# *Practical Software Engineering*

## *Agile Methods*

FH JOANNEUM
Internettechnik
Software Design
WS 2018

# *A look at history…*

**The “beginning”:**

- 60ties From “Hacking” to  “Concurrency“
- 70ties “Structured methods” (Waterfall)
- Late 80ties RAD, CMM, Spiral, OO Methods

**Paradigm shift 1**:

“Software development is like production”

- 90ties Heavyweight Methods (e.g. RUP)

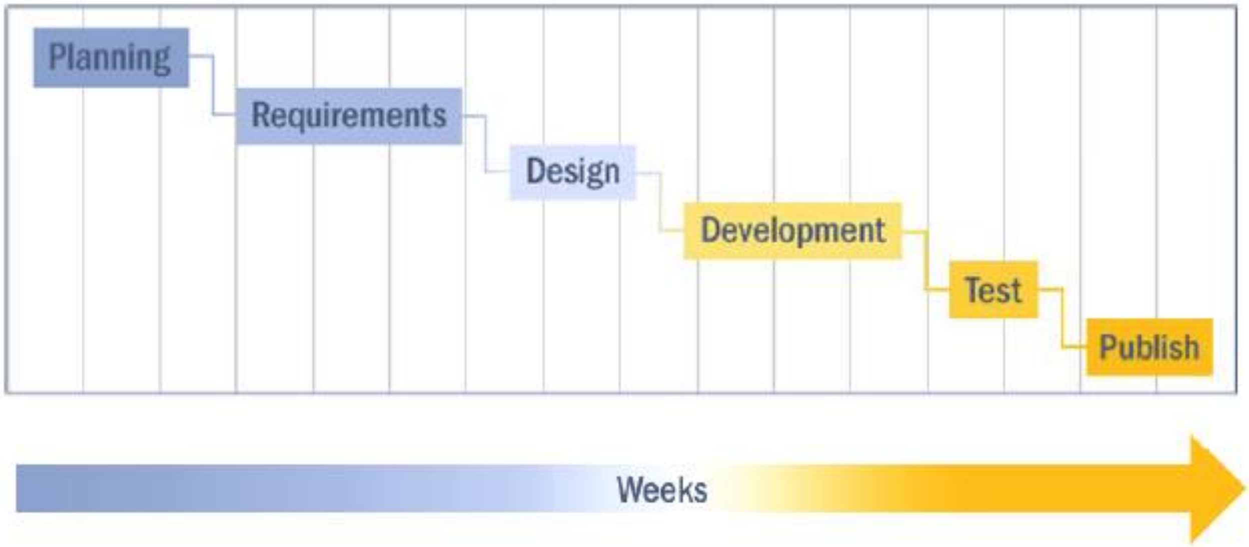
**Paradigm shift 2**:

“Software development is like developing new products (all at once)”

- Millenium change: Leightweight methods / Agile

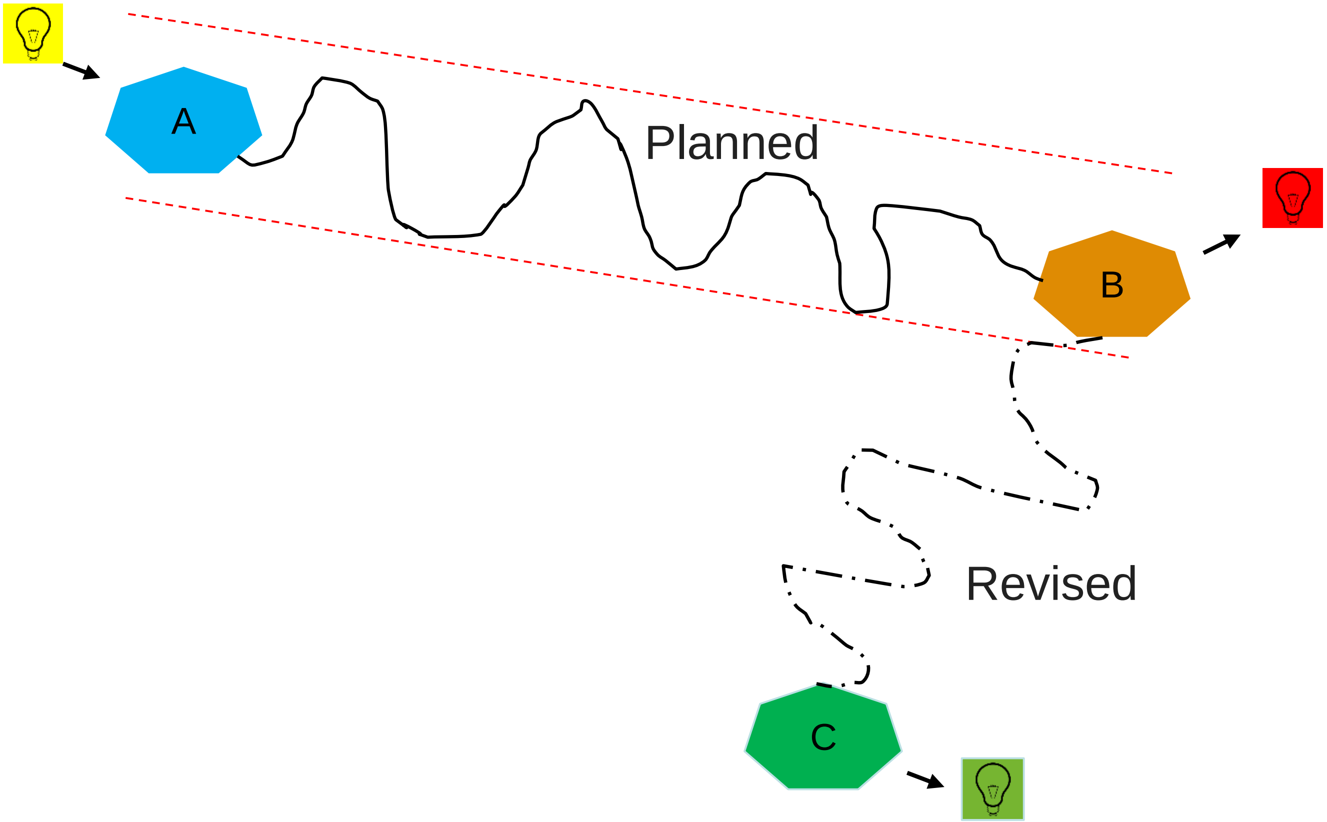
# *The waterfall model*

- A “plan driven” approach



© by Mitch Lacey (taken from the „SCRUM for Managers.pdf“, Link: http://mitch.lacey.com)

# *A typical Software Project…*



# … the end

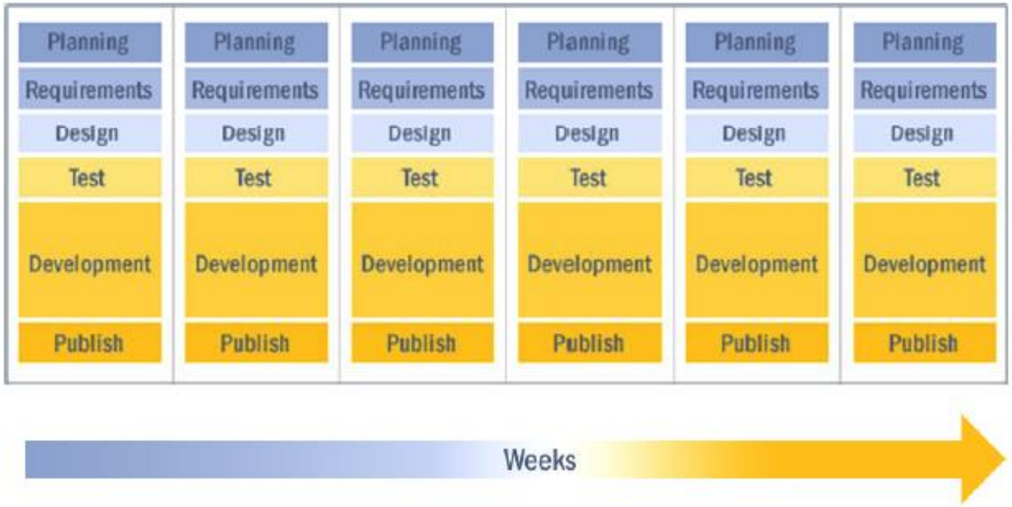
# *Why?*

- Things Change…
  - Customers often do not exactly know what they want or need at the beginning of a project
- Late feedback
  - Customers
  - Design vs. Implementation
  - Long development phases
  - Quality (test phase)
  - Time to market
  - Budget

# *Evolutionary /Iterative Approach*

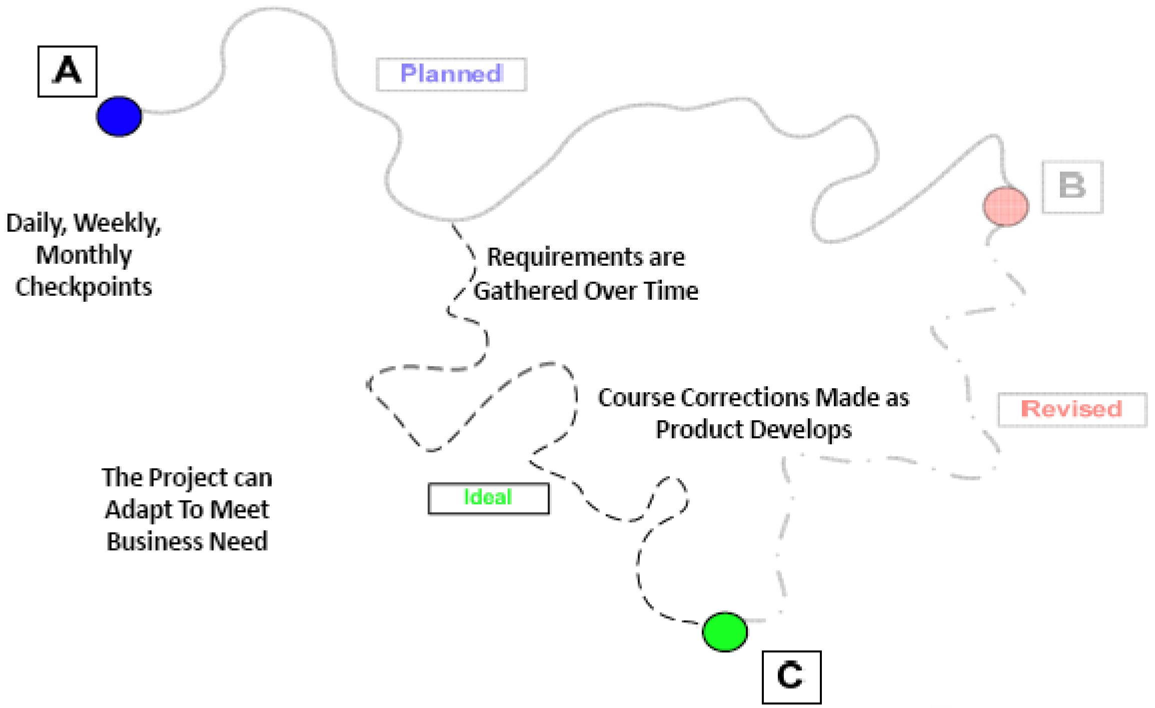
**Iterative development** breaks down a project by subsets of functionality. Each iteration produces tested, integrated code that is as close to production quality as possible.

„Do a bit of everything in each iteration“



© by Mitch Lacey (taken from the „SCRUM for Managers.pdf“, Link: http://mitch.lacey.com)

# *Goal of the Evolutionary / Iterative Approach*


A
Planned
B
Daily, Weekly, Monthly Checkpoints
Requirements are Gathered Over Time
Course Corrections Made as Product Develops
Revised
The Project can Adapt To Meet Business Need
Ideal
C

© by Mitch Lacey (taken from the „SCRUM for Managers.pdf“, Link: http://mitch.lacey.com)

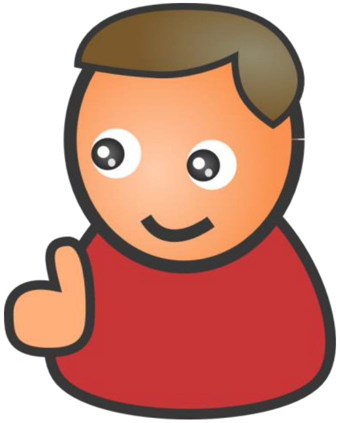
# DER PERMANENTE ENGPASS

Das Projekt läuft nicht gut…

Ich tue eh schon was ich kann!

Wo sind die nächsten Tasks?
RTFM

Meeting! Meeting! Meeting!

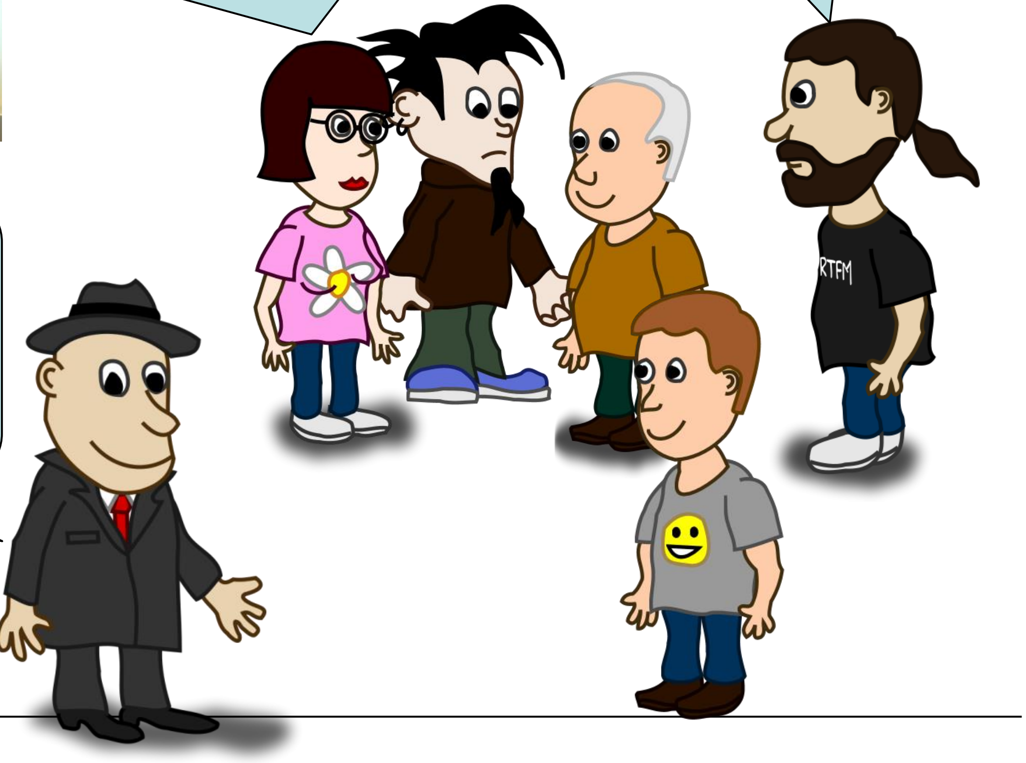


Ich bin der permanente Engpass!

