich überschneiden und teilweise auch gleichzeitig durchgeführt werden.

Die Testaktivitäten werden in 4 unterschiedliche Prozessgruppen aufgeteilt:

* **Organisatorischer Testprozess**
* **Testmanagementprozesse**
* **Statische Testprozesse**
* **Dynamische Testprozesse**



Abb. 4‑1 Testprozesse

## Organisatorischer Testprozess

Der **Organisatorische Testprozess** regelt die Erstellung und Verwaltung von organisatorischen Testspezifikationen. Diese Spezifikationen regeln die unternehmensweite Testpolitik und sind nicht projektbezogen.

## Testmanagementprozess

Der **Testmanagementprozess** sollte ans jeweilige Projekt angepasst werden, um unterschiedlichen Bedürfnissen in verschiedenen Testphasen und Testtypen gerecht zu werden.

Er gliedert sich in 3 Teilprozesse:

* **Testplanung**
* **Testüberwachung & Steuerung**
* **Testabschluss**
	+ 1. Testplanung (*Wer? Testmanager aus IT-Org?*)

Während der **Testplanungsphase** wird der Testplan entwickelt. Abhängig von der Projektphase kann der Testplan ein Projekttestplan, oder ein Testplan für unterschiedliche Phasen des Projekts sein (z.B.: Systemtestplan, Perfomancetestplan).

Der Input für den **Testplanungsprozess** kann folgende Informationen einschließen:

* die organisatorische Testpolitik
* die organisatorische Teststrategie
* Standards
* den Projekttestplan, wenn unterschiedliche Testphasen und Testtypen geplant werden
* Fehlerreports
* den Projektmanagementplan
* geeignete Produktdokumentationen (wie z.B.: Systemanforderungen, Testobjektspezifikationen)
* den Softwareentwicklungsplan
* Projekt und Produktrisiken
* Testplan Updates



Abb. 4‑2 Teilprozess Testplanung

Mit der Planung des Testprozesses wird am Anfang des Softwareentwicklungsprojekts begonnen. Während des Projektfortschritts sind regelmäßig Kontrollen der bisherigen Planung und gegebenenfalls Aktualisierungen und Anpassungen vorzunehmen.

Folgende Aufgaben fallen in die **Testplanung**sphase:

* Bestimmen des Umfangs und der Risiken des Testens
* Identifikation der Testziele
* Auswahl der zu testenden Softwarekomponenten
* Definition einer projektbezogenen Testvorgehensweise
* Bestimmen und Einplanen benötigter Ressourcen
* Definition der Testorganisation
* Zeitplanung über alle Phasen von Planung bist Bericht
* Definition von Testmetriken zur projektbegleitenden Bewertung des Testfortschrittes
* Bestimmen der Testendekriterien
* Festlegen von Testmethoden und Teststufen unter Berücksichtigung von funktionalen und nichtfunktionalen Merkmalen des Testobjekts

Die in der Testplanungsphase gewonnenen Erkenntnisse werden im **Testplan** festgehalten

Während des Testprozesses muss der Testplan immer wieder neu angepasst werden, sobald neue Informationen verfügbar sind.

* + 1. Testüberwachung & Steuerung (*Wer? Testmanager aus IT-Org?)*

Der **Testüberwachungs- und Steuerungsprozess** gewährleistet die Einhaltung des Testplans, sowie der unternehmenspolitischen Teststrategien. Dieser Prozess begleitet das gesamte Projekt.



Abb. 4‑3 Testüberwachungs- und Steuerungsprozess

Aufgaben der **Testüberwachung & -steuerung:**

* Messen, Analysieren, Überwachen und Dokumentieren von
* Testfortschritt (gegen den Testplan)
* Risiko- und Testüberdeckung
* Fehlerinformationen (Fehleranzahlen, Fehlerdichte, Fehlerklassen)
* Testendekriterien
* Planabweichungen
* Anstoß von Korrekturmaßnahmen
	+ 1. Testabschluss

Die abschließenden Testaktivitäten lassen sich in vier Hauptgruppen einteilen

* Sicherstellen, dass alle Testaufgaben tatsächlich abgeschlossen sind
* Vergleich geplanter/durchgeführter/ausgelassener Tests
* Abschluss von Vorfallsberichten und Abweichungsmeldungen
* alle erfassten Fehler sind entweder behoben und nachgetestet, auf später verschoben oder akzeptiert
* Lieferung aller geplanten Arbeitsergebnisse an Personen/Stellen, die diese benötigen
* Liste bekannter Fehler an Support oder Endbenutzer
* Testmittel an Folgeprojekte
* Rückblickende Besprechung mit Analyse aller Testprozessphasen („Lessons Learned“) zur Verbesserung des Test- sowie auch des Entwicklungsprozesses
* Archivierung aller Arbeitsergebnisse

## Statischer Testprozess

Statische Softwaretestverfahren gehören zu den analysierenden Verfahren beim Softwaretest und unterteilen sich in

* **Reviewprozesse**
* **Statische Analyseprozesse**

Statische Testverfahren zeichnen sich dadurch aus, dass die Software bei diesen Tests nicht ausgeführt wird, im Gegensatz zu den dynamischen Testverfahren.

* + 1. Reviewprozesse

Bei Reviews nutzt man die menschlichen Denk- und Analysefähigkeiten, um durch Lesen und Nachvollziehen das Testobjekt zu prüfen.

Es gibt 5 Typen von systematischen oder formalen **Software-Reviews:**

* **Technisches Review**
* **Informelles Review**
* **Walkthrough**
* **Inspektion**
* **Audit**

Diese Reviewarten können prinzipiell auf alle Arbeitsergebnisse im Softwareentwicklungsprozess angewendet werden und sind ein effizientes Mittel, um frühzeitig Fehler und Unstimmigkeiten festzustellen. Die Beseitigung der Fehler und Unstimmigkeiten führt zu einer verbesserten Qualität der untersuchten Dokumente und wirkt sich positiv auf den gesamten Entwicklungsprozess aus.

