# Vorwort

* Quellen sind Hauptsächlich durch Lektüre gestützt
  + Verwendet selbst spezielle Quellen
  + Wer sind diese Quellen?

# ATDD – Kurzbeschreibung (CHECK – Geschrieben)

* Vorteile ATDD (S2 – Why ATDD Is Beneficial)
  + Automatisierung bringt weitere Vorteile (Automation After Communication – S5)
* Was ist ATDD (S2 – What Are Acceptance Tests)
  + ATDD sind keine traditionellen Akzeptanztests oder Systemtests
  + Die Implementierung folgt der Erstellung von Anforderungstests
* Wo werden AT’s genutzt (S3 – Where Are Acceptance Tests Used)
  + Keine
* Wer erstellt AT’s (S3 – Who Creates the Acceptance Tests)
  + AT werden gemeinsam von Kunden, Entwicklern und Testern entwickelt (S6/S14)
* Welche Typen der Software werden abgedeckt (S3 – What Types of Software Are Covered)
* Ursprung von ATDD (S5 – ATDD Lineage)
* Zitat: „Acceptance test driven development: The answer is 42. Now implement it.“ (S6)

# Einleitung ATTD (CHECK – Geschrieben)

* Software entwickeln (S9/S10)
  + Mit ATTD – ohne ATDD Beispiel / Unterschiede
* Wichtigkeit von ATDD – Beispiel (S10/11)
  + Eine Aussage ungleich ein Test (S11)
    - Eine Aussage kann mehrere AT beinhalten (S14)
  + Absolute Tests sind nicht absolut festgesetzt (S11)
    - 80/20 Prinzip (evtl. Grafik zeigen)
  + AT ist ein Standard für die Verifizierung von richtiger Softwareimplementierung (S12)
* Unterscheidung von AT von anderen Tests (S12)
  + Kunde versteht und spezifiziert mitunter Tests
  + AT ändern sich nicht nach einer Änderung der Implementierung (S14)
* Die Triade
  + Kunde, Entwickler, Tester (S15)
  + Aufgabenverteilung und Agilität (S15)
    - Soccer-Beispiel
  + Weitere Rollen (S16 - Note 2)
  + Erstellen qualitativ hochwertige Software (S21)
* Vergleich zwischen Post- / Pre-Implementation (S17-20)
  + Post – „Klassiche“ / nicht TDD angelehnte Entwicklung
    - Grafik (S17)
    - Nachteile beim Entwickeln
  + Pre – TDD-Style
    - Grafik (S19)
    - Vorteile
      * Quick-Feedback (S18 – Beispiel)
        + Verhindert Loopbacks
      * Face-To-Face (S19)
* ATTD ist „Lean“ und „Agil“ (S20/S21)
  + Von Wem?
  + Ziel
  + Reduziere „Müll“
  + Schaffe Integrität
  + Kontinuierlich verbessern
  + Messbarer Fortschritt

# AT und Test Strategien (S24) (CHECK - Geschrieben)

* AT können und sollten in jeglicher Form auftauchen
  + Jeder Teil der Triade deckt einen Testbereich ab
* Testing Matrix (S24 / Xunit Test Patterns: Refactoring Fig 6.1 „Purpose of Tests“)
  + Erkläre 6 Arten (S24)
  + Funktionale Tests sollten häufig auf mehreren Plattformen laufen (S.28)
* Test Pyramide
  + Zusammenhang Test-Matrix und Test-Pyramide
* Deployments / Plattformen (S25/26)
  + Normalfall
  + GothaerSys-Beispiel
* Test Facetten (S. 27)
* Control / Observer Points für besseres Testen (S.27)
* Tests und Anforderungen sind immer Verknüpft (S.28)
* Neuer Test bedeutet neue Anforderung (S27)
  + Neue Anforderung kann ein Detail einer aktuellen Anforderung sein
  + Anforderung ungleich Feature – wird später im Detail erklärt
  + Quote: „Any test created after the code is written is a new requirement or a new detail on an existing requirement.“ (S.27)
    - Wichtige Aussage, da TDD-Style oberste Priorität hat
    - Entwickler tendieren gerne dazu, mal „schnell“ einen zusätzlichen Test zu schreiben 🡪 gegen TDD
  + Ein „Bug“ kann ebenfalls eine neue Anforderung sein (S.27)
    - „Bug“ in diesem Sinne: eine Anforderung, welche durch Missverständnisse oder fehlendes Wissen erst nach der Entwicklung bekannt wurde, und somit kein echter Bug ist, sondern eine neue Anforderung 🡪 hier gilt es stets zu Unterscheiden
      * Analysis bug

