tgm
Die Schule der Technik

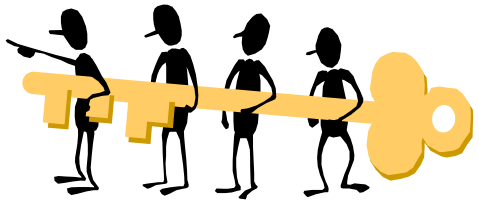
Themencluster: Agile und schlanke Methoden

Thema: Acceptance Test Driven Development

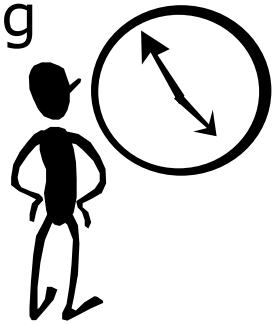
Dr. Walter Rafeiner-Magor
09.09.2013

# Acceptance test-driven development

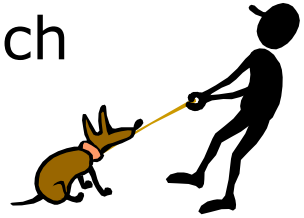
- **Wozu?**
  - Unterschiedliche Teams haben unterschiedliche Wege eine Software zu entwickeln.



  - Vorhandene Acceptance-Tests geben dem Entwickler die Möglichkeit seine Entwicklung daran (objektiv) zu messen.



  - Wenn der Inhalt und die Anzahl der Tests dem Aufwand für die Implementierung entsprechen, kann daraus auch der (ungefähre) Fortschritt abgelesen werden.
  - Acceptance-Tests ändern sich nicht – auch nicht bei Änderungen in der Implementierung!