* + 1. Statische Analyseprozesse

Die statische Analyse hat das Ziel, Fehler im Programmcode oder in formal beschriebenen Softwaremodellen zu finden. Die statische Analyse wird mit entsprechender Werkzeugunterstützung durchgeführt. Einsatzgebiete von statischen Analysewerkzeugen sind

* die Überprüfung gegen Programmierrichtlinien (Compiler)
* Datenflussanalyse
* Kontrollflussanalyse
* Erstellung von **Metriken**

## Dynamischer Testprozess

Der dynamische Testprozess beschreibt wie unterschiedliche Phasen, oder Arten des dynamischen Testens durchgeführt werden sollen.

Es gibt 4 dynamische Testteilprozesse:

* **Testdesign & Implementierung**
* **Bereitstellung und Wartung der Testumgebung**
* **Testdurchführung**
* **Abweichungsmanagement**



Abb. 4‑4 Dynamischer Testprozess

* + 1. Testdesign & Testimplementierung

In der Phase des **Testdesigns** und der **Implementierung** werden die Details des Testprojekts betrachtet:

* Review/Analyse der Testbasis
* Beurteilen der Testbarkeit von Anforderungen und Testobjekten
* Identifikation und Priorisierung der Testziele auf Grundlage der Testobjektanalyse
* Entwurf/Spezifikation logischer Testfälle mit Hilfe der in der Teststrategie festgelegten Testverfahren
* Priorisierung der Testfälle
* Sicherstellung, dass die Testfälle in jedem Fall reproduzierbar, nachprüfbar und bis zu den Anforderungen rückverfolgbar sind

****

 Abb. 4‑5 Testentwurfs- & Implementierungsprozess

Dieser Prozess kann jederzeit verlassen und wieder aufgenommen werden, wenn beispielsweise nach der Testdurchführung weitere Testfälle erforderlich sind um die Testendekriterien zu erreichen, oder ein Teil der Testfälle erst im Zuge einer dieser Prozessphasen abgeleitet werden kann.

Folgende aufeinanderfolgende Aufgaben werden unter Anwendung diverser Testtechniken in dieser Prozessphase erledigt:

* **Leistungsmerkmale identifizieren**
* Die Testbasis wird analysiert um die Anforderungen zu verstehen.
* Die zu testenden Anforderungen werden wenn möglich in Gruppen von Leistungsmerkmalen zusammengefasst. Eine Gruppe kann unabhängig von einer anderen getestet werden. Für den Komponententest wird möglicherweise nur eine Gruppe von Leistungsmerkmalen erforderlich sein, während in höheren Teststufen (z.B.: Systemtest) eine größere Menge an Gruppen erforderlich sein kann, die typischerweise die Architektur des Testobjektes wiederspiegeln.
* Die Leistungsmerkmal-Gruppen werden nach Risiken priorisiert.
* Gegebenenfalls werden die Leistungsmerkmale in der Testentwurfsspezifikation dokumentiert.
* **Testbedingungen ableiten**
* Basierend auf die im Testplan spezifizierten Testendekriterien, werden für jedes Leistungsmerkmal die Testbedingungen ermittelt (zu testendes Element oder Ereignis).
* Falls erforderlich sind die Testbedingungen in der Testentwurfsspezifikation zu dokumentieren.
* **Testschritte ableiten**
* Zur Erfüllung der Testbedingungen werden die Testschritte ermittelt. Testschritte sind Attribute, die zur Ausführung eines Testfalls beschrieben werden müssen. Konkrete Testdaten sind zu diesem Zeitpunkt noch nicht erforderlich.
* Falls erforderlich sind die Testschritte in der Testfallspezifikation zu dokumentieren.
* **Testfälle ableiten**
* die logischen Testfälle werden zu konkreten Testfällen gebildet
* spätestens hier werden die Sollwerte/-reaktionen festgelegt
* die Testdaten werden bereitgestellt
* **Testsuiten zusammenstellen**
* Testfälle, die die selben Ausführungsbedingungen haben, können in Testsuiten zusammengefasst werden
* die Testsuiten sollen in der Testablaufspezifikation erfasst werden
* **Testabläufe ableiten**
* Testfälle und Testschritte werden in eine ausführbare Reihenfolge gebracht (Testszenario)
* Testinput- und Testumgebungsdaten, die in Datenbanken oder anderen Repositories abgelegt werden, werden erstellt
* Testskripte und sonstige Datengeneratoren werden erstellt, die Daten erzeugen, die dann während der Testdurchführung als eingehende Last an das System gesandt werden
* die Ergebnisse in dieser Phase werden in der Testablaufspezifikation dokumentiert
	+ 1. Bereitstellung und Wartung der Testumgebung

Sobald der Testdesign- und Testimplementierungsprozess gestartet hat, lassen sich auch die Anforderungen an die Testumgebung erkennen.



Abb. 4‑6 Bereitstellung/Wartung der Testumgebung

Vor der eigentlichen Testdurchführung muss die Testumgebung klar definiert und aufgebaut sein. Das können sein:

* Betriebssystem
* weitere benötigte Applikationen/Fremdsysteme
* Testwerkzeuge (Testtreiber, Analysetools, Testmanagementwerkzeuge, etc.)
* Testschnittstellen
* Testobjekt

Die Testumgebung soll möglichst ungleich der Entwicklungsumgebung und möglichst nahe der späteren Systemumgebung sein, wobei der Einfluss verwendeter Testtools beachtet werden muss.

Der Aufbau der Testumgebung schließt auch das Konfigurations- & Fehlermanagement, sowie das Testmanagement mit ein.