# Akzeptanztest Beispiel (CHECK – Gothaer BSP fehlt)

* Simple Business-Regel (S29)
  + Beispiel: Erste Tabellen und Interpretation (S30)
  + Austausch innerhalb der Triade face-to-face für richtiges Verständnis von BR
  + Durch Beteiligung mehrerer Stakeholder ergeben sich mehr Fälle (S31)
  + Beispiele der Anforderungen verdeutlichen Anforderungen – schaffen Transparenz über die Anforderung (S37)
    - * Beispiele können direkt als Basis für Tests verwendet werden (S37)
* Einen Akzeptanztest implementieren (S31)
  + Test Script (S31/32)
    - Beispiel: Selenium (Gothaer Beispiel - GEPR)
    - Abhängigkeit vom Interface kann Nachteil sein (Recherchiere: „Nachteil UI Abhängigkeiten“)
  + Test User Interface (S33)
    - Beispiel: GUI / CLI
      * Gothaer Beispiel: Barbara Hamacher – Host-Facettentest?
    - Analogie (S33)
    - Grafik (S34 Figure 4.3)
    - Ruft BR direkt auf (S37)
  + xUnit Test (S34/35)
    - Business Rule Test auf Unit Ebene schwer
      * Änderungen der BR führen zu Änderungen jedes Unit-Tests (S35)
      * Nutzen eignet sich für Testen von implementierter Logik
  + Automatisierter Akzeptanztest (S35/36)
    - Tabellenform muss nicht zwingend konvertiert werden (S35)
      * Weniger Aufwand
      * Möglichkeit: Vorher absprechen, wie Anforderungen notiert werden 🡪 Übereinstimmung mit Style vom Test-Framework für weniger Aufwand und leichtere Umsetzung
    - Kommuniziert direkt mit Modul der BR (S37)
  + Wenn Test nicht als „Test Script“ implementiert ist: Overall Test zusätzlich betrachten
    - Abhängig von Komplexität Anzahl an Testdaten bestimmen (S36)
  + Ein Akzeptanztest verifizieren reicht nicht aus, um sicherzustellen, dass das System den Benutzeranforderungen gerecht wird (Siehe Test Strategien) (S37)