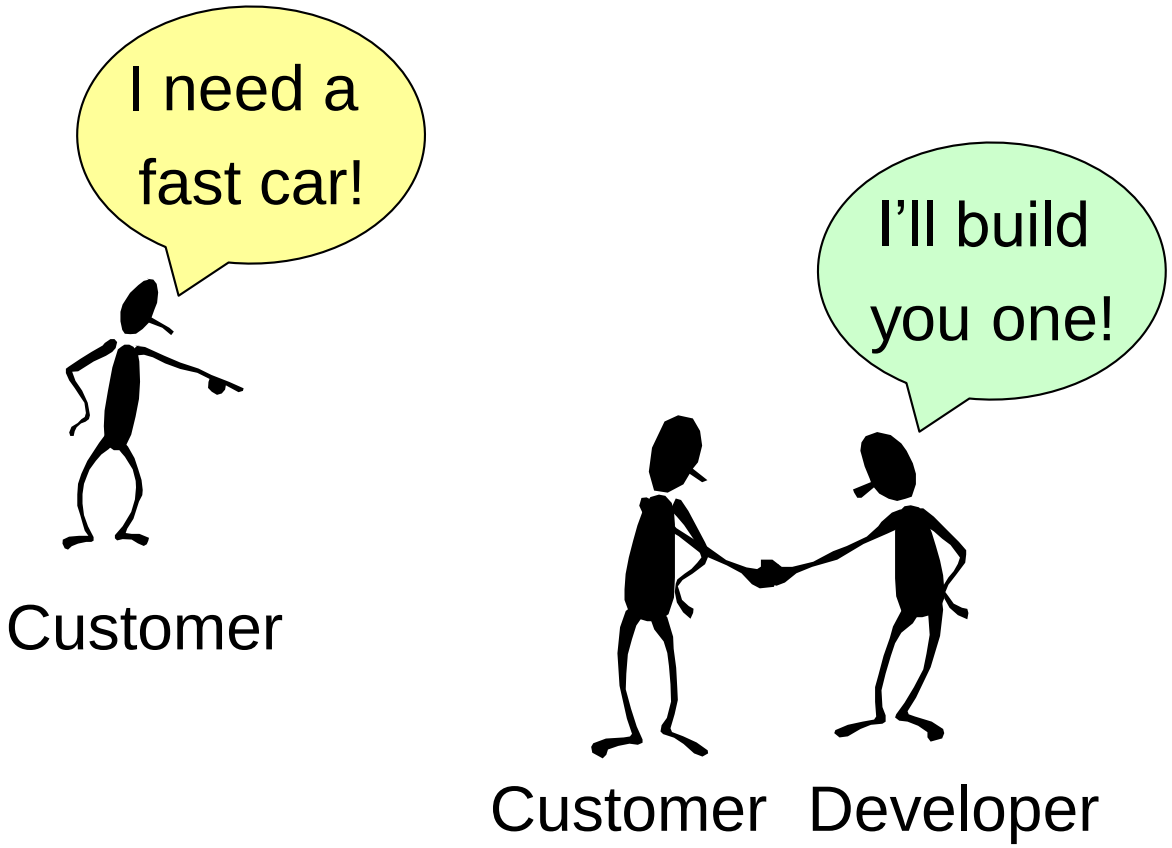






Quelle: projektmanagementhandbuch.de

# Acceptance test-driven development

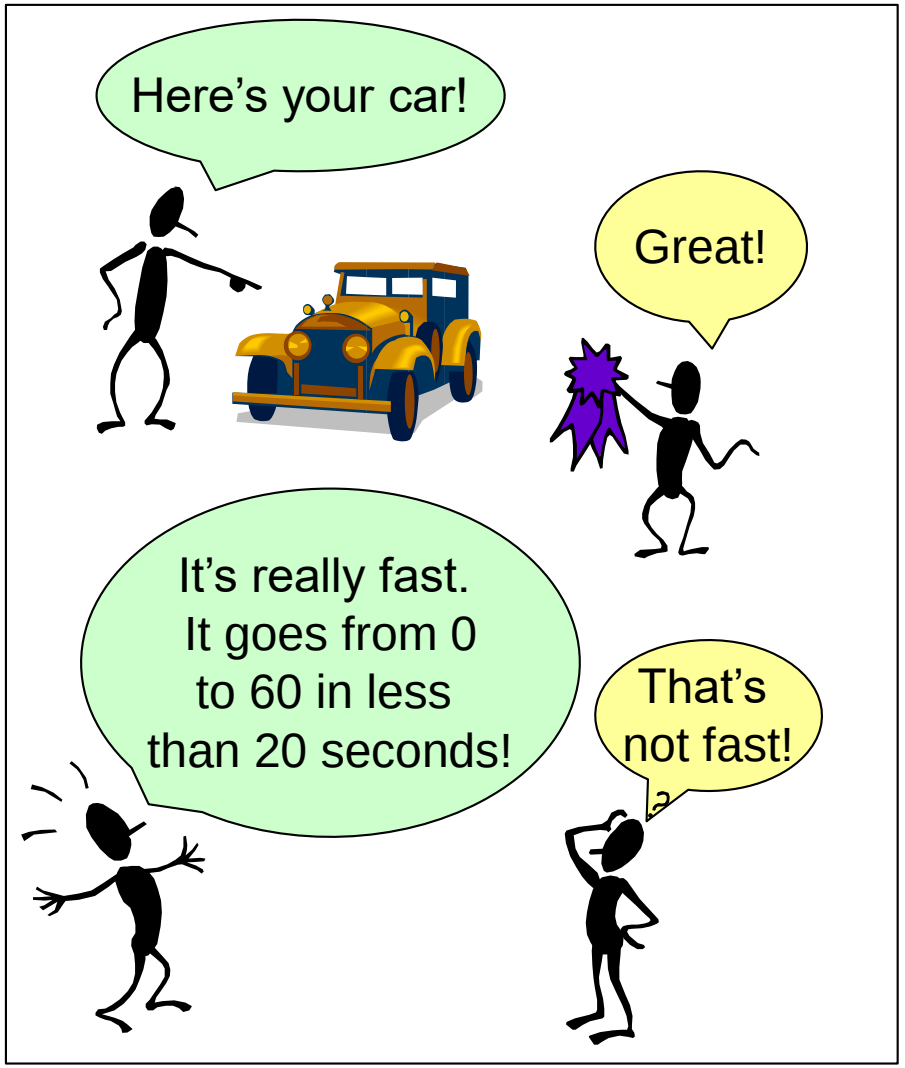


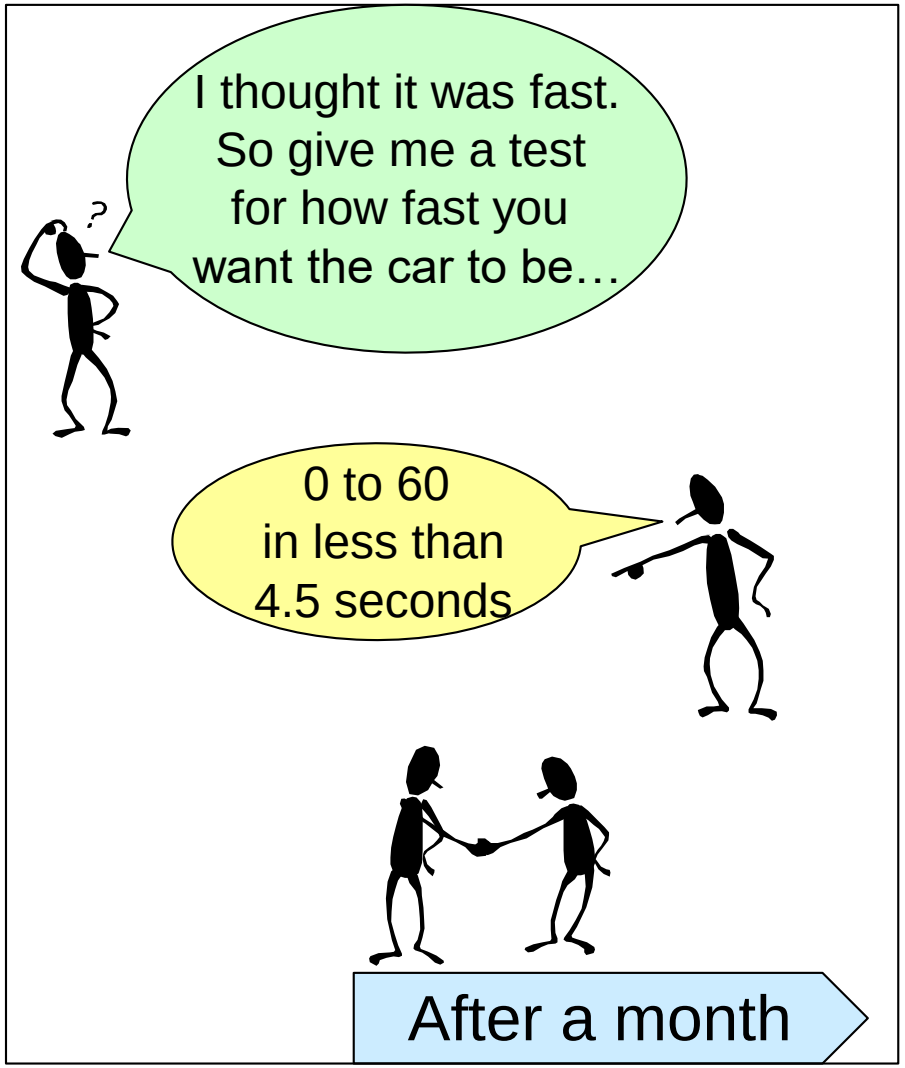

After a month



Quelle: Lean-Agile Acceptance Test - Driven Development

# Acceptance test-driven development

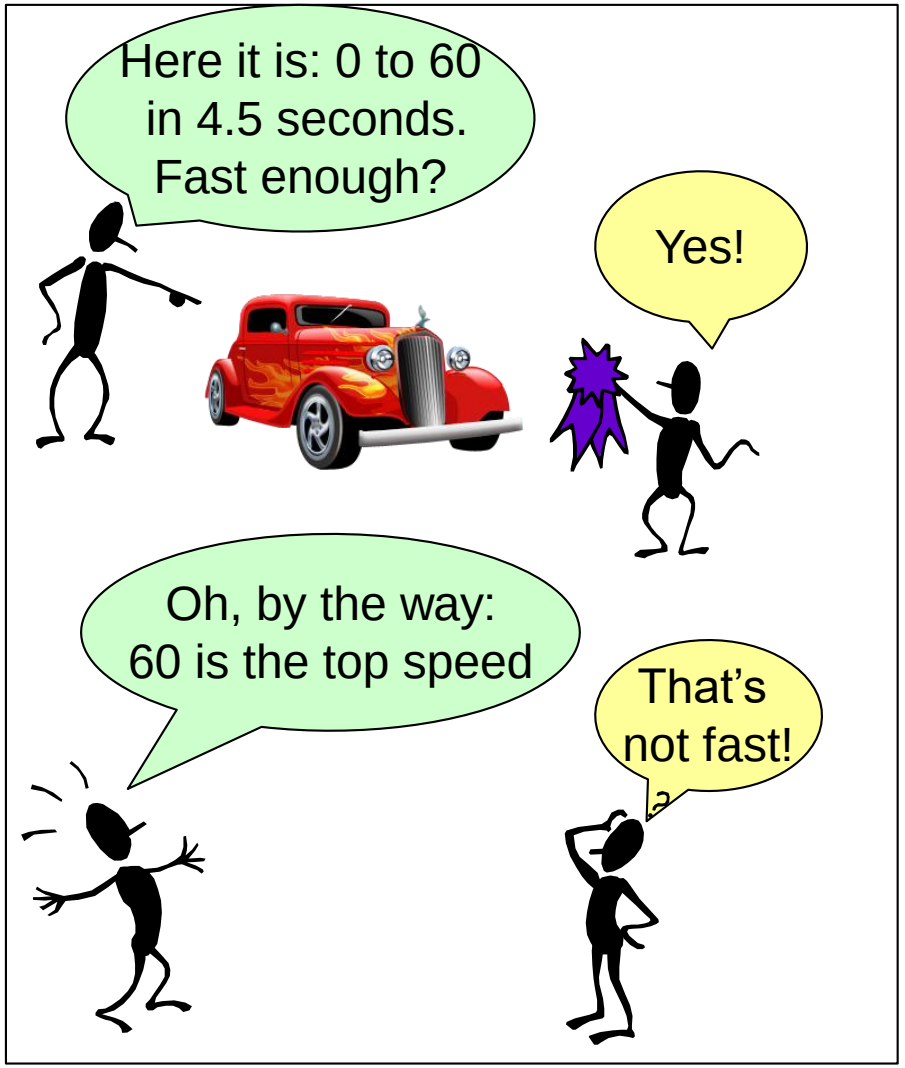


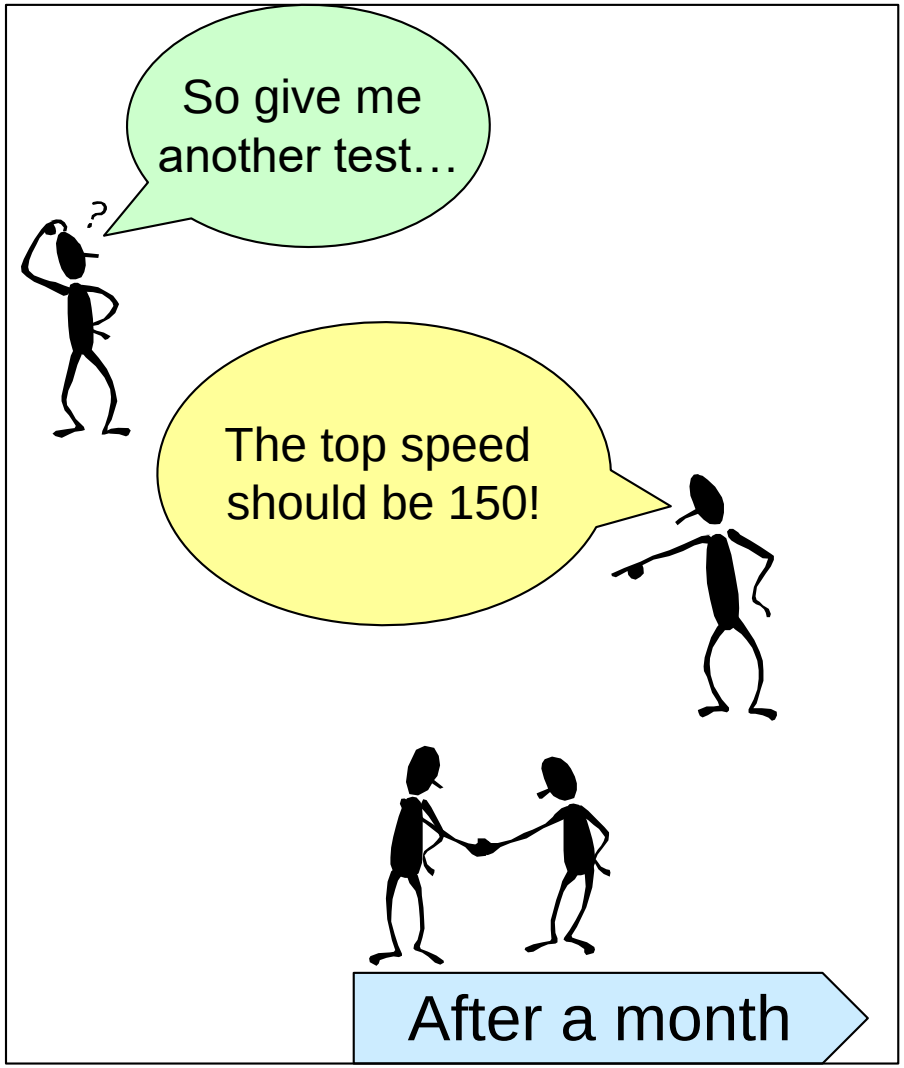






Quelle: Lean-Agile Acceptance Test - Driven Development

# Acceptance test-driven development

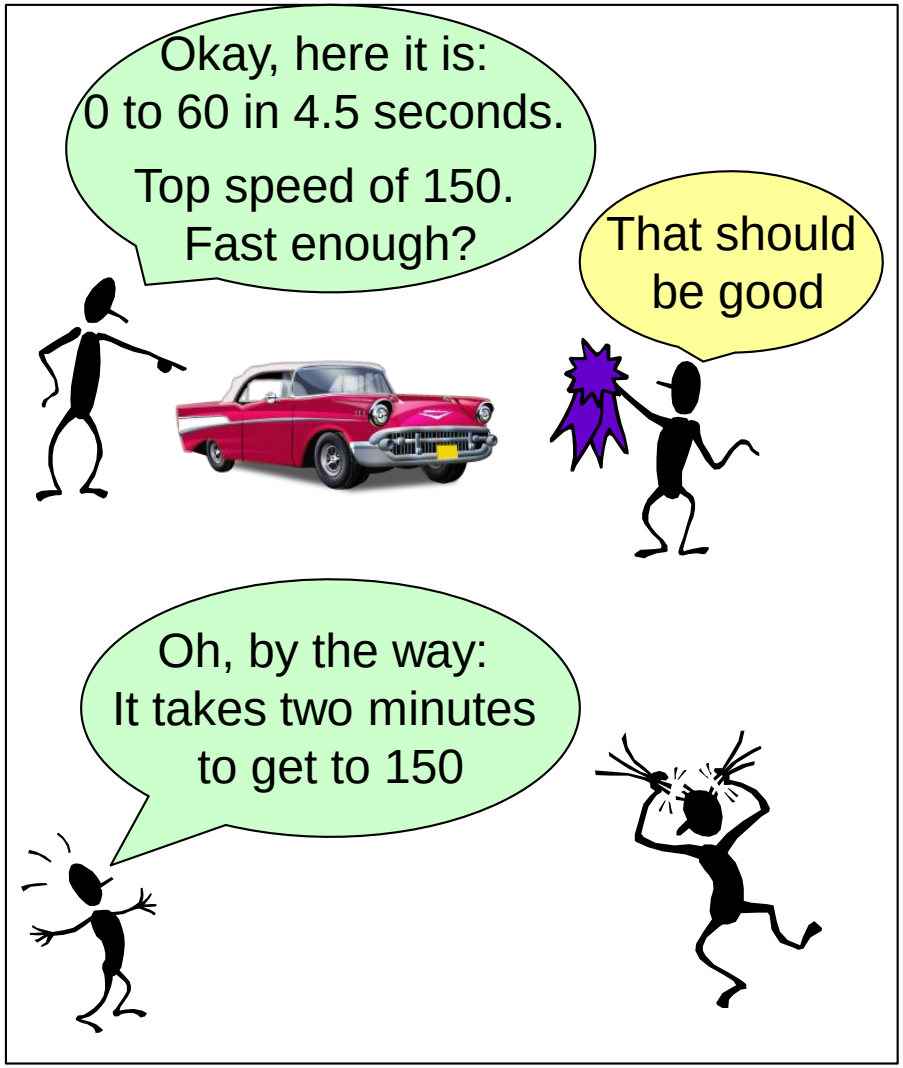





Quelle: Lean-Agile Acceptance Test - Driven Development

# Acceptance test-driven development

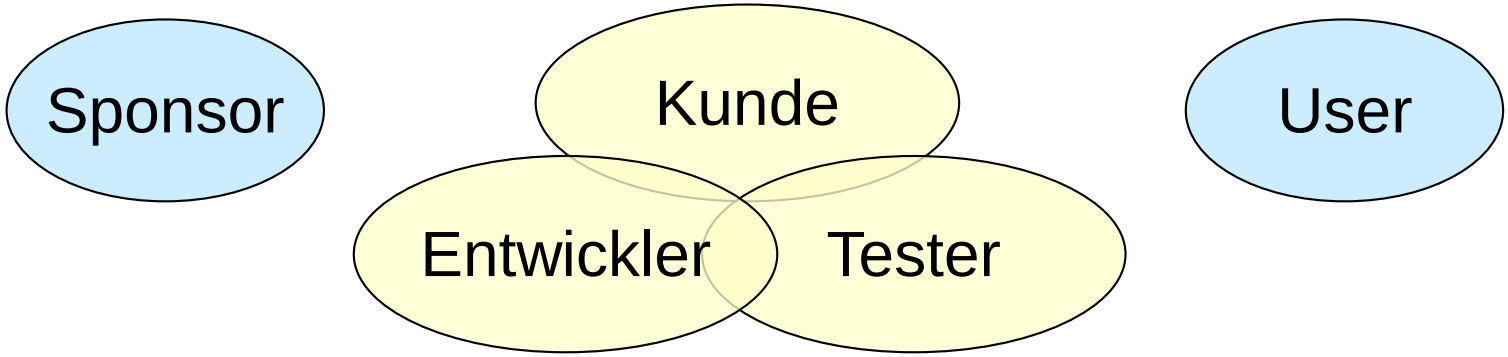