Noch mehr Meetings…

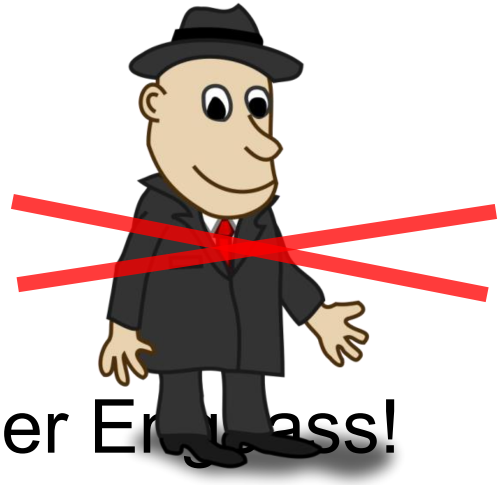
Noch mehr Auswertungen…

# *Bekannte Situation?*



Task B fertig! Starte mit Task D
Hab mir Task E geschnappt!
Was kann ich tun für Euch?
RTFM

# *Er ist weg…*



…der Er[illegible]ass!

# *Kernaspekte für das Team*

- Kollektive Verantwortung
- Selbstorganisation des Einzelnen
- Kontinuierliche Verbesserung
- Effizienz

# *Agile Software Development*

## Manifesto for Agile Software Development (2001)

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

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

# *Agile Software Development*

## Manifesto for Agile Software Development

- **Individuals and interactions**.
  - People are the most important ingredient of success.
  - A team of average programmers who communicate well are more likely to succeed than a group of superstars who fail to interact as a team.
  - Building a team is more important than building an environment.

# *Agile Software Development*

## Manifesto for Agile Software Development

- **Working software**.
  - It is always a good idea for the team to write and maintain a rationale and structure document, but that document needs to be short.
  - Too much documentation is worse than too little.
  - Produce no document unless its need is immediate and significant.

# *Agile Software Development*

## Manifesto for Agile Software Development

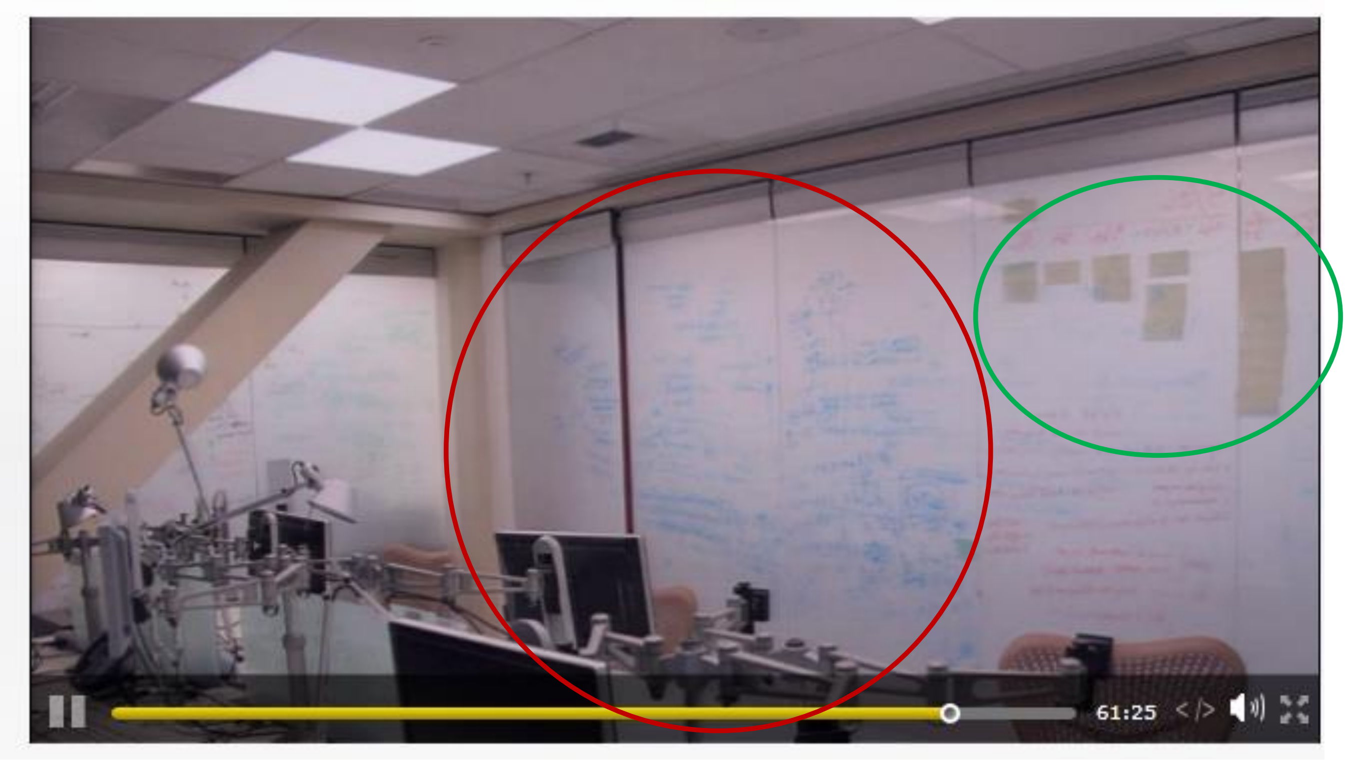
- **Customer collaboration**.
  - Successful projects involve customer feedback on a regular and frequent basis.
  - Rather than depending on a contract, the customer of the software works closely with the development team.

# *Agile Software Development*

## Manifesto for Agile Software Development

- **Responding to change**.
  - The business environment is likely to change, causing the software requirements to change.
  - Customers are likely to alter the requirements once they see the system start to function.

# „Agile“ is…

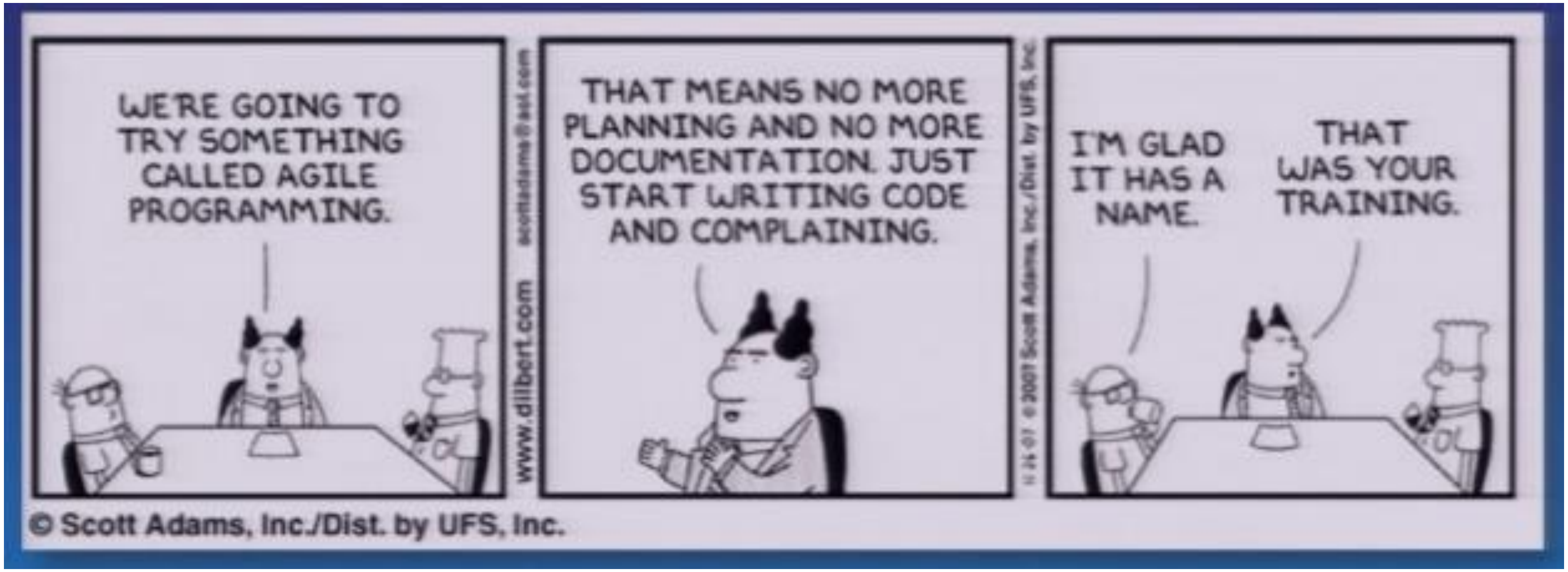


Task Tracking (SCRUM)

Architektur (Doku über Fotos)

**… at Microsoft**

# *„Agile“ is not…*


WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.
www.dilbert.com
scottadams@aol.com
11-26-07 © 2007 Scott Adams, Inc./Dist. by UFS, Inc.

© Scott Adams, Inc./Dist. by UFS, Inc.

# *Agile Software Development*

## Flavors of Agile Development

- **Extreme Programming**
  (Kent Beck)
  Many of XP's practices are old, tried and tested techniques. As well as resurrecting these techniques, XP weaves them into a synergistic whole where each one is reinforced by the others and given purpose by the values.

- **Scrum**
  (Ken Schwaber, Jeff Sutherland, and Mike Beedle)
  Scrum concentrates on the management aspects of software development, dividing development into thirty day iterations (called 'sprints') and applying closer monitoring and control with daily scrum meetings.

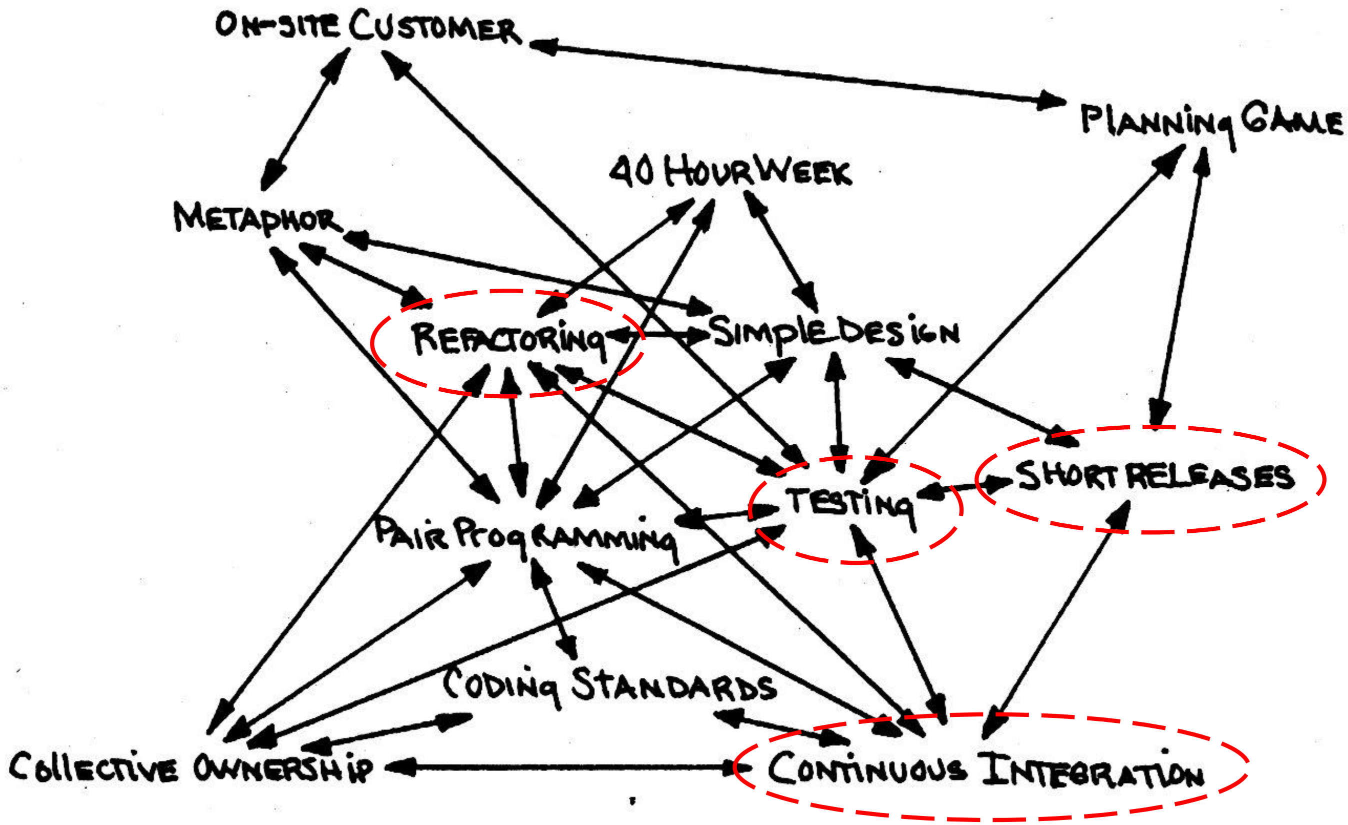
# *Agile Software Development*

## Flavors of Agile Development

- **Crystal**
  (Alistair Cockburn )
   (focusses on projects, family of processes with tailoring, similar ideas to agile Manifesto)
- **Context Driven Testing**
  (Brian Marick, Brett Pettichord, James Bach, Cem Karner)
  (main aspect: brings testers in process and not only developers)
- **Lean Development**
  (Mary and Tom Poppendieck )
  (Origin in automotive production [lean processes, flat hierarchies], influenced agile)
- **Kanban**
  (David Anderson)
  (Origin in automotive production, Reduced parallelity, enhanced time to market)
- **Rational Unified Process**
  (Phillippe Kruchten, Craig Larman)
  (large iterative framework, tailoring necessary, use case driven)