# Entwicklungsprozess

* Kommunikation (S66/S67)
  + Relevanz / Wichtigkeit
    - Kollaborationen müssen unterschiede Verstehen, wie Menschen Prozesse erstellen und verarbeiten (S68)
  + Direkte Kommunikation (Face-to-Face) bietet Vorteile
    - „Kommunikationsprozess“ erläutern
    - Aktives Zuhören / Schreiben
    - Gestiken / Mimik
    - Direktes Feedback
    - Unterschiede
      * Big Picture (Intuitors) / Details (Sensors)
      * Extrovertiert / introvertiert
  + Mehr als nur Wörter (S67)
    - Quote: „Communication is about more than just words. It’s aboput how you organize those words.“ (S68)
* Beschreibt den Entwicklungsprozess von der Erstellung von Akzeptanztests über den Kunden, bis hin zur Implementierung dieser hinsichtlich von BR’s, verschiedenen Tests auf verschiedener Basis / Ebene
* Grafik: Charter 🡪 Feature 🡪 Story 🡪 Scenario 🡪 Test (S39)
  + Idee: Zeige im Verlauf Grafiken mit einem Aspekt als Fokus, um den Überblick zu behalten, in welchen Schritt man sich befindet
* Charter
  + Definition
  + Erste Idee von Projekt (S40)
    - Vision, Mission, Zielsetzung, Prinzipien (S41/S46)
    - Ziele sollten SMART sein (S40/S46)
      * Specific, Measurable, Achievable, Relevant, Time boxed
      * Beispiele: good / bad practices
    - Projekt-Akzeptanztests (Objectives [Zielsetzung])
      * Objekte [Zielsetzung] repräsentieren Akzeptanztests für das ganze Projekt (S46)
      * Management Tests (Note 1: S41)
      * Return of Investment (Recherchieren Internet / Spätere Kapitel)
      * Verzögerung von Projekt 🡪 reduziert ROI (S41)
      * Beispiel (S41)
  + Mini-Charter für jedes Release (Bsp. Gothaer GTRM Zyklus) (S43))
    - Time-Box ist dann oft an Release angelehnt
      * Wenn nicht machbar, dann splitten in mehr objectives (subobjectives)
* Feature
  + Definition
  + Aus Charter „High-Level Requirements“ (FEATURES) erstellen (S43)
    - High-Level Requirements sollen Objekte [Zielsetzung] unterstützen (S46)
      * Geben einen besseren Einblick des Ziels
      * HLR besitzen High-Level-Akzeptanzkriterien
    - Brainstorming Workshop mit Moderator mit Facilitator (S43)
      * <https://kommunikationslotsen.de/facilitation/>
      * Kreativität und verschiedene Sichtweisen für Finden von Features (S44)
      * Beispiele für Brainstorming „Spiele“
    - Beispiel-Feature Liste
      * Nicht alle Features müssen für das jetzige Ziel umgesetzt werden – verlgeich: expansion Plan vs. First Projekt Plan (S44)
      * Reduzierung auf Kern Features für Projektziel
      * Können aufgehoben werden
  + Aus Feature 🡪 Akzeptanzkriterien für Feature finden
    - Tester können helfen relevante Kriterien zu finden (S45)
    - Kalkuliere Business-Wert (z.B.: High-Level ROI) (S45)
      * Priorität in Iterationen eingliedern (SCRUM)
      * Priorität in Queue eingliedern (KANBAN)
* Stories
  + Definition
  + Features werden in Stories runtergebrochen (S47)
  + Story Map kann helfen Workflows (Feature) darzustellen (S64)
    - Ein Feature bestehend aus mehreren Stories
  + Stories sollten nach dem INVEST Kriterium erstellt werden (S46/S56)
    - Auflistung einzelner Faktoren (S55)
  + Beteiligte (Stakeholder) an Meetings können sich hier ändern (S48)
    - Z.B.: Sponsor ist hier nicht nötig, da HLR ausreicht
  + Stories können verschiedene Arten sein (S48/S56)
    - Requirement Story (Kleiner Teil vom Feature)
      * User Story
        + Story, mit der der User in Berührung steht
        + Aus Extreme Programming (S48)

Formuliere Stories: Als <Rolle> möchte ich <tue etwas>, sodass <Begründung> (S48)