* + 1. Testdurchführung

Der Testdurchführungsprozess beschreibt, welche Ergebnisse die Ausführung der Testszenarien in der Testumgebung bringen. Der Testdurchführungsprozess wird solange ausgeführt und wiederholt, bis die Testendekriterien erreicht sind.



Abb. 4‑7 Testdurchführungsprozess

Die Teilprozesse der **Testdurchführung**:

#### Testszenarien ausführen

* Ein oder mehrere Testszenarien werden in der Testumgebung ausgeführt. Die Testszenarien können automatisiert mit Hilfe eines Testskripts, manuell nach den Vorgaben in einer Ablauftestspezifikation, oder können beim explorativen Testen unmittelbar ausgeführt werden.
* Das Testergebnis soll beobachtet und dokumentiert werden.

#### Testergebnisse vergleichen

* Die aktuellen und erwarteten Ergebnisse werden für jeden Testfall verglichen.
* Das Testergebnis wird ermittelt. Jede Abweichung, jedes Ereignis, welches während des Testens auftritt und weiterer Untersuchung bedarf, soll dem Abweichungsmanagement gemeldet werden. Gleichzeitig soll die Entwicklung bei der Fehlerursachenanalyse unterstützt werden.
* Wenn ein Fehler behoben wurde, muss der Testdurchführungsprozess neuerlich von Anfang an durchgeführt werden und die Fehlermeldung im Abweichungsmanagementprozess aktualisiert werden.

#### Testdurchführung aufzeichnen

* Die Testdurchführung sollte im Testprotokoll dokumentiert werden.
* Bei automatisierten Tests wird das Ergebnis im Testlog protokolliert.
	+ 1. Abweichungsmanagement

Der Abweichungsmanagementprozess regelt die Erfassung von Fehlern und Abweichungen. Dieser Prozess wird angestoßen, sobald ein Fehlverhalten entdeckt wird, ein unvorhersehbares Ereignis auftritt, oder ein Fehler behoben wurde.

****

Abb. 4‑8 Abweichungsmanagementprozess

#### Testergebnisse analysieren

* Behobene Fehler bewirken eine Aktualisierung der Fehlermeldung.
* Bei neuen Fehlern müssen die Soll-/Ist-Abweichungen analysiert werden um festzustellen, ob es sich um einen Fehlerzustand im Testobjekt, um ungenaue oder fehlerhafte Testspezifikationen/-skripte, oder auch eine fehlerhafte Testinfrastruktur handelt.

#### Fehlermeldung erstellen/aktualisieren

* Wenn erforderlich muss eine Fehlermeldung erstellt, oder aktualisiert werden.

Merkmale einer guten Fehlermeldung sind:

* Der Testablauf ist knapp, aber nachvollziehbar beschrieben.
* Der aufgetretene Fehler ist klar umrissen, reproduzierbar und lokalisierbar.
* Die Testumgebung, in der der Test durchgeführt wurde, ist eindeutig definiert.
* Die verwendeten Testdaten sind dokumentiert.
* Es ist genau ein Fehler beschrieben.

# Testen im Softwarelebenszyklus

Um eine strukturierte und steuerbare Softwareentwicklung durchzuführen, werden **Softwareentwicklungsmodelle** bzw. **Entwicklungsprozesse** eingesetzt. Es gibt eine ganze Reihe unterschiedlicher Modelle. Alle Modelle definieren eine Systematik zur geordneten Projektabwicklung. Meist werden Phasen oder Arbeitsschritte festgelegt, die mit einem bestimmten Ergebnis abzuschließen sind. Ein Phasenabschluß, oder auch Meilenstein ist dann erreicht, wenn die geforderten Ergebnisse fertiggestellt sind und den Qualitätskriterien genügen. Meist werden Rollen beschrieben, denen bestimmte anfallende Aufgaben in der Softwareentwicklung zugeordnet und die von Projektmitarbeitern auszufüllen sind. Teilweise werden bei den Modellen auch die einzusetzenden Methoden und Verfahren in den jeweiligen Phasen beschrieben. Mithilfe der Modelle kann eine für alle Beteiligten gemeinsame und verbindliche Sicht der auszuführenden Tätigkeiten und deren zeitliche Abfolge erreicht werden.

Das Testen findet sich in jedem dieser Vorgehensmodelle wieder, allerdings mit sehr unterschiedlicher Bedeutung und Umfang. Es folgen einige Modelle als Beispiel, wie sich das Testen in den Softwareentwicklungszyklus integrieren lässt, bzw. bereits integriert ist.

## Wasserfallmodell

Das klassische Wasserfallmodell erlaubt Iterationen nur zwischen zwei aufeinanderfolgenden Phasen und durchläuft relativ starr sequentiell folgende Phasen:



Abb. 5‑1 Wasserfallmodell

Der entscheidende Nachteil ist, dass das Testen als „einmalige“ Aktion am Projektende vor der Inbetriebnahme vorgesehen ist. Das bedeutet, dass nur noch Entwicklungsfehler behoben werden können. Fehler die bereits in den vorgehenden Phasen gemacht wurden, können zu teuren Änderungen führen (mehrmals wiederholtes Durchlaufen des Prozesses bei Änderungen).

## Klassisches V-Modell

Das klassische V-Modell erklärt auf klare und präzise Weise, wie und wann das Testen in den Softwareentwicklungsprozess integriert ist.

Die linke Seite des Modells zeigt die Phasen, in denen das System aufgebaut oder erweitert wird. Auf der rechten Seite des Modells befinden sich die Teststufen.