# *Agile Software Development*

**Extreme Programming** (K Beck, W Cunnigham)



## Agile Software Development Techniques

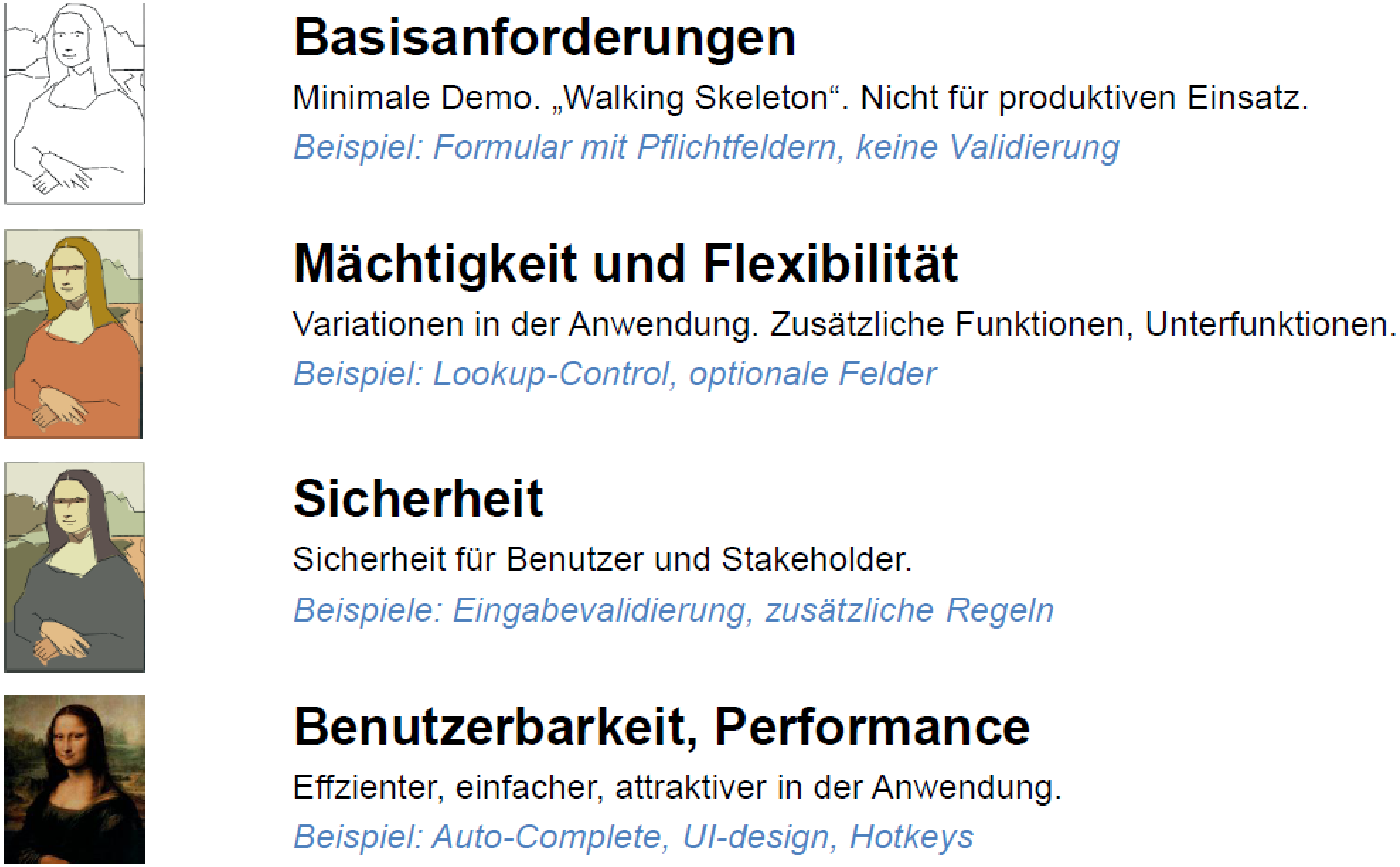
- **Automated regression tests** help by allowing you to quickly detect any defects that may have been introduced when you are changing things. A good rule of thumb is that the size of your unit test code should be the same size as your production code.

- **Continuous integration** keeps a team in sync to avoid painful integration cycles. At the heart of this lies a fully automated build process that can be kicked off automatically whenever any member of the team checks code into the code base.

## Agile Software Development Techniques

- **Refactoring** is a disciplined technique for changing existing software. Refactoring works by using a series of small behavior-preserving transformations to the code base (incremental design).
- **Iterative development** breaks down a project by subsets of functionality. Each iteration produces tested, integrated code that is as close to production quality as possible.
  A common technique with iterations is to use time boxing. This forces an iteration to be a fixed length of time.

# *Entwicklung in kurzen Zyklen?*



**Basisanforderungen**

Minimale Demo. „Walking Skeleton“. Nicht für produktiven Einsatz.

*Beispiel: Formular mit Pflichtfeldern, keine Validierung*

**Mächtigkeit und Flexibilität**

Variationen in der Anwendung. Zusätzliche Funktionen, Unterfunktionen.

*Beispiel: Lookup-Control, optionale Felder*

**Sicherheit**

Sicherheit für Benutzer und Stakeholder.

*Beispiele: Eingabevalidierung, zusätzliche Regeln*

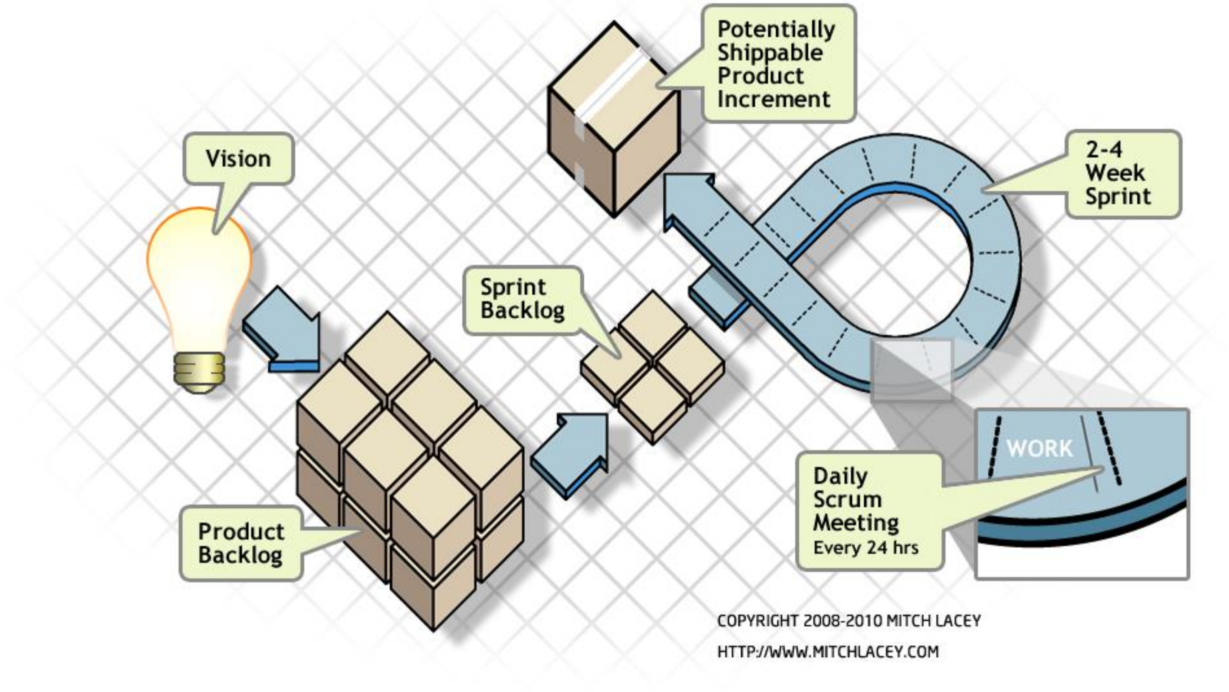
**Benutzerbarkeit, Performance**

Effzienter, einfacher, attraktiver in der Anwendung.

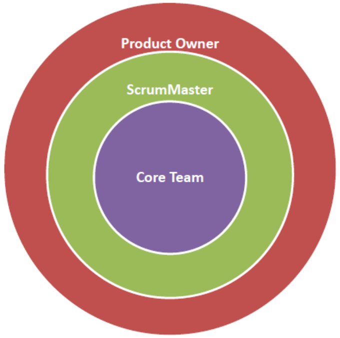
*Beispiel: Auto-Complete, UI-design, Hotkeys*

# *SCRUM*

- Focuses on Project Management
- Based on agile Development-Techniques


Potentially Shippable Product Increment
Vision
2-4 Week Sprint
Sprint Backlog
WORK
Daily Scrum Meeting Every 24 hrs
Product Backlog

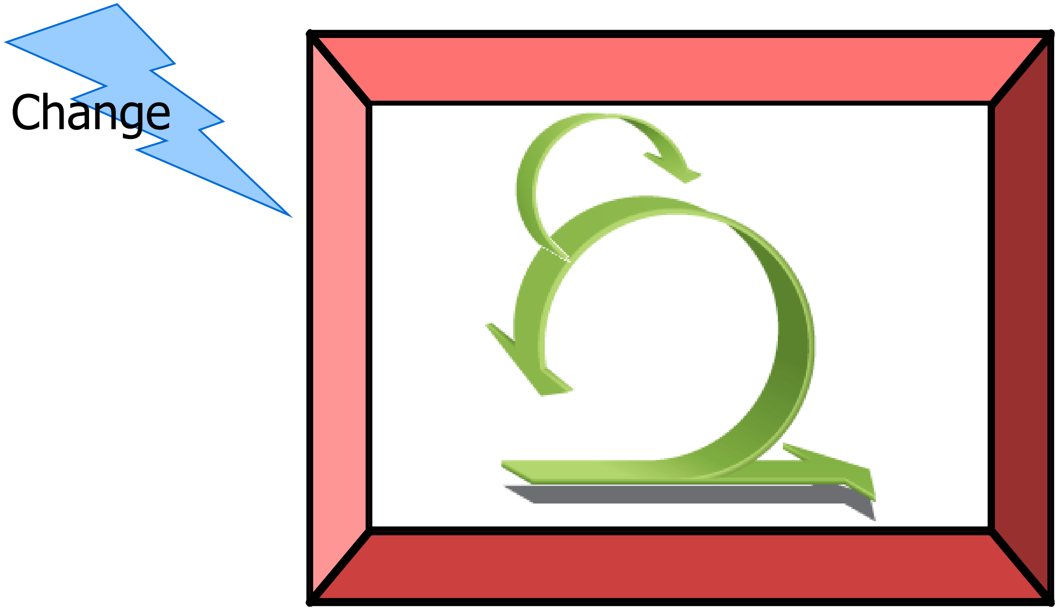
COPYRIGHT 2008-2010 MITCH LACEY
HTTP://WWW.MITCHLACEY.COM



# *SCRUM - Sprints*

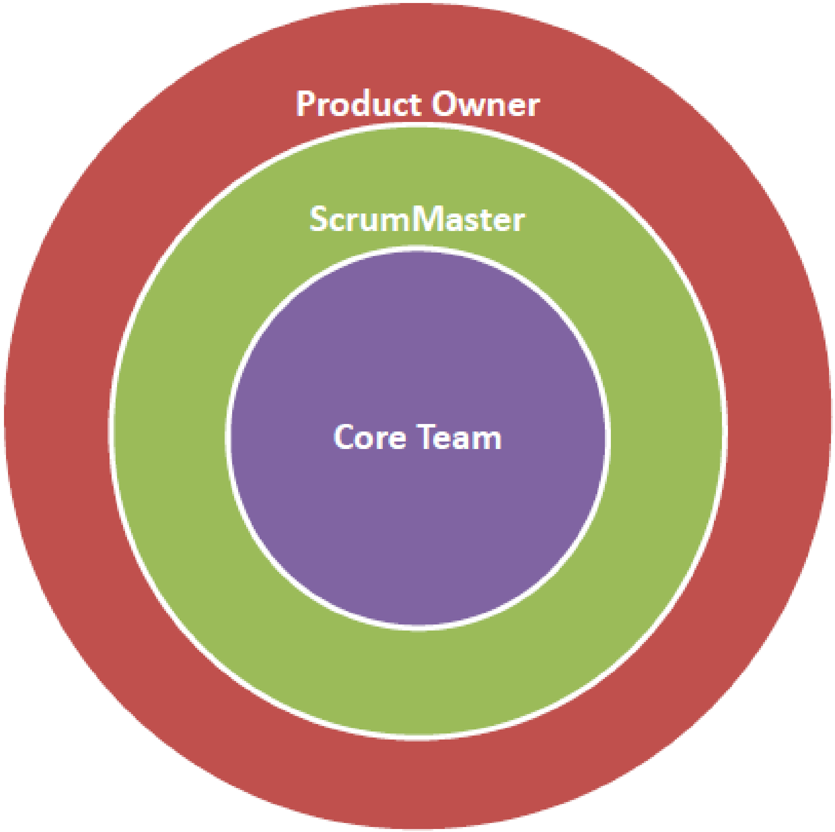
- Scrum projects make progress in a series of “sprints”
  - Analogous to Extreme Programming iterations
- Typical duration is 2–4 weeks or a calendar month at most (Most established: 2 weeks)
- A constant duration leads to a better rhythm
- Product is designed, coded, and tested during the sprint
- **NOGO**: Architecture Sprints etc

# *No changes during a sprint*

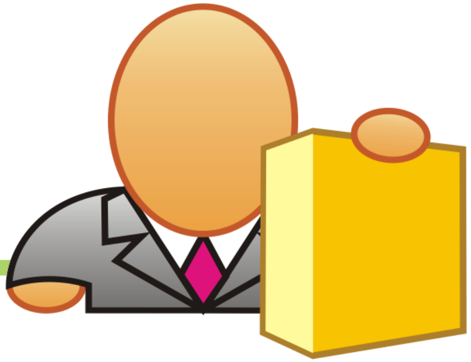


- Plan sprint durations around how long you can commit to keeping change out of the sprint

# *SCRUM Roles*



# *Product owner*

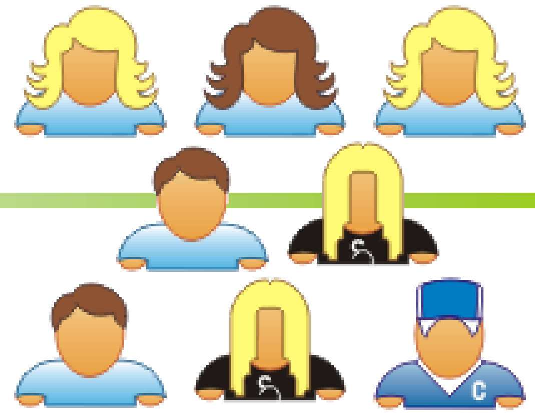


- Define the features of the product
- Decide on release date and content
- Be responsible for the profitability of the product
- Prioritize features according to market value
- Adjust features and priority every iteration, as needed
- Accept or reject work results
- An option: Product Owner Proxy (Inhouse)

# *The ScrumMaster*

- Represents management to the project
- Responsible for enacting Scrum values and practices
- Removes impediments
- Ensure that the team is fully functional and productive
- Enable close cooperation across all roles and functions
- Shield the team from external interferences

# *The Team*



- Typically 5-9 people
- Cross-functional:
  - Programmers, testers, user experience designers, etc.
- Members should be full-time
  - May be exceptions (e.g., database administrator)
- Teams are self-organizing
  - Ideally, no titles but rarely a possibility