* + - * + User Stories besitzt Rolle, Aktion, Grund (S51/S56)
      * Constraint Story
        + Einfache Bedingung zum System sein
        + Hier ist oft Constraint Akzeptanzkriterium
    - Developer Story (S54)
      * Story entsteht in der Regel aus Requirement Stories
      * Technische Story für Entwickler für Implementierung
  + Fachliche Begrifflichkeiten (S54)
    - Wichtig, vorher abzustimmen, um Missverständnisse zu vermeiden
    - Einheitliches Verständnis und Definitionen für bessere Kommunikation
    - Stories sollten in Kundensprache geschrieben werden (S56)
  + Rollen (S49)
    - Brainstorming mit Facilitator
    - Findung verschiedener User-Arten für Anwender und diverser Szenarien bzw. Prozesse(S56) (Admin, Standardnutzer, spezielle Nutzer)
      * Keine Individuen (S56)
    - Können gut für Exploratory Tests genutzt werden
      * Prüfen, ob sich das System mit verwendeter Rolle richtig verhält
    - Rollen gliedern sich nach Interessen und Aufgaben
    - Rollenattribute (S49/S50)
      * Nutzungsfrequenz (Wie oft benutzt die Rolle das System)
      * Domäne (Welche Bereiche eines Systems nutzt die Rolle)
      * Technisches Wissen (Wie gut kennt sich die Rolle mit dem System / der Technik aus)
      * Generelle Ziele (Was möchte die Rolle erreichen)
      * Beispiel
      * Verschiedene Sichtweisen und Attributarten helfen beim Finden von weiteren Rollen und Aufgaben
    - Erstellen von Persona können zum Verständnis Helfen (S50/S51)
      * Imaginäre Person
      * Verhalten, Eigenschaften, Charakter, Hobbies
      * Geben ein Bild zu einer Rolle ab
  + Erstelle Akzeptanzkriterien für Stories (S52) [ähnlich wie bei Features (HLR)]
    - Helfen bei der Einschätzung für Größe / Aufwand (S56)
  + Stories können mit ROI eingeschätzt werden für Business Wert (S53)
    - Recherchieren, Vor- und Nachteile?
    - Nicht alle Stories besitzen Business Wert, da sie erst mit anderen Kombiniert einen liefern (S54)
  + Wenn Stories zu groß erscheinen 🡪 erstelle mehrere Sub-Stories (S53)
  + Erstelle später Developer Stories auf Basis von User / Constraint Stories (S54)
    - Hier nur Entwickler und Tester
* Scenario
  + Use-Cases finden (S57)
    - Beschreibt Szenario (S68)
    - Evtl. noch weitere Methoden? (Recherche)
      * Note S57: Event/Response Chapter 15
    - Definition / Vorteile Use-Case
    - Template (S59/S68)
    - Vorsicht: Analysis Paralysis (S58)
      * Entwickle erst dann Use-Case, wenn Story benötigt wird
    - Involviert i.d.R. anderen Use-Case oder Aktion, welche außerhalb des Softwaresystems auftritt. (S59)
    - Hier Kann Kunde mitwirken und formalen Prozess / Workflow beschreiben (S58)
      * Mit Tester/Entwickler wird Prozessablauf notiert
    - Positivfall / Exceptions / Alternativen / Business Rules(S60/61)
      * Mit im Template als Unterpunkt
    - Wenn Use-Case mit Exceptions zu groß 🡪 erstelle spezielle Exception Use-Cases (S62) 🡪 eigene User Stories (S68)
  + Für jeden Use-Case Akzeptanztests erstellen (S63/S68)
    - Mindestens einen für Main Course, Exceptions, Alternatives
    - Pre-Kondition sind Setup
    - Post-Kondition sind erwartetes Ergebnis
      * Wenn abgebrochen, anderes Ergebnis (S63/S68)
      * Alternativen führen auch zu Post-Kondition (S68)
    - Komplizierte Business Rules können in eigene Tests enden
      * Positiv und Negativtests
  + Use-Cases eignen sich für Dokumentation (S63)
  + Prototyp-Interfaces / Use-Case Diagramme helfen für den Kunden zum Verständnis (S65/S66)
* Test
  + Alle sind beteiligt (S69)
    - Tester schreib zb detailliertere, technischere Tests für die Aufgabe
      * Weißt welche Inputs wichtig/kritisch sein können
  + Nicht alle Tests werden immer direkt gefunden (S70)
    - Während Implementierung oder Exploration Tests
    - Wenn das häufig passiert 🡪 herangehensweise überprüfung und Wurzelproblem dazu finden
  + Given-When-Then Struktur (S71 / S81)
    - Figure 8.3
    - Beispiel aus 1. Kapitel
      * Business Rule Test / calculation test
  + Verbessere Verständnis durch Test-Struktur durch Domain-Sprache
  + Jedes Szenario ist anders 🡪 unterschiedliche Tests
    - Manchmal einer, manchmal ganzes Set
  + Test-Informationen darstellen (S81)
    - Calculation Table für BR (S73)
    - Data Table (S74)
      * Muss nicht wie Datenbank sein 🡪freie Collection
    - Action Table (S75)
      * Beschreibt Ablauf
  + Bewerte Tests nach Erstellung für Erweiterungen/Ergänzungen (S77)
  + Vergleich Table / Text Form (S80)
  + Hinweis von vorherigen Kapiteln: Tests können auf allen Leveln laufen (S80)

# Test-Frameworks

* Frameworks (S298)

# Zusammenfassung