Abb. 5‑2 V-Modell

* **Projektinitialisierungsphase**

Ausgangspunkt für die Entwicklung von Softwaresystemen ist die Projektinitialisierung. In ihr werden die Wünsche und Anforderungen an das spätere System dokumentiert. Zweck, Leistungsmerkmale und Qualitätsanforderungen müssen aus den Anforderungen hervorgehen. Die Projektdefinition ist Grundlage für die Spezifikatio der Abnahmetestfälle. Die Anforderungen müssen so präzise formuliert sein, dass sich Testfälle ableiten lassen.

* **Anforderungsanalysephase**

Beim funktionalen Systementwurf, der Anforderungsanalyse werden auf Grundlage der Anforderungen die Funktionen und Dialogabläufe spezifiziert. Die dabei entstehende Dokumentation wird von den Testern zur Erstellung der funktionalen Systemtestfälle und zum Test der Benutzeroberfläche benötigt.

* **Anwendungsentwurfsphase**

Im technischen Systementwurf, dem Anwendungsentwurf werden die Schnittstellen zur Systemumgebung definiert und der Zerlegung des Systems in überschaubare Teilsysteme sowie deren Schnittstellen und Zusammenspiel vorgenommen. Beim Integrationstest werden diese Informationen verwendet um die Systemumgebung zu simulieren und um die Schnittstellen zwischen den Teilsystemen ausreichend testen zu können.

* **Anwendungskomponenten**

Die Spezifikation der Komponenten beinhaltet das Verhalten und den inneren Aufbau bzw. Ablauf innerhalb der Komponente. Diese Informationen werden sowohl für die Programmierung als auch für den Test der Komponenten benötigt.

## XP (Extreme Programming)

XP beschreibt eine agile iterative Vorgehensweise beim Softwareentwicklungsprozess mit „Test-First-Ansatz“. XP ist ein strukturiertes Vorgehensmodell und stellt die Teamarbeit und stete Kommunikation zwischen allen Beteiligten in den Vordergrund. Das Testen hat eine hervorgehobene Bedeutung. Es wird von Projektbeginn an berücksichtigt und steht im Mittelpunkt der Überlegungen. Die Methode geht davon aus, dass der Auftraggeber die Anforderungen an die zu erstellende Software zu Projektbeginn noch nicht komplett kennt und strukturieren kann und sich im Laufe eines Projektes die Prioritäten ändern können und zu Beginn geforderte Funktionen möglicherweise im Laufe der Zeit in einer anderen Form benötigt werden. Neue Funktionalität wird permanent entwickelt, integriert und getestet. Dem Auftraggeber bietet XP, gerade durch seine kurzen Entwicklungszyklen, jederzeit die Möglichkeit steuernd auf das Projekt einzuwirken.



* **Aufbauorganisation**

Neben dem Entwicklungsteam gibt es im Wesentlichen den Auftraggeber. Dieser kann auch der Produktbesitzer sein. Der Produktbesitzer entscheidet über die genaue Vorgehensweise. Innerhalb des Entwicklungsteams sollte es keine Rollentrennung geben.

* **Anforderungsmanagement**

Der Umgang mit den Anforderungen und deren Verwirklichung ist eine zentrale Komponente. Auf eine vollständige Erhebung aller Anforderungen zu Beginn des Projektes wird verzichtet. Stattdessen werden die sich erst im Laufe der Umsetzung ergebenden Anforderungen berücksichtigt. Ständiger Gedankenaustausch mit dem Auftraggeber, Offenheit für Änderungen und stetige Integration wirken den Risiken, Fehler erst spät zu entdecken, oder ein Produkt nicht zur Zufriedenheit des Auftraggebers zu entwickeln, entgegen. Anforderungen werden nicht selten zunächst als Prototypen bereitgestellt. Dabei handelt es sich um Versionen, die noch nicht die volle, endgültige Funktionalität besitzen.

* **Planung**

Ein Sprint beinhaltet Funktionen, die insgesamt und für sich geschlossen die Bereitstellung einer neuen Version rechtfertigen. Der Releaseplan besteht aus Iterationen. Iterationen dauern zwischen einer und vier Wochen.

* **User-Storys**

Die innerhalb der Iterationen umzusetzenden Neuerungen werden mit User-Stories beschrieben (einer schlankeren Form der Use Cases).Das ganze Team ist bei der Erstellung beteiligt. Die Anforderungen werden auf Story Cards geschrieben und für alle sichtbar platziert. Die User Stories werden mit Hilfe von Storypoints nach Risiko bezüglich Zeitplan, Kosten oder Funktionalität und dem Nutzen priorisiert. Die Testfälle werden vor der Programmierung eines jeden Bausteins oder Inkrements erstellt bzw. programmiert (z.B.: mit JUnit). Die Testfälle dienen als Spezifikation der geforderten Funktionalität der einzelnen Stories. Durch die Automatisierung der Testdurchführung können neue oder geänderte Programmteile sehr schnell überprüft werden.

Eine gute User Story erfüllt die sogenannten INVEST-Kriterien. INVEST steht dabei für:

* **Indipendent:** Eine Story ist unabhängig von anderen Stories. Eine Story darf nicht davon abhängen, dass zuerst eine andere Story umgesetzt werden muss.
* **Negotiable:** User Stories sind kein geschriebenes Gesetz. Kunden und Entwickler besprechen und präzisieren sie gemeinsam. Während der Kunde also die Funktionalität kurz und grob beschreibt, werden die Details mit den Entwicklern zusammen ausgearbeitet.
* **Valuable:** Die Stories sollten einen erkennbaren Mehrwert liefern. Ansonsten gibt es keinen Grund, sie zu realisieren. Der beste Weg zu wertvollen Stories ist der, sie vom Nutzer selbst schreiben zu lassen.
* **Estimatable:** Eine Story muss so überschaubar sein, dass die Entwickler die Umsetzung der Anforderung einschätzen können. Zudem müssen die Entwickler natürlich über die entsprechenden fachlichen und technischen Kompetenzen verfügen.
* **Small:** Über den konkreten Umfang von User Stories muss letztlich das Team entscheiden. Stories können „zu groß“ und „zu klein“ sein. Als grobe Regel gilt: Die komplette Umsetzung einer Story soll mindestens einen halben Personentag und maximal zehn Personentage erfordern.
* **Testable:** Die Tests bilden den Maßstab dafür, ob eine Story erfolgreich abgeschlossen wurde oder nicht. Daher muss die Testbarkeit zwingend gewährleistet sein.
* **Aufwandsabschätzung**

Der Aufwand wird in Storypoints abgeschätzt. Dies sind relative Aufwandsabschätzungen, also der Entwicklungsaufwand für eine Story im Vergleich zu anderen.