- **fast car!**
  - Die Anforderungen sind nicht eindeutig!
  - Der Kunde erstellt Tests zur Klarstellung „fast car“
- **definierte Tests können auch angepasst werden**
  - “0 to 60 in less than 4.5 seconds”
    Was ist zu tun, falls die Implementierung 4.6 Sekunden erreicht?
  - Der Kunde entscheidet, ob der Mehraufwand für den Verkauf des Autos notwendig und sinnvoll ist!



Quelle: Lean-Agile Acceptance Test - Driven Development

# ATDD: System und Rollen




- **Anforderungen**
  - Jede Anforderung hat zumindest einen Acceptancetest
- **Acceptance test**
  - Sichert die Korrektheit der Implementierung
  - Die Erstellung erfolgt in Zusammenarbeit mit dem Kunden!



Quelle: Lean-Agile Acceptance Test - Driven Development

# ATDD: Lean Software Development

- **Beginn**
  - Die sieben Prinzipien der schlanken Softwareentwicklung gehen auf Mary und Tom Poppendieck zurück.
- **Die sieben Prinzipien**
  - Eliminate waste
  - Amplify learning
  - Decide as late as possible
  - Deliver as fast as possible
  - Empower the team
  - Build integrity in
  - See the whole





Quelle: Poppendieck, Mary, and Tom Poppendieck. Implementing Lean Software Development: From Concept to Cash

# ATDD: Lean Software Development

## Eliminate waste:

Nur notwendige Funktionalität

Keine Verzögerungen im SEP

Klare Anforderungen