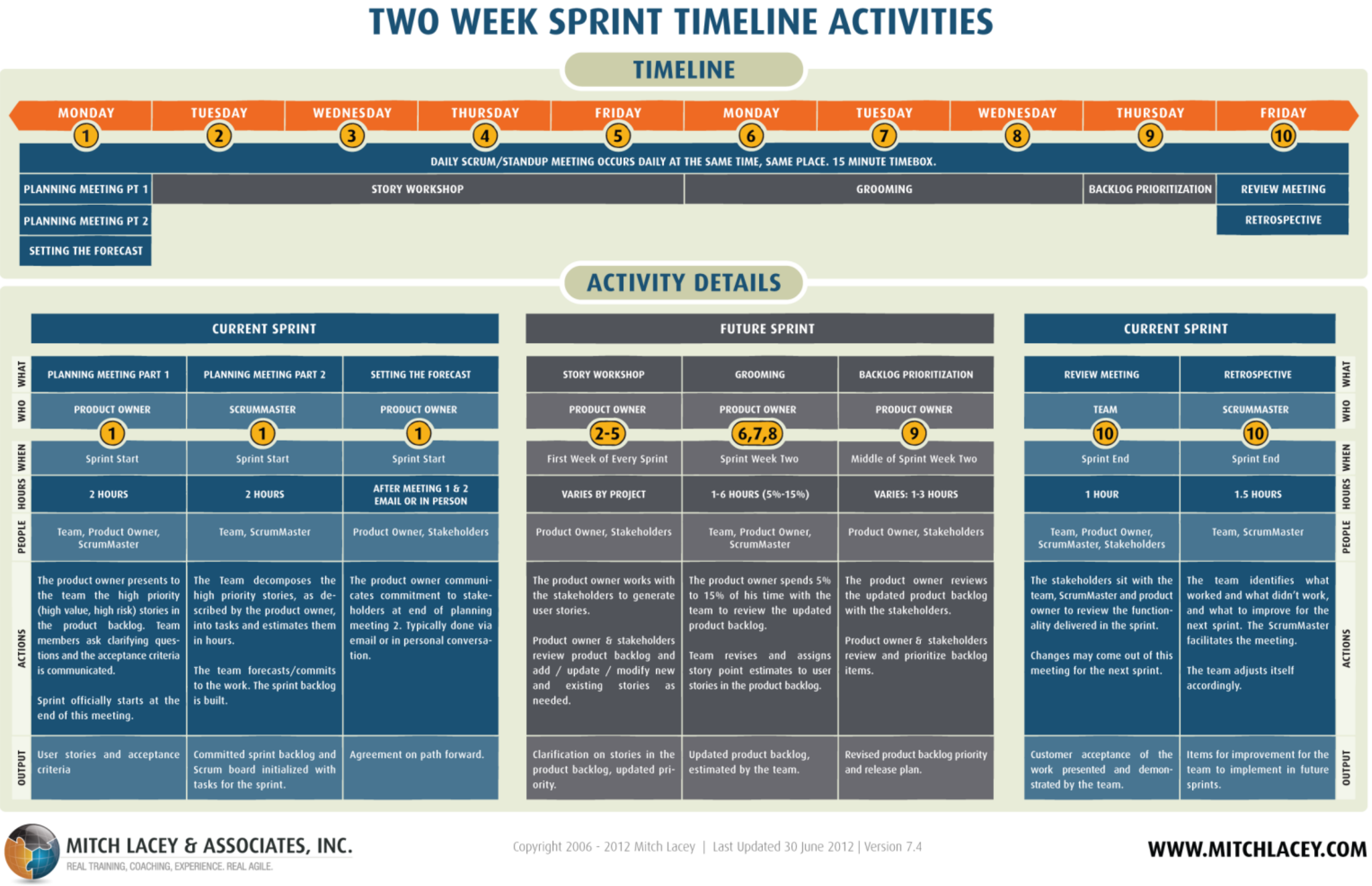
# *Scrum Meetings*

- Sprint Planning
  - Planning, Estimation (by whole Team)
- Daily scrum meeting
  - Stand Up, 15 minutes, not for problem solving
- Sprint Review
  - What was accomplished, demo
- Sprint Retrospective
  - Periodically check what was not working
  - Not at each sprint

# *Scrum Artifacts*

- Product Backlog
  - User Stories
- Sprint Backlog
  - Tasks
- Burn Down Charts
  - Management Reports

# A Sprint – More in Detail



## TWO WEEK SPRINT TIMELINE ACTIVITIES

### TIMELINE

| MONDAY | TUESDAY | WEDNESDAY | THURSDAY | FRIDAY | MONDAY | TUESDAY | WEDNESDAY | THURSDAY | FRIDAY |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| DAILY SCRUM/STANDUP MEETING OCCURS DAILY AT THE SAME TIME, SAME PLACE. 15 MINUTE TIMEBOX. | | | | | | | | | |
| PLANNING MEETING PT 1 | STORY WORKSHOP | | | | GROOMING | | | BACKLOG PRIORITIZATION | REVIEW MEETING |
| PLANNING MEETING PT 2 | | | | | | | | | RETROSPECTIVE |
| SETTING THE FORECAST | | | | | | | | | |

### ACTIVITY DETAILS

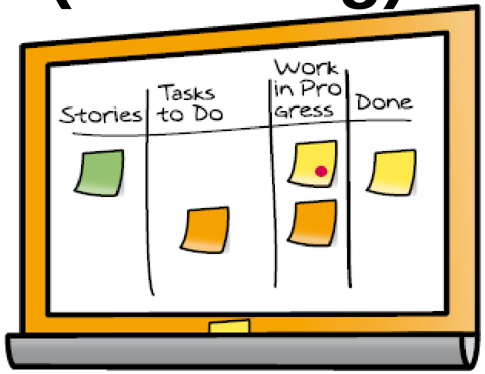
| | CURRENT SPRINT | | | FUTURE SPRINT | | | CURRENT SPRINT | |
|---|---|---|---|---|---|---|---|---|
| WHAT | PLANNING MEETING PART 1 | PLANNING MEETING PART 2 | SETTING THE FORECAST | STORY WORKSHOP | GROOMING | BACKLOG PRIORITIZATION | REVIEW MEETING | RETROSPECTIVE |
| WHO | PRODUCT OWNER | SCRUMMASTER | PRODUCT OWNER | PRODUCT OWNER | PRODUCT OWNER | PRODUCT OWNER | TEAM | SCRUMMASTER |
| | 1 | 1 | 1 | 2-5 | 6,7,8 | 9 | 10 | 10 |
| WHEN | Sprint Start | Sprint Start | Sprint Start | First Week of Every Sprint | Sprint Week Two | Middle of Sprint Week Two | Sprint End | Sprint End |
| HOURS | 2 HOURS | 2 HOURS | AFTER MEETING 1 & 2 EMAIL OR IN PERSON | VARIES BY PROJECT | 1-6 HOURS (5%-15%) | VARIES: 1-3 HOURS | 1 HOUR | 1.5 HOURS |
| PEOPLE | Team, Product Owner, ScrumMaster | Team, ScrumMaster | Product Owner, Stakeholders | Product Owner, Stakeholders | Team, Product Owner, ScrumMaster | Product Owner, Stakeholders | Team, Product Owner, ScrumMaster, Stakeholders | Team, ScrumMaster |
| ACTIONS | The product owner presents to the team the high priority (high value, high risk) stories in the product backlog. Team members ask clarifying questions and the acceptance criteria is communicated.<br>Sprint officially starts at the end of this meeting. | The Team decomposes the high priority stories, as described by the product owner, into tasks and estimates them in hours.<br>The team forecasts/commits to the work. The sprint backlog is built. | The product owner communicates commitment to stakeholders at end of planning meeting 2. Typically done via email or in personal conversation. | The product owner works with the stakeholders to generate user stories.<br>Product owner & stakeholders review product backlog and add / update / modify new and existing stories as needed. | The product owner spends 5% to 15% of his time with the team to review the updated product backlog.<br>Team revises and assigns story point estimates to user stories in the product backlog. | The product owner reviews the updated product backlog with the stakeholders.<br>Product owner & stakeholders review and prioritize backlog items. | The stakeholders sit with the team, ScrumMaster and product owner to review the functionality delivered in the sprint.<br>Changes may come out of this meeting for the next sprint. | The team identifies what worked and what didn't work, and what to improve for the next sprint. The ScrumMaster facilitates the meeting.<br>The team adjusts itself accordingly. |
| OUTPUT | User stories and acceptance criteria | Committed sprint backlog and Scrum board initialized with tasks for the sprint. | Agreement on path forward. | Clarification on stories in the product backlog, updated priority. | Updated product backlog, estimated by the team. | Revised product backlog priority and release plan. | Customer acceptance of the work presented and demonstrated by the team. | Items for improvement for the team to implement in future sprints. |

MITCH LACEY & ASSOCIATES, INC.
REAL TRAINING, COACHING, EXPERIENCE. REAL AGILE.

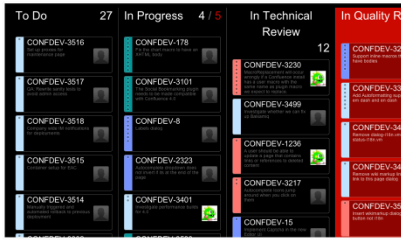
WWW.MITCHLACEY.COM

# *Agile –Tracking / Reporting*

## Taskboard (Tracking)



## Best Friend: Whiteboard



CONFDEV-3516 Set up proxies for maintenance page

CONFDEV-3517 QA: Rewrite sanity tests to avoid admin access

CONFDEV-3518 Company wide IM notifications for deployments

CONFDEV-3515 Container setup for EAC

CONFDEV-3514 Manually triggered and automated rollback to previous deployment

CONFDEV-178 Fix the chart macro to have an XHTML body

CONFDEV-3101 The Social Bookmarking plugin needs to be made compatible with Confluence 4.0

CONFDEV-8 Labels dialog

CONFDEV-2323 Autocomplete dropdown does not invert if its at the end of the page

CONFDEV-3401 Investigate performance builds for 4.0

CONFDEV-3230 MacroReplacement will occur wrongly if a Confluence install has a user macro with the same name as plugin macro we expect to replace.

CONFDEV-3499 Investigate whether we can fix up Balsamiq

CONFDEV-1236 A user should be able to update a page that contains links or references to deleted content

CONFDEV-3217 Autocomplete icons jump around when you click on them

CONFDEV-15 Implement Captcha in the new Editor UI

CONFDEV-32 Support inline macros t have bodies

CONFDEV-33 Add Autoformatting sup em dash and en dash

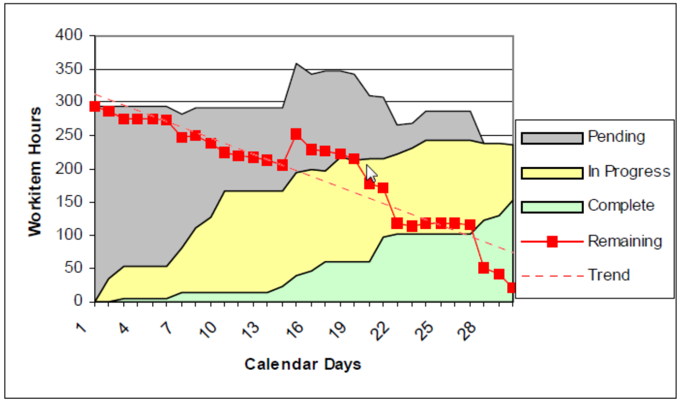
CONFDEV-34 Remove dialog-i18n.vm status-i18n.vm

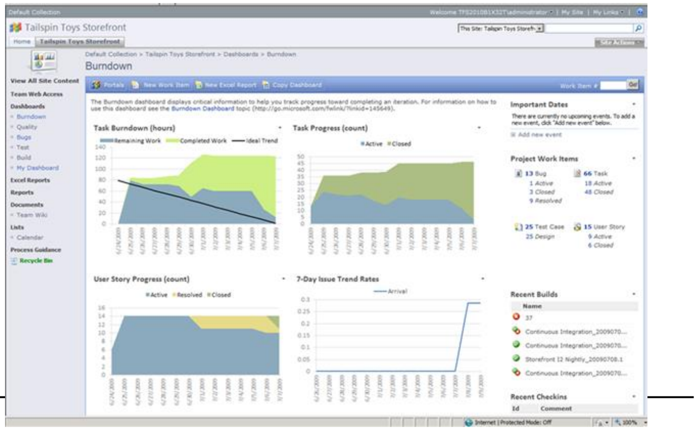
CONFDEV-34 Remove wiki markup lin link to this page dialog

CONFDEV-35 Insert wikimarkup dialog button not i18n

Tool: Atlassian Grasshopper

## Burn Down Chart (Reporting)





The Burndown dashboard displays critical information to help you track progress toward completing an iteration. For information on how to use this dashboard see the Burndown Dashboard topic (http://go.microsoft.com/fwlink/?linkid=145649).

Important Dates

There are currently no upcoming events. To add a new event, click "Add new event" below.

Add new event

Project Work Items

13 Bug: 1 Active, 3 Closed, 9 Resolved

66 Task: 18 Active, 48 Closed

25 Test Case: 25 Design

15 User Story: 9 Active, 6 Closed

Recent Builds

| Name |
|---|
| 37 |
| Continuous Integration_2009070... |
| Continuous Integration_2009070... |
| Storefront I2 Nightly_20090708.1 |
| Continuous Integration_2009070... |

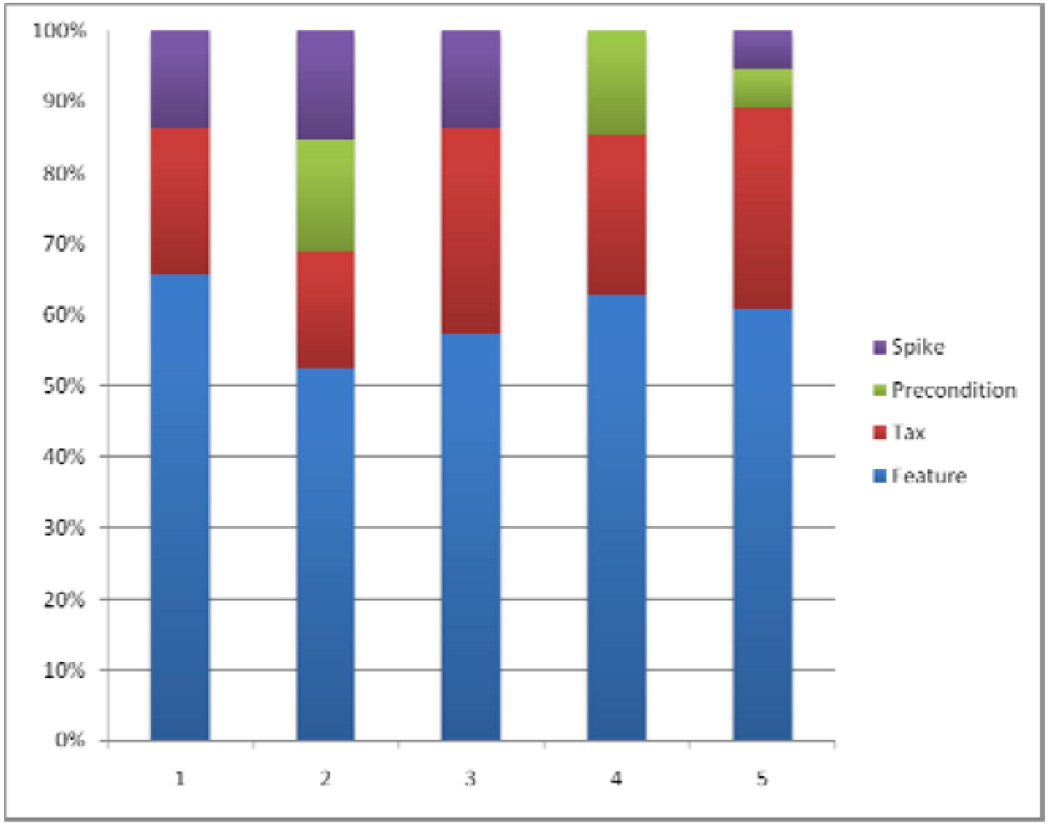
Recent Checkins

| Id | Comment |
|---|---|

Tool: TFS2010 SCRUM Dashboard

- Simple
- Team velocity (Story Points / Features per iteration)
- Pipeline is full (Teammembers take tasks)