* **Entwicklung und Abschluss**

User-Storys werden zu Beginn der Iteration in technische Arbeitspakete (Tasks) zerlegt. Jeder Entwickler nimmt sich jeweils ein Arbeitspaket vor. Die Entwickler arbeiten in Paaren, wobei die Möglichkeit besteht, dass der eine Entwickler die Unittests schreibt, während der andere implementiert. Beim nächsten Task können die Rollen getauscht werden (Ping-Pong-System). Die Implementierung einer User-Story ist erst abgeschlossen, wenn alle einzelnen Tasks dieser User-Story abgearbeitet, die Tests geschrieben und alle erfolgreich durchgelaufen sind.

Der Auftraggeber ist dann für den Akzeptanztest bei jedem Release verantwortlich. Ein nicht bestandener Akzeptanztest wird als Zeichen dafür gesehen, dass zusätzliche Unittests benötigt werden.

## Scrum

Scrum ist ein Vorgehensmodell, das versucht die Prinzipien der agilen Softwareentwicklung in die Praxis umzusetzen. Die einzelnen Prozesse bestehen aus ineinander verschachtelten Schleifen, die dem Feedback dienen und jeweils aus den Phasen Planung, Durchführung, Überprüfung und Anpassung bestehen.

Der Scrum-Prozess besteht aus sechs Rollen, sechs Meetings und neun Artefakten. Rollen sind dabei in erster Linie das Team, sowie die Rollen Product Owner und Scrum Master. Das Team organisiert sich selbst.

****

Abb. 5‑3 Scrum Cycle

* **Rollen**
* **Product Owner**

Der Product Owner ist verantwortlich dafür, dass das Team die gewünschten Funktionalitäten in der richtigen Reihenfolge erstellen und arbeitet kontinuierlich am Product Backlog und der Releaseplanung.

* **ScrumMaster**

Der ScrumMaster hilft dem Team, die Ziele zu erreichen, indem er daran arbeitet, dass Blockaden gelöst werden, die Mitarbeiter zu schulen und dass der Scrum-Prozess eingehalten wird.

* **Team**

Das Entwicklerteam besteht aus 5-10 Personen und ist idealerweise interdisziplinär zusammengesetzt (Entwickler, Architekten, Tester). Ein Tester im SCrum-Team ist ein vollwertiges Mitglied. Er schätzt den Testaufwand bei der Sprint-Planung und nimmt am Daily Scrum teil.

* **Manager**

Das Management schafft den Rahmen, in dem sich das Team, der Product Owner und der ScrumMaster bewegen. Es stellt die Ressourcen und Richtlinien zur Verfügung.

* **Costumer**

Der Kunde ist Auftraggeber des Projekts.

* **User**

Der Anwender ist eine wesentliche Informationsquelle für das Scrum-Team. Er definiert beim Sprint Planning gemeinsam mit dem Product Owner die Anforderungen.

* **Meetings**
* **Sprint Planning Meeting 1 - Anforderungen klären**

Der Product Owner erläutert die Product Backlog Items und definiert gemeinsam mit den Teammitgliedern das Ziel für den anstehenden Sprint. Dann werden mit Hilfe von Storypoints die Backlog Items ausgewählt, die zu diesem Ziel passen und die das Team schaffen kann. So entsteht das Selected Product Backlog. Wichtig: Das Team alleine bestimmt, wie viele Backlog Items es auswählt.

* **Sprint Planning Meeting 2 - Design und Planung**

Im zweiten Sprintplanungsmeeting wird die Erreichung des Zieles im Detail geplant: welche Architektur, welche Interfaces, Erstellung der Testfälle, usw. Am Ende des Meetings liegt eine Liste aller notwendigen Aufgaben vor, das Sprint Backlog.

* **Daily Scrum - Koordinieren und Feedback**

Jeden Tag treffen sich die Teammitglieder zur gleichen Zeit am selben Ort für 15 Minuten zu einem vom ScrumMaster moderierten Meeting. In diesem Meeting beantworten die Teammitglieder sich gegenseitig jeweils 3 Fragen:

* Welche Aktivitäten habe ich seit der letzten Daily-Scrum-Besprechung abgeschlossen?
* Woran plane ich bis zur nächsten Daily Scrum zu arbeiten?
* Werde ich in irgendeiner Form an der Ausführung einer Aktivität behindert?

Die Teammitglieder wählen selbst die Aufgabe aus, die sie als Nächstes übernehmen wollen. Die Teammitglieder informieren den ScrumMaster über Blockaden und Probleme, damit dieser sie so schnell wie möglich zur Lösung bringen kann.

* **Estimation Meeting - Vorausplanen und Schätzen**

Product Owner und Teammitglieder aktualisieren mindestens einmal im Sprint das Product Backlog. Dieses Meeting dient auch dazu, den Releaseplan zu aktualisieren.

* **Sprint Review - Resultate Präsentieren**

Am Ende des Sprints präsentiert das Team die erarbeiteten Funktionalitäten. Nicht getestete oder instabile Funktionalitäten werden nicht gezeigt und gelten als nicht geliefert.