Vollständiges Testen

Keine Bürokratie

Effiziente Kommunikation

## Amplify learning:

Tests sobald der Code vorhanden

Statt detaillierter Planung: Alternativen

Kurze Iterationszyklen

Häufige kurze Feedback-Sitzungen

Diskussion der Randbedingungen

auf Basis der zukünftigen Lösung

## Decide as late as possible:

Entscheidungen bei Fakten (nicht aufgrund von Annahmen bzw. Vorhersagen)

Iterative Ansatz: kostengünstigere Ansatz für Änderung bzw. Fehlerbehebung

Je größer das geplante System, desto mehr Kapazität für Änderungen einplanen

# ATDD: Lean Software Development

## Deliver as fast as possible:

Die aktuellen Anforderungen des Kunden können umgesetzt werden.

set-based design: Parallele Erstellung von Alternativen, Auswahl der Besten

Je früher eine Version geliefert wird, desto schneller kann Feedback eingearbeitet und eine neue Iteration gestartet werden.

## Empower the team:

"find good people and let them do their own job“

Entwickler benötigen realistische Ziele, Motivation und die Möglichkeit eigene Entscheidungen zu treffen.



Quelle: Lean-Agile Acceptance Test - Driven Development

# ATDD: Lean Software Development

## Build integrity in:

Wahrgenommene Integrität: Preis, wie wird es vorgestellt, geliefert, installiert wie intuitiv ist die Verwendung; wie gut werden die Probleme gelöst?