# Reporting – More in Detail



**Spike**
- Investigation, Prototyping

**Precondition**
- Setup, CI, Deployment,…

**Tax**
- Cost in Business (e.g. meetings and other things to be fulfilled within your organization)

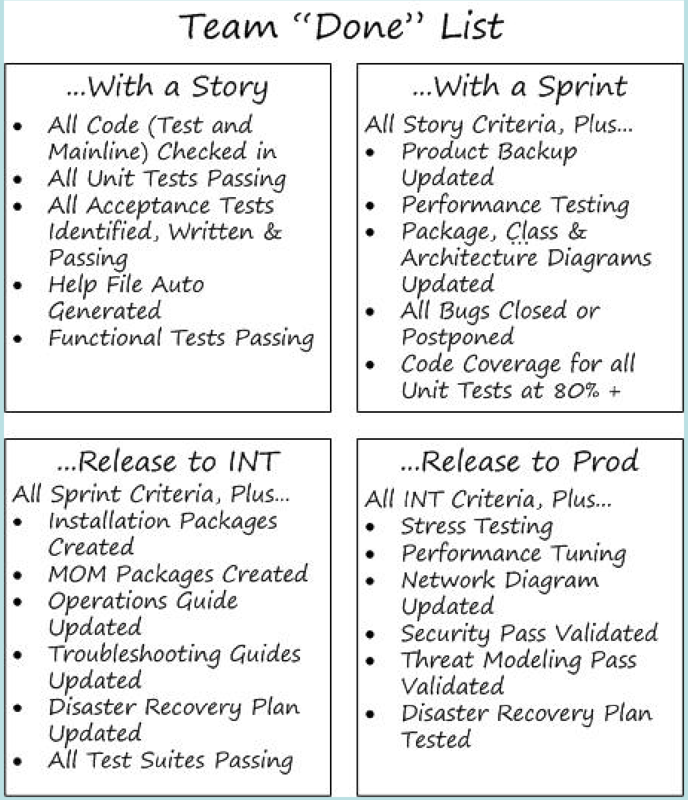
**Feature**
- Customer value driven

© by Mitch Lacey (taken from the „SCRUM for Managers.pdf“, Link: http://mitch.lacey.com)

# *Digression: When are you done?*

- Done is **NOT** after checking in your code!

- A sample „Done“ List taken from SCRUM
  (adopt it if necessary)



Team "Done" List

...With a Story

- All Code (Test and Mainline) Checked in
- All Unit Tests Passing
- All Acceptance Tests Identified, Written & Passing
- Help File Auto Generated
- Functional Tests Passing

...With a Sprint

All Story Criteria, Plus...

- Product Backup Updated
- Performance Testing
- Package, Class & Architecture Diagrams Updated
- All Bugs Closed or Postponed
- Code Coverage for all Unit Tests at 80% +

...Release to INT

All Sprint Criteria, Plus...

- Installation Packages Created
- MOM Packages Created
- Operations Guide Updated
- Troubleshooting Guides Updated
- Disaster Recovery Plan Updated
- All Test Suites Passing

...Release to Prod

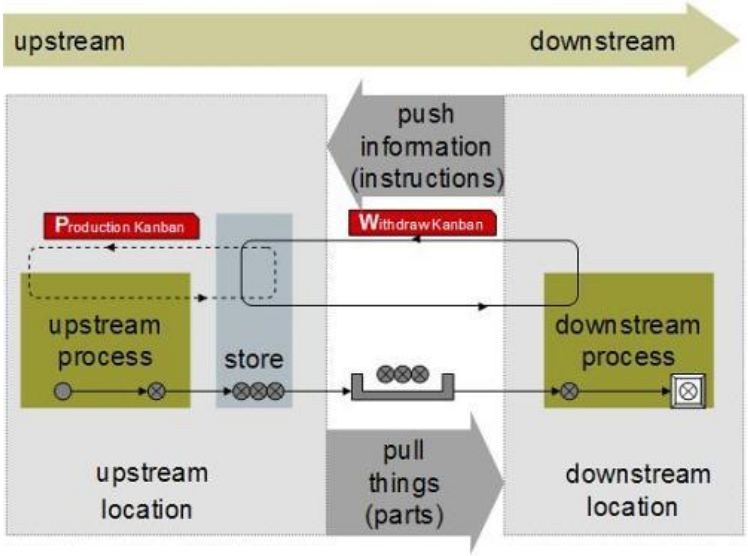
All INT Criteria, Plus...

- Stress Testing
- Performance Tuning
- Network Diagram Updated
- Security Pass Validated
- Threat Modeling Pass Validated
- Disaster Recovery Plan Tested

© by Mitch Lacey (Link: http://mitch.lacey.com)

# *Kanban – the principle*

## Toyota Production System (Principle)



The downstream process pulls parts when needed, and provides production instructions, by exchanging Kanbans at the "store"

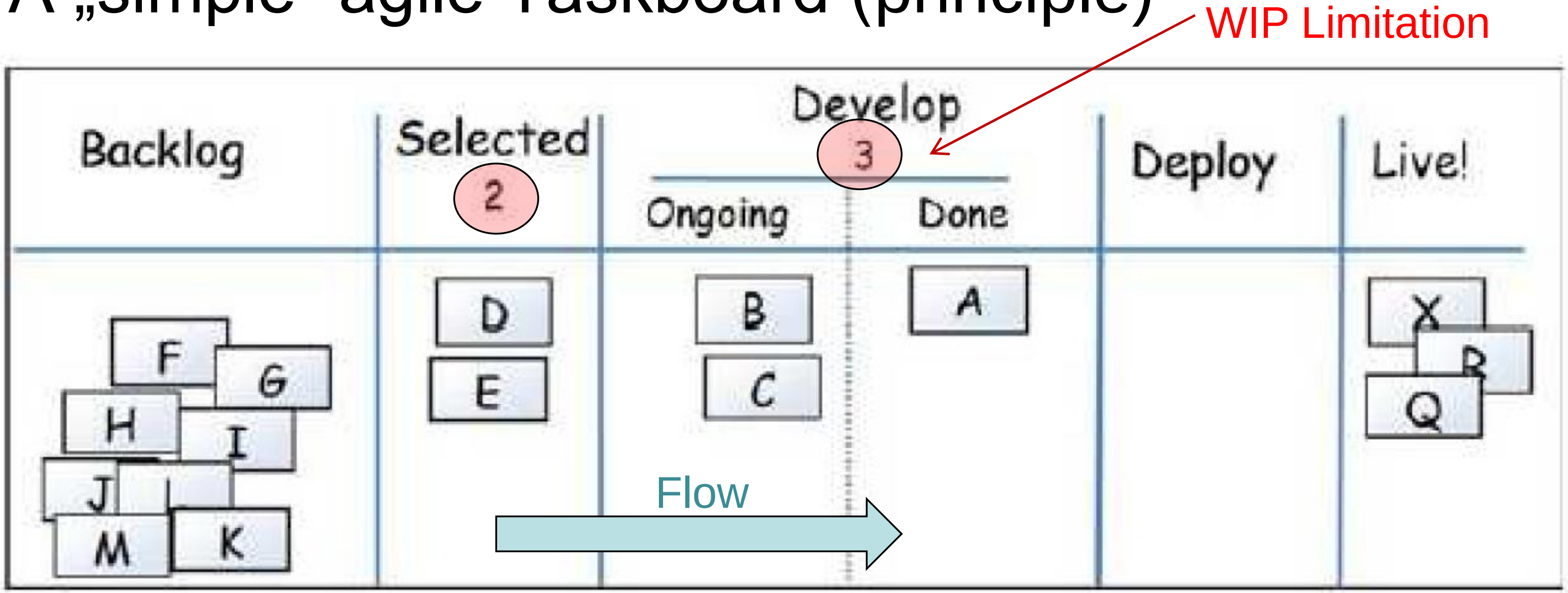
- Upstream produces parts to store
- Downstream pulls parts from store and finishes them
- Downstream pushes „needs“

**Key ideas**

- Flow principle (store is buffer)
- Limit Work in progress
- Optimize cycle time

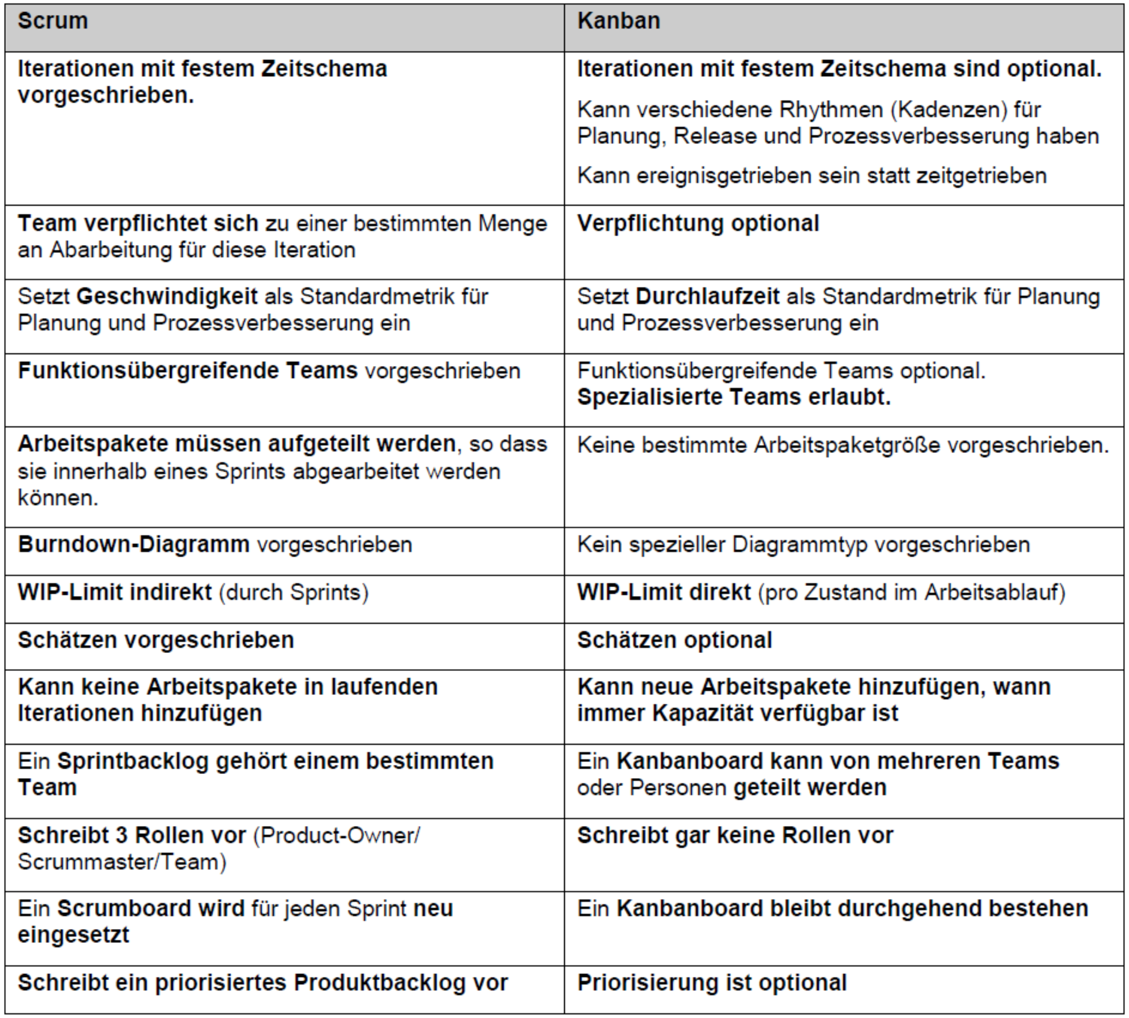
# *Kanban in a SWE-Process*

## A „simple“ agile Taskboard (principle)



Kanban does not define the workflow itself
(even waterfall would be possible for each item)