* **Sprint-Retrospektive - sich ständig verbessern**

Die Sprint-Retrospektive ermöglicht das systematische Lernen des Teams. Hier wird analysiert, welche Arbeitsprozesse verbessert werden müssen. Die Resultate werden im Impediment Backlog festgehalten und lassen sich so als Verbesserungsvorschläge in das Sprint Planning einbringen.

* **Artefakte**
* **Vision**

Eine Vision definiert für alle Projektbeteiligten den Zweck, den Kontext und die Ziele des zu entwickelnden Systems.

* **Product Backlog Item**

In Scrum wird die zu liefernde Funktionalität als Backlog Item bezeichnet.

* **Product Backlog**

Das Product Backlog ist eine Liste von Product Backlog Items, die vom Product Owner nach ihrer Bedeutung für den Projekterfolg priorisiert werden.

* **Sprint Goal**

Der Product Owner legt gemeinsam mit dem Team das Ziel des Sprints fest.

* **Selected Product Backlog**

Das Selected Product Backlog ist das Ergebnis des ersten Sprint Planning Meetings. Es enthält die Backlog Items, die das Team am Ende des Sprints liefern kann.

* **Die Aufgaben/Tasks**

Mit Aufgaben, wird alles bezeichnet, was getan werden muss, um das Ziel des Sprints zu erreichen.

* **Sprint Backlog**

Im Sprint Planning Meeting 2 entsteht die Liste der Aufgaben, die das Team abarbeiten muss, um das Sprint Goal zu erreichen. Das Team nützt das Sprint Backlog dazu, um alle anstehenden Aufgaben zu überblicken und um sich zu orientieren, wer gerade was tut und was noch getan werden muss.

* **Impediment Backlog**

Hier werden alle Blockaden gesammelt, die einem Team aus dem Weg geräumt werden müssen, damit das Team produktiver werden kann.

* **Product Increment**

Das Product Increment ist das vorzeigbare Ergebnis des Sprints in Form von auslieferbarer Software.

* **Planung (Plan)**

Ein Scrum Projekt beginnt mit der Planung. Die Anforderungsliste (User Storys), der sogenannte Product Backlog, definiert den groben Aufwand für das Team, woraus sich der zeitliche Aufwand und die Verantwortlichkeiten für jeden Einzelnen ableiten lassen. Ein Sprint umfasst maximal einen Zeitraum von 2-4 Wochen.

* **Level of Done**

Das entscheidende Prinzip ist: Am Ende eines Sprints muss das Entwicklungsteam potenziell nutzbare Funktionalität liefern. Diese Vorgabe muss an die jeweiligen Entwicklungsbedingungen angepasst werden. Deshalb wird zwischen Product Owner und dem Entwicklungsteam der Level of Done vereinbart. Der ScrumMaster arbeitet mit dem Scrum-Team daran, den Level of Done kontinuierlich zu erhöhen. Im Idealfall wird am Ende des Sprints an den End-User ausgeliefert.



Abb. 5‑4 Beispiel: Scrum Taskboard

* **Durchführung (Do)**

Der Sprint dient allen dazu, die gesteckten Ziele zu erreichen. User Stories mit der höchsten Priorisierung werden zuerst bearbeitet. Alle arbeiten gemeinsam an jeweils einer User Story. Das Team macht sich erst dann an die Bearbeitung der nächsten Funktionalität, wenn alle zuvor vereinbarten Akzeptanz- und Testkriterien erfüllt und die Funktionalität damit tatsächlich fertig ist.

* **Überprüfung (Check)**

Im Sprint Review wird anhand der Ergebnisse überprüft, ob die Ziele (Level of Done) erreicht wurden.

* **Anpassung (Act)**

In der Sprint Retrospektive werden Verbesserungsnotwendigkeiten identifiziert, so dass sie in der nächsten Planung berücksichtigt werden können.



Abb. 5‑5 Scrumprozess

* **Testprozesse in Scrum**

Testen ist ein integrierter Bestandteil jedes agilen Entwicklungsprojektes. Die Tests basieren auf den User Stories. Ein **Anforderungsreview** ist daher zweckmäßig. Jede User Story sollte daher nicht nur „As a user, I can… such that…“ sondern auch „Verify that …“ (**Acceptance criteria**) beinhalten. Die Verwendung von Acceptance Criteria ergibt die Möglichkeit ausführbare Tests abzuleiten, die parallel zur eigentlichen Implementierung entwickelt werden und ein guter Weg sind, um die Abnahmekriterien der User Story zu bestimmen.

Ein weiterer wichtiger Punkt ist ein automationsunterstütztes Risikomanagement der **Regressionstests**, da jeder Sprint-Build auf den vorhergehenden aufsetzt und sichergestellt werden muss, dass Änderungen keine unerwünschten Seiteneffekte auf unmodifizierte Teile der Software haben.

 Auch auf die Änderungen der Anforderungen und deren Einfluss auf die Testartefakte sollte toolunterstützt eingegangen werden können.

Der **Systemtest** überprüft alle funktionalen und nichtfunktionalen Anforderungen, sowohl der User Stories, als auch aller High-Level Anforderungen, die existieren. Darauf folgt dann der **Akzeptanztest**, der sicherstellen soll, dass die Funktionalität auch den Bedürfnissen der Anwender entspricht.

## Kanban

Kanban ist ein Vorgehen, das bei der Softwareentwicklung die Anzahl paralleler Arbeiten, den **Work in Progress (WiP)**, reduziert und somit schnellere Durchlaufzeiten erreicht und Probleme – insbesondere Engpässe – schnell sichtbar macht.