Konzeptuelle Integrität: System ist in Komponenten geteilt, welche im Bezug auf Flexibilität, Wartbarkeit, Effektivität und Reaktionszeiten gut zusammen-arbeiten.

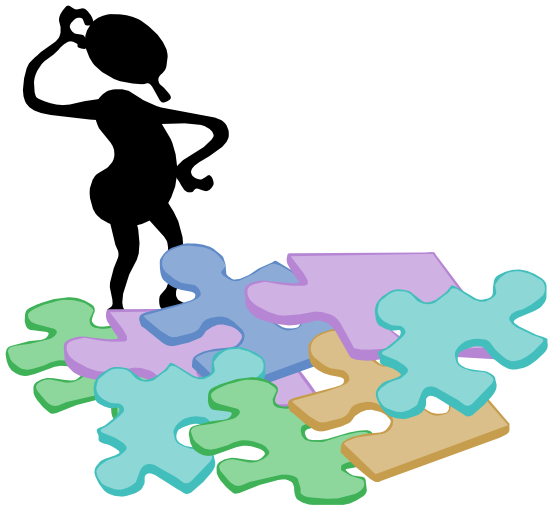
## See the whole:

“Think big, act small, fail fast; learn rapidly"

Heutige Softwaresysteme sind nicht einfach die Summe der Teile, sondern auch die Beziehung zwischen den Komponenten.

# ATDD: Test-Strategien

- **Was wird getestet:**
  - Acceptance-Tests sind Funktionstests (OAT, UAT)
  - Unit-Tests lenken die Entwicklung
  - Tests und Anforderungen sind verknüpft
  - Tests sichern die Funktion von Komponenten und Modulen
- **Wo und wie wird getestet:**
  - Tests finden auf Test- und Integrationsplattformen statt
  - Häufige Tests sind notwendig!

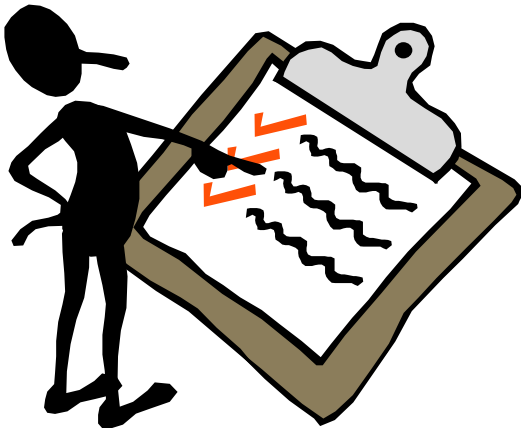




Quelle: Lean-Agile Acceptance Test - Driven Development

# ATDD: Umsetzung der Acceptancetests

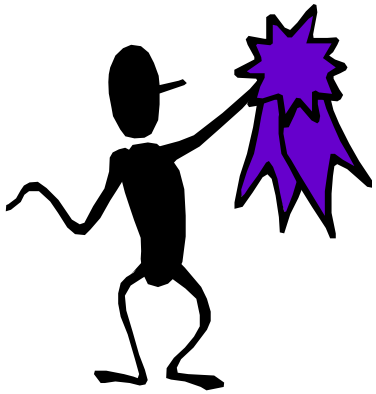
- **Ausgangspunkt:**
  - Konkrete Beispiele erläutern die Anforderungen
  - Die Beispiele können als Test für die Implementierung verwendet werden
- **Umsetzungsmöglichkeiten:**
  - Die Implementierung des Use-Cases wird mittels UI getestet
  - Unit-Test in einem Testing-Framework (JUnit, PHPUnit,…)
  - Ein automatischer Test für den Geschäftsanwendungsfall
  - Integrations- und Systemtests (E2E)



Quelle: Lean-Agile Acceptance Test - Driven Development

# ATDD: Zusammenfassung

- **Vorteile:**
  - Die erstellten Tests geben dem Entwickler wiederholbare Kontrolle, ob die Anforderungen an die Funktionalität erfüllt sind.
  - Fortschrittskontrolle durch regelmäßiges Feedback des Kunden
- **Agiles Manifest:**
  - Menschen wichtiger als Prozesse: ja
  - Laufende Software wichtiger als Doku: ja
  - Zusammenarbeit mit dem Kunden wichtiger als Vertragsverhandlungen: ja
  - Veränderungen begrüßen statt Planverfolgung: ja



# Vielen Dank!