# Scrum vs. Kanban - Differences

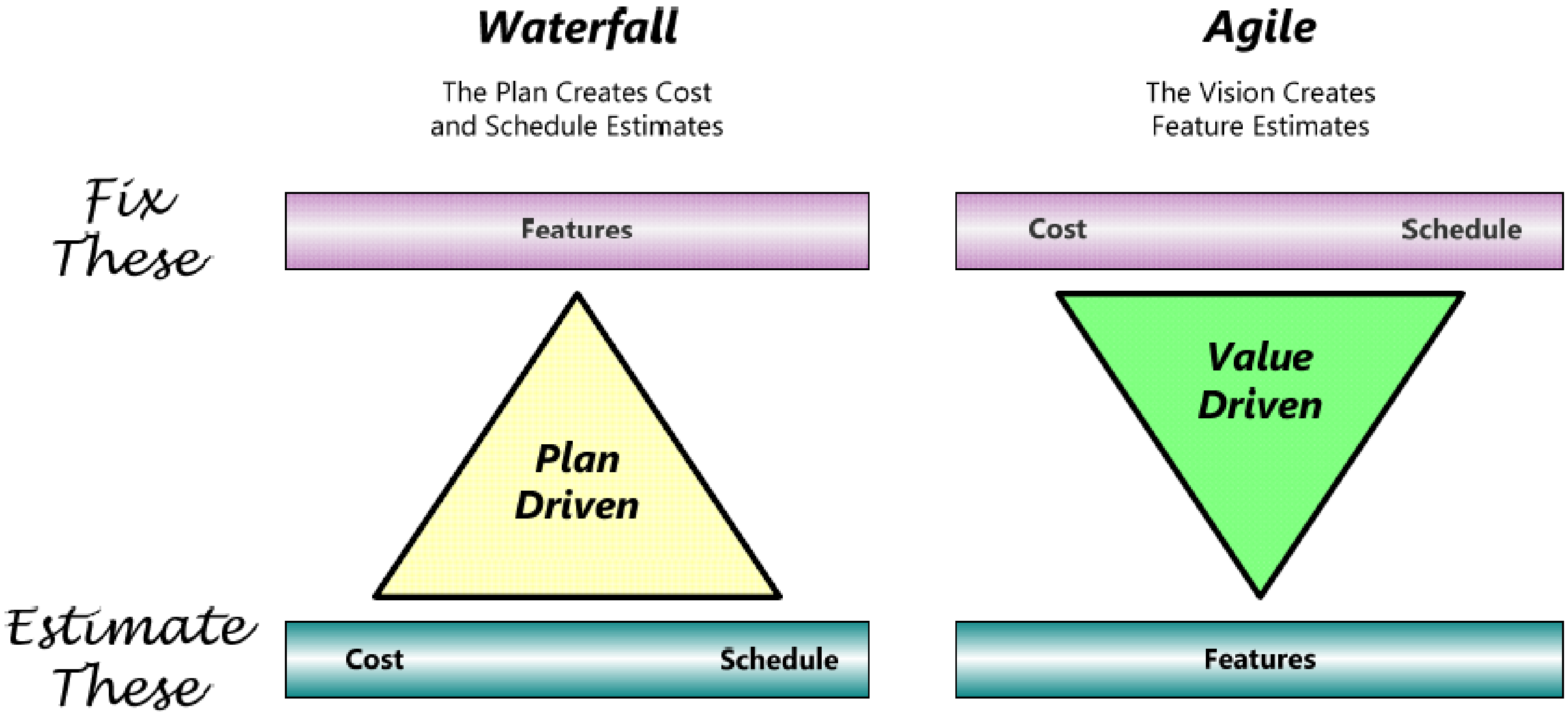


| Scrum | Kanban |
|---|---|
| **Iterationen mit festem Zeitschema vorgeschrieben.** | **Iterationen mit festem Zeitschema sind optional.**<br>Kann verschiedene Rhythmen (Kadenzen) für Planung, Release und Prozessverbesserung haben<br>Kann ereignisgetrieben sein statt zeitgetrieben |
| **Team verpflichtet sich** zu einer bestimmten Menge an Abarbeitung für diese Iteration | **Verpflichtung optional** |
| Setzt **Geschwindigkeit** als Standardmetrik für Planung und Prozessverbesserung ein | Setzt **Durchlaufzeit** als Standardmetrik für Planung und Prozessverbesserung ein |
| **Funktionsübergreifende Teams** vorgeschrieben | Funktionsübergreifende Teams optional. **Spezialisierte Teams erlaubt.** |
| **Arbeitspakete müssen aufgeteilt werden**, so dass sie innerhalb eines Sprints abgearbeitet werden können. | Keine bestimmte Arbeitspaketgröße vorgeschrieben. |
| **Burndown-Diagramm** vorgeschrieben | Kein spezieller Diagrammtyp vorgeschrieben |
| **WIP-Limit indirekt** (durch Sprints) | **WIP-Limit direkt** (pro Zustand im Arbeitsablauf) |
| **Schätzen vorgeschrieben** | **Schätzen optional** |
| **Kann keine Arbeitspakete in laufenden Iterationen hinzufügen** | **Kann neue Arbeitspakete hinzufügen, wann immer Kapazität verfügbar ist** |
| Ein **Sprintbacklog gehört einem bestimmten Team** | Ein **Kanbanboard kann von mehreren Teams** oder Personen **geteilt werden** |
| **Schreibt 3 Rollen vor** (Product-Owner/ Scrummaster/Team) | **Schreibt gar keine Rollen vor** |
| Ein **Scrumboard wird** für jeden Sprint **neu eingesetzt** | Ein **Kanbanboard bleibt durchgehend bestehen** |
| **Schreibt ein priorisiertes Produktbacklog vor** | **Priorisierung ist optional** |

**Quelle:** http://www.infoq.com/resource/news/2010/01/kanban-scrum-minibook/en/resources/KanbanAndScrum-German.pdf

# *Agile and Contracts*

# Agile vs. Plan Driven



© by Mitch Lacey (taken from the „SCRUM for Managers.pdf“, Link: http://mitch.lacey.com)

# *We need to rethink contracts*

Agile Contracts must **contain** …

- Costs
- Timeline
- Vision and Feature Estimations

… but must also **permit changes** Change

Fixed everything does not work with Agile

# *Agile vs. Fixed Price Projects(1)*

- Fixed price Contracts typically define
  - Fixed set of Features (Scope)
  - Fixed schedule
  - Scope change: is Change Request (extra costs)
- Agile paradigm
  - Scope change is expected, features may change (no extra costs)
  - Fixed schedule
  - budget limitation (i.e. time & material like contract)
  - But: does not fit with classical Fixed price contracts
- Issue: Customers are used to fixed price contracts

# *Agile vs. Fixed Price Projects(2)*

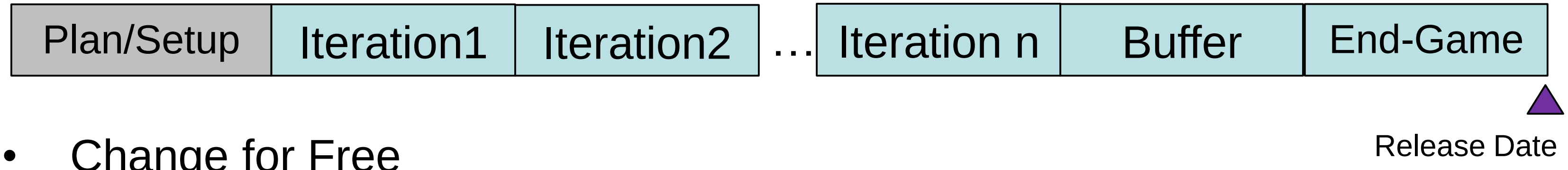
Possible Options

- Convince customer that agile works better (time & material with budget limitation)
  - Not always easy
    - Time & material issue: No warranty, no risk at supplier side
    - Comparison to other suppliers (“price pressure” does not work that easily)
    - Often does not fit to customers policies: Established RfP/RfQ processes
- Fixed price with change volume and feature changes at no cost
  - Define change volumes in contract (feature changes at no extra costs)
  - Define only a subset of fixed features with options to change
  - Customer needs to trust you!
- Develop agile internally in your organization
  - Some people say: don’t to it, I disagree
  - Why: Agile is a mindset and evolves your organization

# *Agile Fixed Price Contract Model*

## According to M Lacey / J Sutherland

- Product Backlog (with must have and optional features)
- Time Schedule (Iterations with Buffer)



- Change for Free
  - Define fixed price contract based on
    - # of iterations (internal: include “buffer iterations”)
    - Team velocity (Story  Points / Features per Iteration)
    - Changes are included if other features are dropped
- Money for Nothing
  - Customer checks whether features have no more ROI
  - Customer aborts project at 20% of remaining contract volume at the end of an iteration
  - If team is faster, team gets a percentage of remaining contract volume as award for being faster

# *Summary*

- Agile Methods
  - – Focus on Teamwork
  - – Development in small Iterations (running software)
  - – Are based on fast feedback
  - – Are a Mind Change
  - – Are Effective and Adaptive
  - – Even aspects of Agile help!
  - – “Issue” with Customers: Contracts

# *References*

- *Martin Fowler*
  **“The New Methodology”**
  http://martinfowler.com/articles, 13 Dec. 2005

- SCRUM (used in slide deck)
  Mike Cohn, Mountain Goat Software
  Mitch Lacey, MitchLacey.com

- Andreas Opelt, Boris Gloger, Wolfgang Pfarl, Ralf Mittermayr
  **“Der Agile Festpreis”**, Sept. 2012

# *Practical Software Engineering*

*User Stories*

FH JOANNEUM

Software Design

WS 2018

# *Agenda*

- Requirementserhebung durch User Stories
- Schätzen von User Stories

# *User Stories*

## Writing Stories

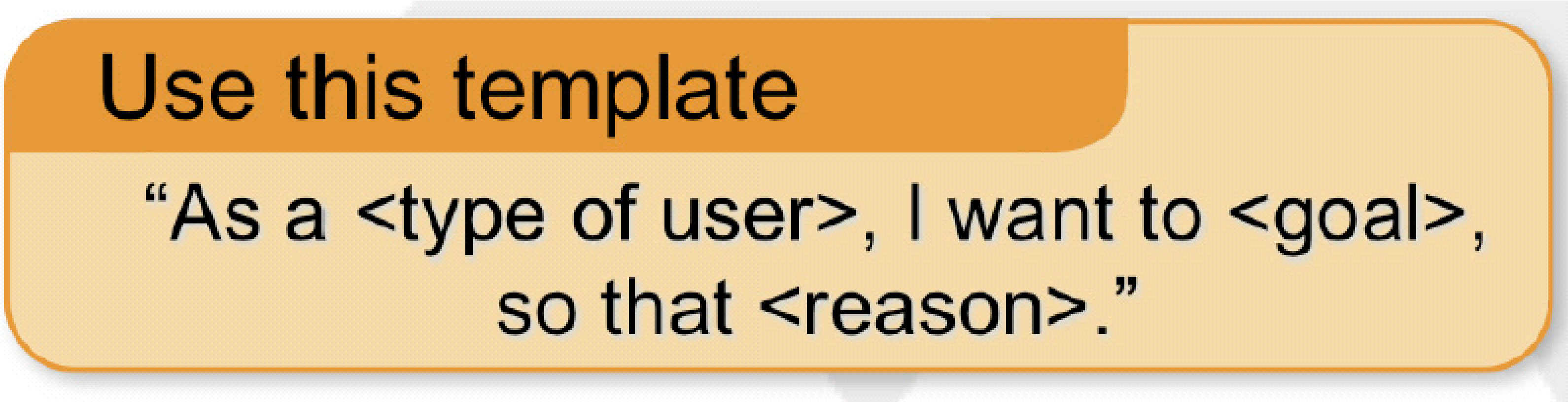
> A **user story** (also called feature) is a chunk of functionality that is of value to the customer.

- Stories must be understandable to the customer.
- A user story is nothing more than an agreement that the customer and developers will talk together about a feature.
- Stories need to be of a size that you can build a few of them in each iteration.
- Stories should be independent of each other.
- Each story must be testable.

# *User Stories*

- Typischerweise 1 oder 2 Sätze

- Etabliertes Template



Use this template

"As a <type of user>, I want to <goal>, so that <reason>."

- Tipp für Requirements Analyse: Verwenden Sie aktive Sprache

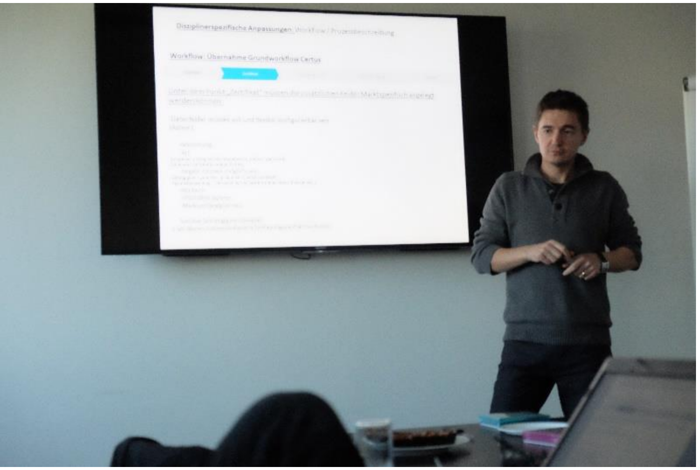
# *Erhebung von User Stories*

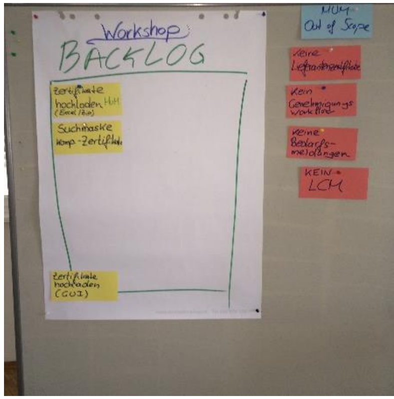
Oft verwendete Techniken

- Stakeholder Interviews
- Fragebögen
- Beobachtung
- „In die Lehre gehen“
- Story Writing Workshops (agil)

# *Impressions of a user story writing workshop*

Explanations, questions, discussions…




MUM
Out of Scope
Workshop
BACKLOG
Zertifikate hochladen HM (Excel/Bin)
Suchmaske komp-Zertifikate
Zertifikate hochladen (GUI)
keine Lieferantenzertifikate
Kein Genehmigungs workflow
keine Bedarfs-meldungen
KEIN LCM

… decisions & documentation

# *Typische Fragen in Story Workshops*

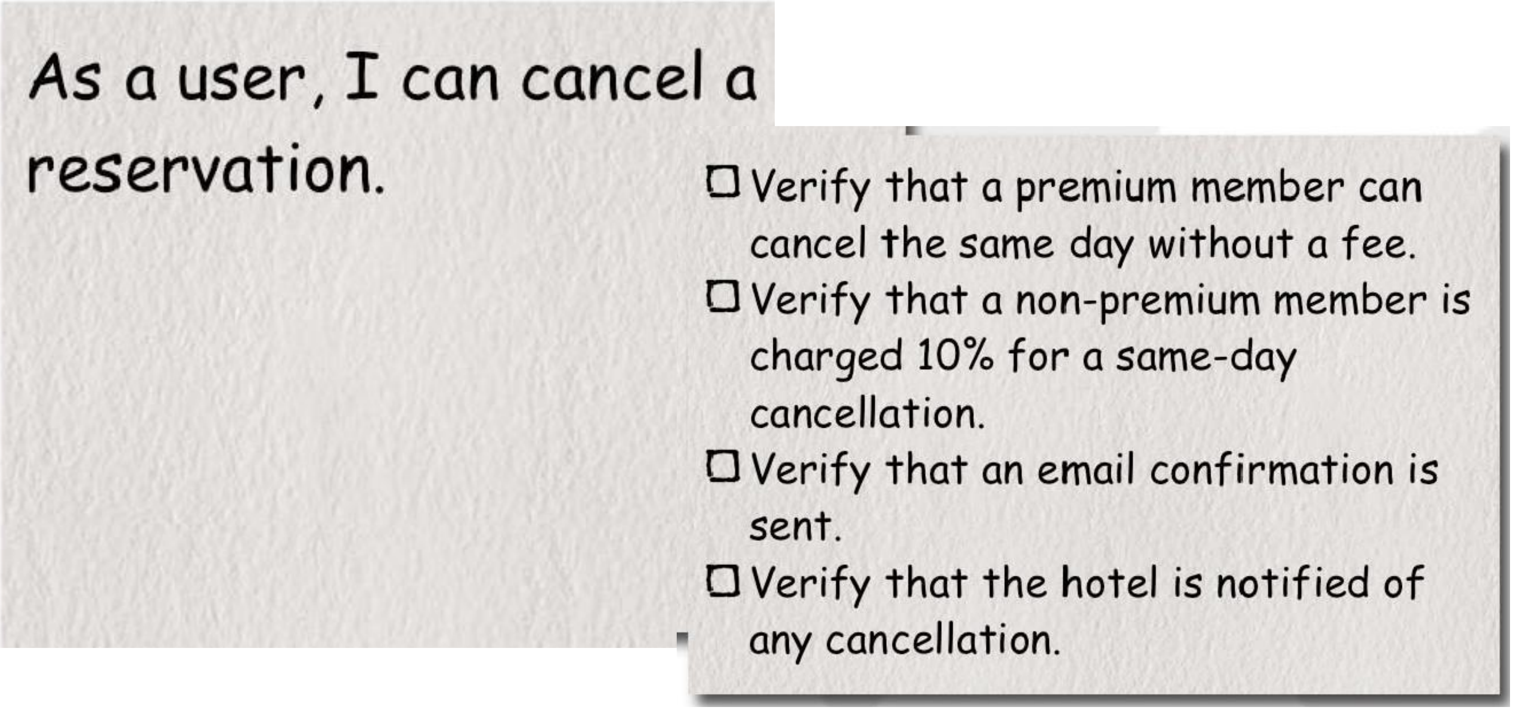
- **Ask about processes**
  - What goal do your want to achieve?
  - Who does what?
  - What are typical KPIs?
  - When does… start /end?
  - If we did …, what would happen?
- **Ask about specific details**
  - How exactly do you …?
  - What do you want to achieve?
  - Why do you that?
  - What do you do in case x?
  - What needs to happen next?
- **Ask about input and output information**
  - Who will deliver the inputs for the feature?
  - Who will receive the outputs of the feature?
  - What needs to be checked / validated?
- **Ask if they have alternative ideas**
  - What if …?
  - Could it be done also …?
- **Ask for other sources of information**
  - Do you have examples/ sample data?
  - Ask the experts to draw diagrams / flow charts

# *Wie kommt man zu Akzeptanzkriterien*

- Verfeinern der User Stories
- Erarbeiten von Beispielen

# User Stories - Akzeptanzkriterien

- Finden von Akzeptanzkriterien durch Detaillieren von User Stories



As a user, I can cancel a reservation.