Der erste Schritt bei der Einführung von Kanban besteht darin, den bestehenden Workflow, die vorhandene Arbeit sowie Probleme zu visualisieren. Dies wird in Form eines Kanban-Boards getan, das. aus einem einfachen Whiteboard und Haftnotizen oder Karteikarten bestehen kann. Jede Karte auf dem Board repräsentiert dabei eine Aufgabe.

Kanban basiert auf einem ‚**Pull-System**‘. Das bedeutet, dass ein neuer Task genommen und nicht gegeben wird, und zwar nur dann, wenn die Kapazitäten im Team vorhanden sind.

Die Phasen der Entwicklung werden am **Kanban-Board** mit vertikalen Spalten dargestellt. Stories/Tasks starten links und durchlaufen alle Phasen bis sie den Kriterien von ‚DONE‘ entsprechen. Für jede Phase ist ein oberes Limit (**WiP**) für die Anzahl der Tasks definiert, die sich gleichzeitig in dieser Phase befinden dürfen. Sobald ein Task eine Phase verlässt, entsteht Platz für einen neuen. Die an einer Phase beteiligten Personen entscheiden eigenständig, welcher Task diesen Platz einnimmt. Störungen im Datenfluss werden damit rasch erkennbar.



* **Die Kerneigenschaften von Kanban:**
* **Visualisiere den Fluss der Arbeit:** Die Prozessschritte (z.B.: Anforderungsdefinition, Programmierung, Dokumentation, Test, Inbetriebnahme) werden für alle Beteiligten gut sichtbar auf einem **Kanban-Board** visualisiert. Die einzelnen Anforderungen (es können Tasks, Features, User Storys, Minimal Marketable Features (MMF) usw. sein) werden auf Karteikarten oder Haftnotizen festgehalten und durchwandern mit der Zeit als so genannte Tickets das Kanban-Board von links nach rechts.
* **Begrenze die Menge angefangener Arbeit:** Die Anzahl der Tickets (**Work in Progress – WiP**), die gleichzeitig an einer Station bearbeitet werden dürfen, wird limitiert. Wenn beispielsweise die Programmierung gerade zwei Tickets bearbeitet, und das Limit für diese Station zwei beträgt, darf sie kein drittes Ticket annehmen, auch wenn die Anforderungsdefinition ein weiteres bereitstellen könnte. Hierdurch entsteht ein Pull-System, bei dem sich jede Station ihre Arbeit bei der Vorgängerstation abholt, anstatt fertige Arbeit einfach an die nächste Station zu übergeben.
* **Miss und steuere den Fluss:** Die Mitglieder eines Kanban-Prozesses messen typische Größen wie Längen von Warteschlangen, Zykluszeit und Durchsatz, um festzustellen, wie gut die Arbeit organisiert ist, oder wo man noch etwas verbessern kann. Dadurch wird die Planung erleichtert und die Produktivität gesteigert.
* **Mache die Regeln für den Prozess explizit:** Um sicherzustellen, dass alle Beteiligten des Prozesses wissen, unter welchen Annahmen und Gesetzmäßigkeiten man arbeitet, werden möglichst alle Regeln, die es gibt, explizit gemacht. Dazu gehören z.B. eine Definition des Begriffes "fertig" (**Definition of Done - DoD** ), Bedeutung der einzelnen Spalten, Antworten auf die Fragen: wer zieht, wann zieht man, wie wählt man das nächste zu ziehende Ticket aus der Menge der vorhandenen Tickets aus, usw.
* **Verwende Modelle, um Chancen für kollaborative Verbesserungen zu erkennen:** Modelle sind Vereinfachungen über den Prozess. Ein beliebtes Modell ist z.B. das von Wert, Fluss und Verschwendung aus der "Lean IT". Andere Modelle basieren auf den Ideen von Deming oder auf der Engpasstheorie, auf systemischem Denken oder auf der Komplexitätstheorie. Modelle können dabei helfen, ein besseres Prozessverständnis zu erreichen und Experimente zu finden, die zu einer Verbesserung des Prozesses führen.
* Die Visualisierung und die Begrenzung des WiP sind einfache Mittel, mit denen rasch sichtbar wird, wie schnell die Tickets die verschiedenen Stationen durchlaufen und wo sich Tickets stauen. Die Stellen, vor denen sich Tickets häufen, während an den nachfolgenden Stationen freie Kapazitäten vorhanden sind, werden als **Bottlenecks**bezeichnet. Durch Analysen des Kanban-Boards können immer wieder Maßnahmen ergriffen werden, um einen möglichst gleichmäßigen Fluss (**Flow**) zu erreichen. Beispielsweise können die Limits für einzelne Stationen verändert werden, es können Puffer eingeführt werden (insbesondere vor Bottlenecks, die durch nur zeitweise Verfügbarkeit von Ressourcen entstehen), die Anzahl der Mitarbeiter an den verschiedenen Stationen kann verändert werden, technische Probleme werden beseitigt usw. Dieser kontinuierliche Verbesserungsprozess (japanisch**: Kaizen**) ist wesentlicher Bestandteil von Kanban.
* **Priorisierung:** Um zu entscheiden, welche Storys/Tasks/Features zu welchem Zeitpunkt in das Kanban-System gegeben werden, wird nach dem Prinzip entschieden, welchen Verlust der Anwender hat, wenn ihm die Funktionalität erst zu einem späteren Zeitpunkt zur Verfügung steht.
* **Service Level Agreements (SLA):** Prinzipiell werden die Tickets nach dem FIFO-Vorgehen (**First in, First out**) behandelt. Um jedoch unterschiedliche Wichtigkeit der Tickets darzustellen, können Service-Arten definiert werden:

**Beschleunigt (Expedite)**

 Diese Tickets müssen mit hoher Priorität behandelt werden. Je nach Domäne kann es nötig sein, dass das gesamte Team seine momentane Tätigkeit stoppt, um dieses Ticket zu bearbeiten.