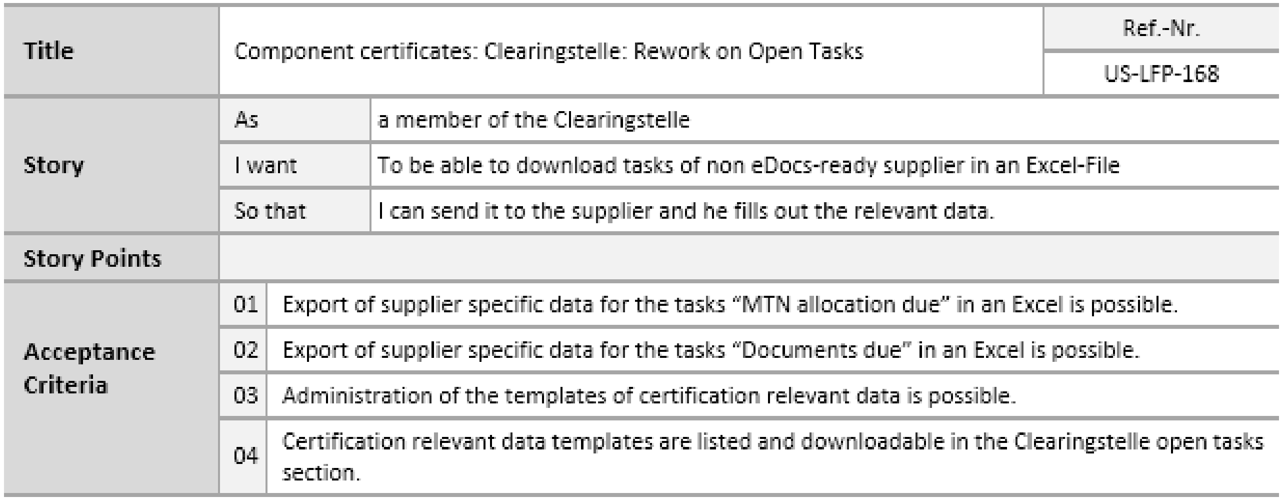
□ Verify that a premium member can cancel the same day without a fee.

□ Verify that a non-premium member is charged 10% for a same-day cancellation.

□ Verify that an email confirmation is sent.

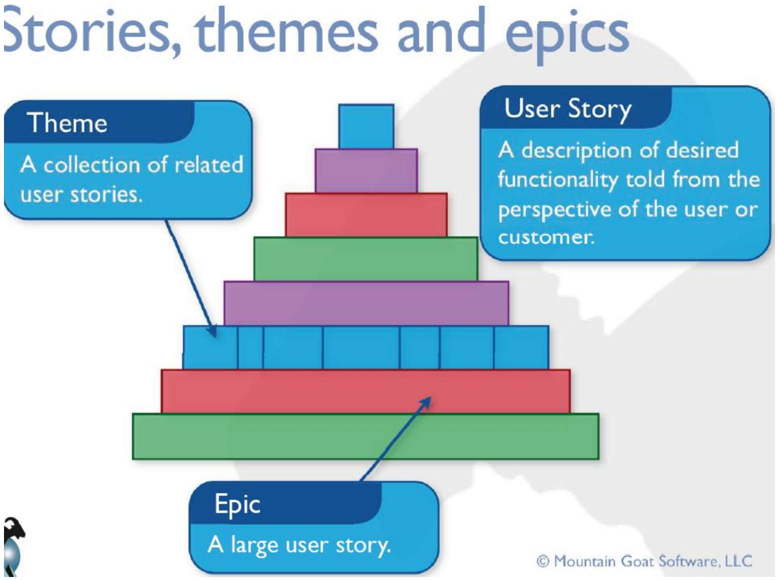
□ Verify that the hotel is notified of any cancellation.

# Ein Beispiel aus der Praxis



| | | | |
|---|---|---|---|
| **Title** | Component certificates: Clearingstelle: Rework on Open Tasks | | Ref.-Nr. |
| | | | US-LFP-168 |
| **Story** | As | a member of the Clearingstelle | |
| | I want | To be able to download tasks of non eDocs-ready supplier in an Excel-File | |
| | So that | I can send it to the supplier and he fills out the relevant data. | |
| **Story Points** | | | |
| **Acceptance Criteria** | 01 | Export of supplier specific data for the tasks “MTN allocation due” in an Excel is possible. | |
| | 02 | Export of supplier specific data for the tasks “Documents due” in an Excel is possible. | |
| | 03 | Administration of the templates of certification relevant data is possible. | |
| | 04 | Certification relevant data templates are listed and downloadable in the Clearingstelle open tasks section. | |

# *Wenn Stories zu groß sind…*



Stories, themes and epics

Theme
A collection of related user stories.
User Story
A description of desired functionality told from the perspective of the user or customer.
Epic
A large user story.

© Mountain Goat Software, LLC

Usually start with Epics

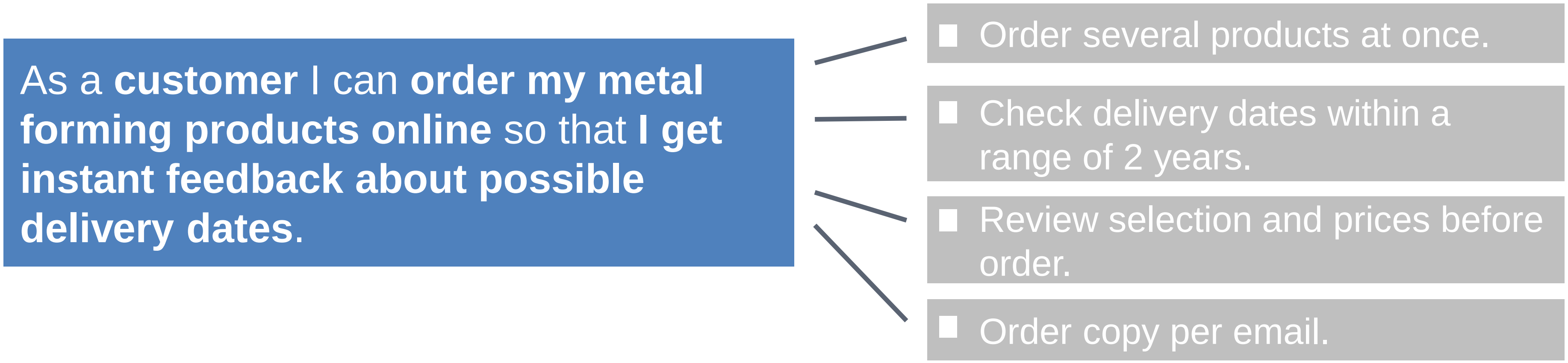
Two options for decomposing

- Create sub story of story
- Create story of acceptance criteria

Typically

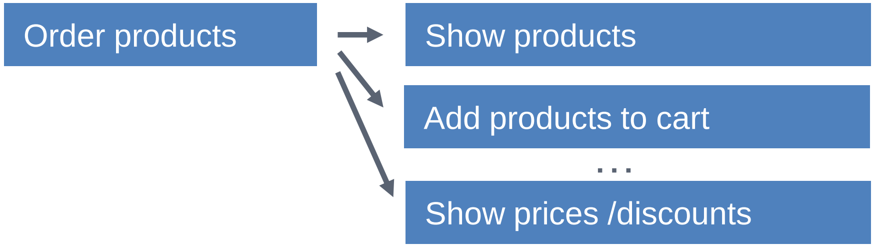
- 5-10 Themes per epic
- 3-10 User Stories per theme (more if needed)

# *Agile requirements engineering with User Stories*



As a **customer** I can **order my metal forming products online** so that **I get instant feedback about possible delivery dates**.

- Order several products at once.
- Check delivery dates within a range of 2 years.
- Review selection and prices before order.
- Order copy per email.



Order products

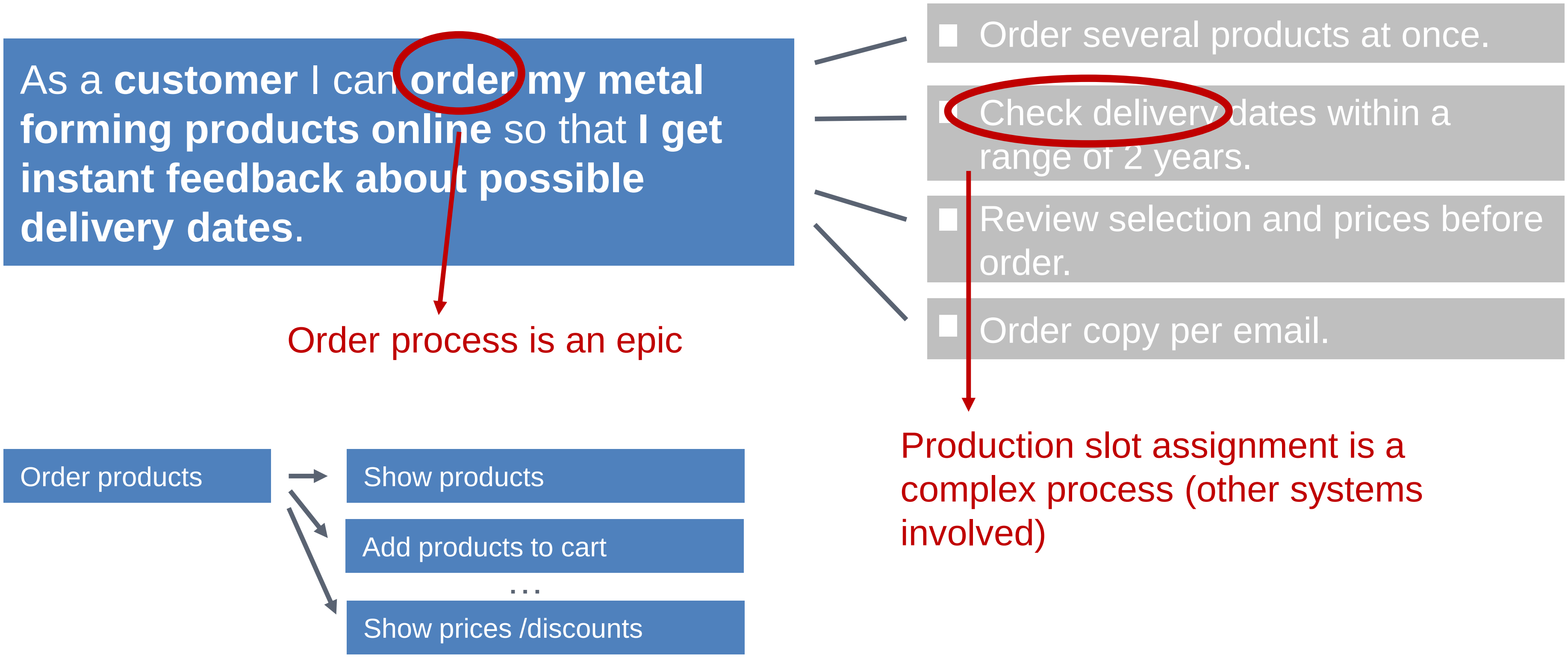
Show products

Add products to cart

…

Show prices /discounts

# *Agile requirements engineering with User Stories*



As a **customer** I can **order my metal forming products online** so that **I get instant feedback about possible delivery dates**.

- □ Order several products at once.
- □ Check delivery dates within a range of 2 years.
- □ Review selection and prices before order.
- □ Order copy per email.

Order process is an epic

Production slot assignment is a complex process (other systems involved)

Order products

Show products

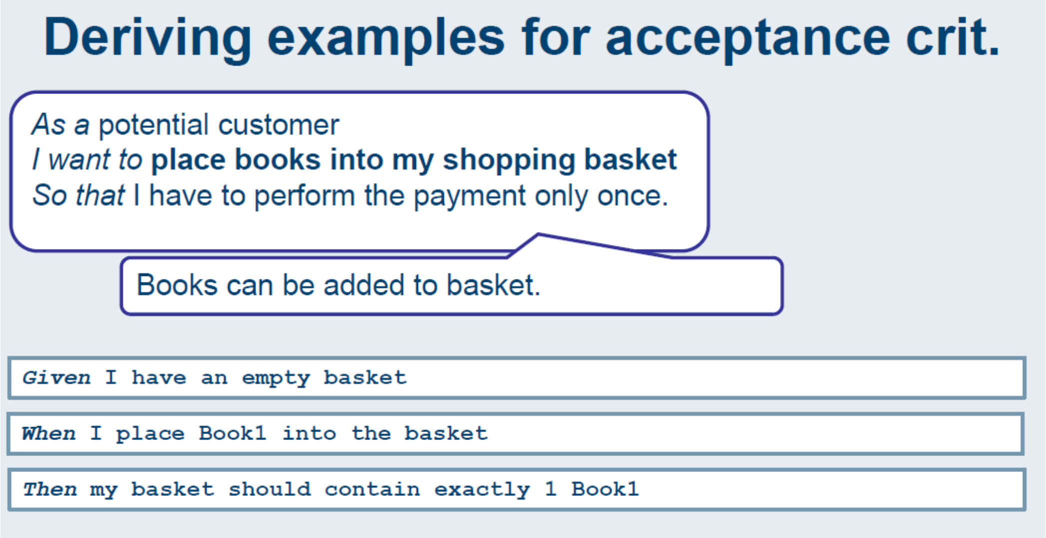
Add products to cart

…

Show prices /discounts

# *Formalisierte Akzeptanzkriterien(1)*

- Ein Ansatz aus “Behaviour Driven Development“



## Deriving examples for acceptance crit.

*As a* potential customer
*I want to* **place books into my shopping basket**
*So that* I have to perform the payment only once.