**Fester Termin (Fixed Date)**

 Wenn eine Funktionalität erst zu einem fixen Termin benötigt wird (zum Beispiel weil dann eine Gesetzesänderung wirksam wird), dann werden die entsprechenden Tickets so durch das Kanban-System geschleust, dass die Funktionalität kurz vor diesem Stichtag produktiv geht.

**Vage (Intangible)**

 Wenn der Geschäftswert und/oder die Verzögerungskosten für eine neue Funktionalität vage sind, werden die entsprechenden Tickets nachrangig behandelt. Das Team kann zum Beispiel definieren, dass sich zu jedem Zeitpunkt nur ein solches Ticket im System befinden darf.

**Standard (Standard)**

 Alle anderen Tickets zählen zur Standard-Serviceklasse. In der Regel werden Sie nach FIFO behandelt. Das Team kann jedoch auch andere/zusätzliche Regeln definieren.

* **Gemeinsamkeiten von Kanban und Scrum:**
* schlank ("lean") und agil
* Pull-System
* begrenzen den WiP
* setzen auf Transparenz, um den Prozess zu verbessern
* fokussieren darauf, möglichst schnell und möglichst häufig releasefähige Software-Inkremente auszuliefern
* basieren auf selbstorganisierenden Teams
* erfordern, dass Anforderungen in kleine Einheiten heruntergebrochen werden
* In beiden wird der Releaseplan immer wieder optimiert, indem empirische Daten ausgewertet werden (Team-Geschwindigkeit / Durchlaufzeiten)
* **Unterschiede zwischen Kanban und Scrum:**

|  |  |
| --- | --- |
| **Kanban** | **Scrum** |
| Iterationen sind optional. Es kann unterschiedliche Takte für Planung, Releases und Prozessverbesserung geben. | Iterationen mit gleichen Längen sind vorgeschrieben. |
| Commitments sind optional. | Das Team vereinbart, eine bestimmte Menge an Arbeit während der nächsten Iteration zu erledigen. |
| Die Durchlaufzeit (*Cycle Time*) wird als Basis-Metrik für Planung und Prozessverbesserung verwendet. | Die Team-Geschwindigkeit (*Velocity*) ist die Basis-Metrik für Planung und Prozessverbesserung. |
| Cross-funktionale Teams sind optional. Experten-Teams sind erlaubt. | Cross-funktionale Teams sind vorgeschrieben. |
| Keine Vorschrift bezüglich der Größe von Anforderungen. | Anforderungen müssen so aufgeteilt werden, dass sie sich innerhalb einer Iteration erledigen lassen. |
| Es ist kein bestimmter Diagrammtyp vorgeschrieben. | *Burndown*-Charts werden verwendet. |
| WiP wird direkt limitiert. | WiP wird indirekt limitiert (durch die Menge an Anforderungen, die in einen Sprint „passt“). |
| Schätzungen sind optional. | Schätzungen sind vorgeschrieben. |
| Neue Anforderungen können zu jedem Zeitpunkt an das Team gegeben werden, falls Kapazitäten frei sind. | Während eines laufenden Sprints können keine neuen Anforderungen an das Team gegeben werden. |
| Gibt keine Rollen vor. | Schreibt drei Rollen vor (*Product Owner,* Scrum Master*, Team).* |
| Ein Kanban-Board kann von mehreren Teams und/oder Einzelpersonen geteilt werden. | Ein Scrum-Board gehört einem einzelnen Team. |
| Ein Kanban-Board wird kontinuierlich weitergepflegt. | Das Scrum-Board wird nach jedem Sprint gelöscht und neu aufgesetzt. |
| Priorisierung ist optional. | Schreibt vor, dass alle Einträge im *Backlog* priorisiert sein müssen. |

* + **Agiles Testen:**

Wie in allen agilen Entwicklungsprozessen müssen auch in Kanban Qualitätssicherung und Test mit der höheren „Taktfrequenz“ mithalten. Auf allen Testebenen – von Unittest über Integrationstest bis zum Systemtest – entstehen Testaktivitäten, die in „Mikrozyklen“von 24 Stunden Dauer ablaufen können. Allen Beteiligten liegt so stets das tagesaktuelle, vollständige Bild über Entwicklungsstand und Produktqualität vor. Fehler und mögliche Korrekturen können unmittelbar diskutiert, priorisiert und umgesetzt werden.

* **ISTQB\_/GTB Standardglossar der Testbegriffe**

**Deutsch/Englisch**

**Herausgeber: German Testing Board e.V.**

**Dr. Matthias Hamburg, Dr. Uwe Hehn**

[**http://www.software-tester.ch/PDF-Files/CT\_Glossar\_DE\_EN\_V21.pdf**](http://www.software-tester.ch/PDF-Files/CT_Glossar_DE_EN_V21.pdf)

* **ISTQB Lehrplan**
* **Andreas Spillner/Tilo Linz
Basiswissen Softwaretest
Aus- und Weiterbildung zum Certified Tester
Foundation Level nach ISTQB-Standard**
* **Graham Bath / Judy McKay**

**Praxiswissen Softwaretest – Test Analyst und Technical Test Analyst**

**Aus- und Weiterbildung zum Certified Tester
Advanced Level nach ISTQB-Standard**

* **Andreas Spillner/Thomas Roßner/Mario Winter/Tilo Linz**

**Praxiswissen Softwaretest – Testmanagement**

**Aus- und Weiterbildung zum Certified Tester
Advanced Level nach ISTQB-Standard**

* **Wikipedia**

[**http://de.wikipedia.org**](http://de.wikipedia.org)

[**http://de.wikiversity.org/wiki/Kurs:Software-Test**](http://de.wikiversity.org/wiki/Kurs%3ASoftware-Test)