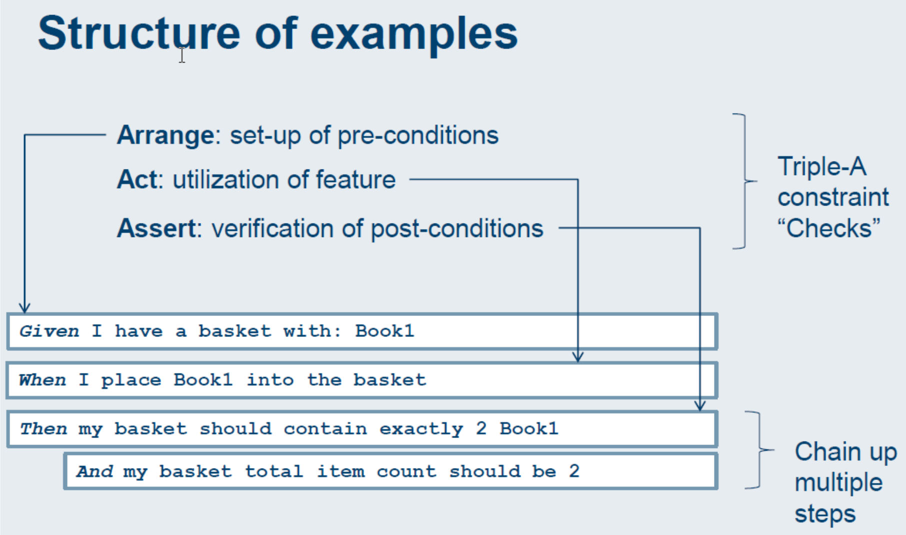
Books can be added to basket.

```
Given I have an empty basket
When I place Book1 into the basket
Then my basket should contain exactly 1 Book1
```

- Hintergrund
  - Specification by Examples
  - Solche Akzeptanzkriterien lassen sich in Code übersetzen
  - Akzeptanztests sind die lebendige, aktuelle Spezifikation

# Formalisierte Akzeptanzkriterien(2)

- Pattern aus BDD



## Structure of examples

**Arrange**: set-up of pre-conditions

**Act**: utilization of feature

**Assert**: verification of post-conditions

Triple-A constraint "Checks"

```
Given I have a basket with: Book1
When I place Book1 into the basket
Then my basket should contain exactly 2 Book1
    And my basket total item count should be 2
```

Chain up multiple steps

- BDD Tools übersetzen Akzeptanzkriterien in Unit-Tests
- Tools
  - Java: JBehave
  - .NET: Specflow

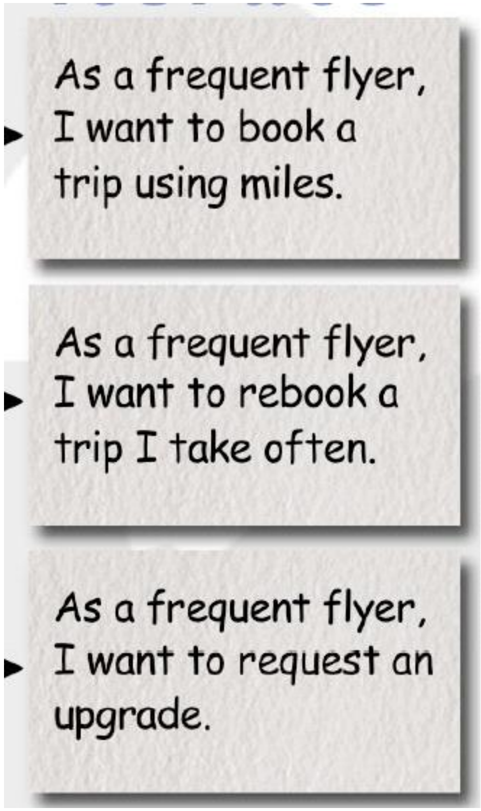
# *User Stories vs. Use Cases (1)*

- User Stories (agil)
  - Beschreiben Wünsche der Kunden (Stakeholder) sehr informal
  - Fokus liegt im Gesprochenen („Versprechen drüber zu reden“)
  - Sind abstrakter und können mehrer Use Cases beinhalten
  - Entsprechen eher High Level Requirements aus der OOA
  - Typischerweise vom Kunden geschrieben
- Use Cases („Klassisch“ aus OOA)
  - Beschreiben wie System das gewünschte Verhalten erfüllt
  - Beschreiben komplette Interaktionen zwischen User und System
  - Sehr formal
  - Fokus auf korrekt niedergeschriebener Beschreibung
  - Vom Berater beschrieben

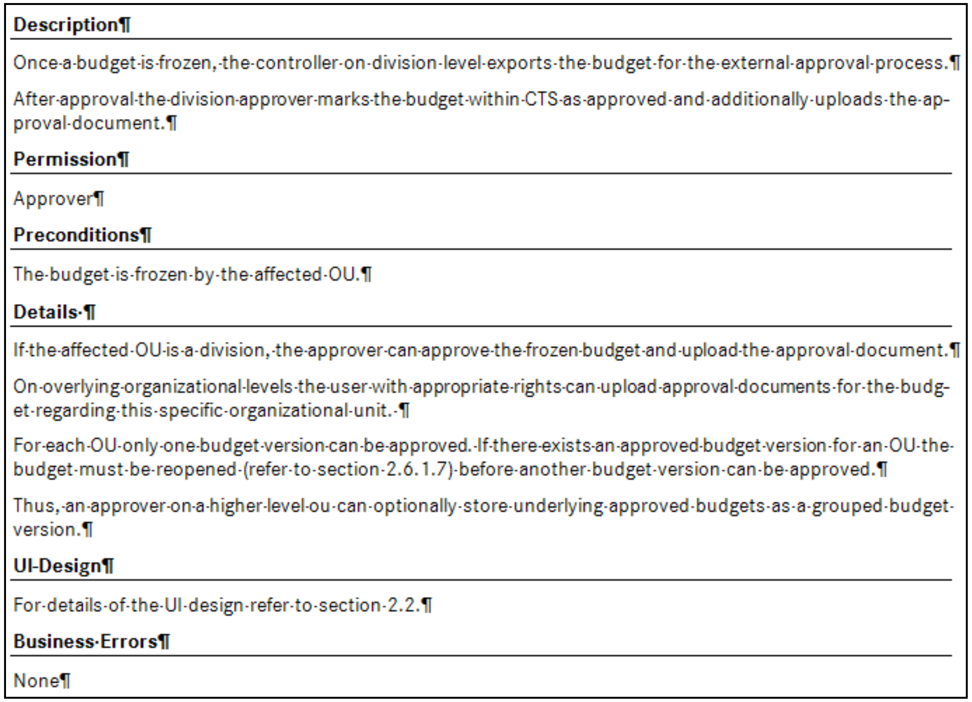
# User Stories vs. Use Cases (2)

- Examples

My experience: Customers do most often not read that much text



- As a frequent flyer, I want to book a trip using miles.
- As a frequent flyer, I want to rebook a trip I take often.
- As a frequent flyer, I want to request an upgrade.



**Description¶**

Once·a·budget·is·frozen,·the·controller·on·division·level·exports·the·budget·for·the·external·approval·process.¶

After·approval·the·division·approver·marks·the·budget·within·CTS·as·approved·and·additionally·uploads·the·approval·document.¶

**Permission¶**

Approver¶

**Preconditions¶**

The·budget·is·frozen·by·the·affected·OU.¶

**Details·¶**

If·the·affected·OU·is·a·division,·the·approver·can·approve·the·frozen·budget·and·upload·the·approval·document.¶

On·overlying·organizational·levels·the·user·with·appropriate·rights·can·upload·approval·documents·for·the·budget·regarding·this·specific·organizational·unit.·¶

For·each·OU·only·one·budget·version·can·be·approved.·If·there·exists·an·approved·budget·version·for·an·OU·the·budget·must·be·reopened·(refer·to·section·2.6.1.7)·before·another·budget·version·can·be·approved.¶

Thus,·an·approver·on·a·higher·level·ou·can·optionally·store·underlying·approved·budgets·as·a·grouped·budget·version.¶

**UI-Design¶**

For·details·of·the·UI·design·refer·to·section·2.2.¶

**Business·Errors¶**

None¶

# *User Stories - Priorisierung*

- Ist Aufgabe des Kunden

- Immanente Fragen
  - Was muss zuerst gemacht werden, was später?
  - Was kann eventuell weggelassen werden?

- Eine Methode:
  - **Prio1**: Muss in dieser Iteration
  - **Prio2**: Kann in dieser Iteration, Muss in nächster
  - **Prio3**: Kommt in einer Iteration, eventuell auch nicht

- Scrum/Agil: Prio 1 bis n (absolut)

# *User Stories und User Interfaces*

- Literatur: Möglichst spät integrieren

  **Meine Erfahrung**: Sehr oft können Business Stakeholder sich ohne Masken nicht auf Details festlegen

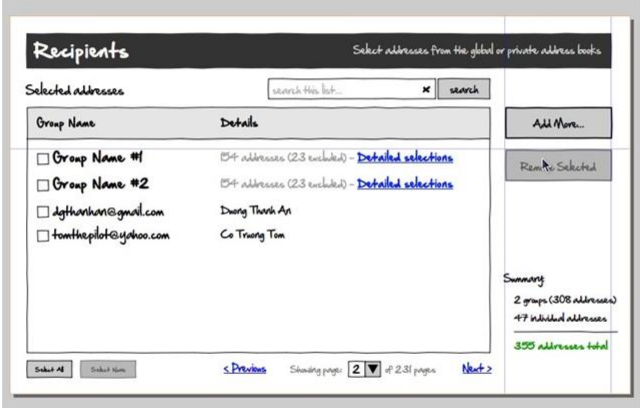
  **Abhilfe**: Einfache Prototypen

- Typische Vorgehensweise
  - 1) User Stories erstellen
  - 2) Priorisierung
  - 3) Stories verfeinern
  - 4) Prototyp erarbeiten (User Interface) für relevante User Stories
  - 5) Stakeholder Feedback einholen und einarbeiten
  - 6) Umsetzung starten

# *User Interface Design*

- Tipp: Keine High End GUIs in Requirements Phase
- Warum?
  - Stakeholder verlieren sich in GUI
  - Stakeholder vergessen Features

- Tipp: Sketching Tools oder Papier Prototypen
  - Bsp: Open Source Tool: Pencil



Recipients

Select addresses from the global or private address books

Selected addresses

| Group Name | Details |
|---|---|
| □ Group Name #1 | 154 addresses (23 excluded) – Detailed selections |
| □ Group Name #2 | 154 addresses (23 excluded) – Detailed selections |
| □ dgthanhan@gmail.com | Duong Thanh An |
| □ tomthepilot@yahoo.com | Co Truong Tom |

Summary

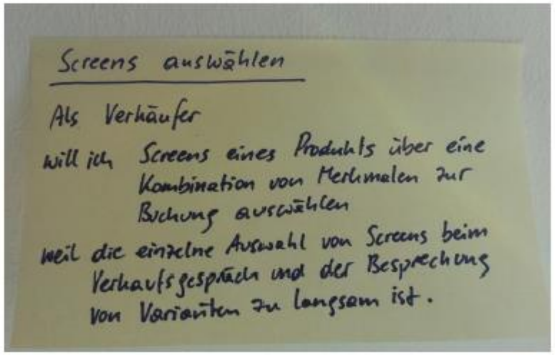
2 groups (308 addresses)
47 individual addresses
355 addresses total

Showing page: 2 of 231 pages

# *Wie macht man es in der Praxis?*

User Story

- Als High Level Anforderung
- Zur Priorisierung
- Im Backlog



Screens auswählen

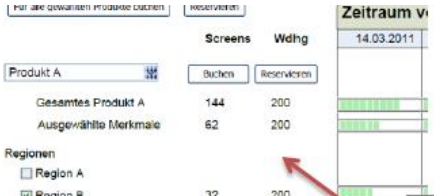
Als Verkäufer

will ich Screens eines Produkts über eine Kombination von Merkmalen zur Buchung auswählen

weil die einzelne Auswahl von Screens beim Verkaufsgespräch und der Besprechung von Varianten zu langsam ist.

Verfeinern zum gegebenen Zeitpunkt!

# Wie macht man es in der Praxis?

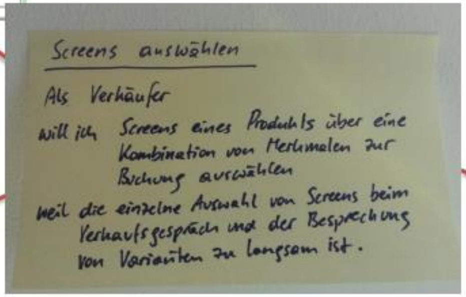


Für alle gewählten Produkte buchen | Reservieren

Zeitraum v

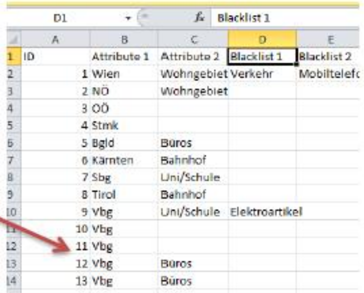
| | Screens | Wdhg | 14.03.2011 |
|---|---|---|---|
| Produkt A | Buchen | Reservieren | |
| Gesamtes Produkt A | 144 | 200 | |
| Ausgewählte Merkmale | 62 | 200 | |
| Regionen | | | |
| □ Region A | | | |
| ☑ Region B | 32 | 200 | |

**UI Scribble**



Screens auswählen

Als Verkäufer
will ich Screens eines Produkts über eine Kombination von Merkmalen zur Buchung auswählen
weil die einzelne Auswahl von Screens beim Verkaufsgespräch und der Besprechung von Varianten zu langsam ist.



D1 | Blacklist 1

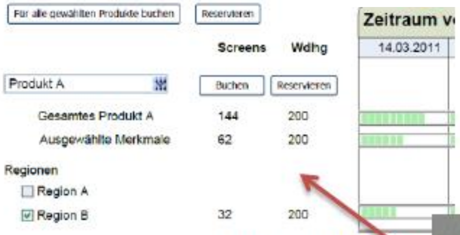
| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | ID | Attribute 1 | Attribute 2 | Blacklist 1 | Blacklist 2 |
| 2 | 1 | Wien | Wohngebiet | Verkehr | Mobiltelefc |
| 3 | 2 | NÖ | Wohngebiet | | |
| 4 | 3 | OÖ | | | |
| 5 | 4 | Stmk | | | |
| 6 | 5 | Bgld | Büros | | |
| 7 | 6 | Kärnten | Bahnhof | | |
| 8 | 7 | Sbg | Uni/Schule | | |
| 9 | 8 | Tirol | Bahnhof | | |
| 10 | 9 | Vbg | Uni/Schule | Elektroartikel | |
| 11 | 10 | Vbg | | | |
| 12 | 11 | Vbg | | | |
| 13 | 12 | Vbg | Büros | | |
| 14 | 13 | Vbg | Büros | | |

**Bestehende Artefakte**
**Musterartefakte**

**Schlüsselbeispiele**

© by Techtalk

# Wie macht man es in der Praxis?

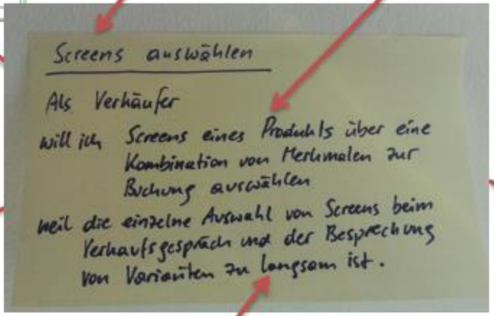


| | Screens | Wdhg | 14.03.2011 |
|---|---|---|---|
| Produkt A | | | |
| Gesamtes Produkt A | 144 | 200 | |
| Ausgewählte Merkmale | 62 | 200 | |
| Regionen | | | |
| □ Region A | | | |
| ☑ Region B | 32 | 200 | |

UI Scribble

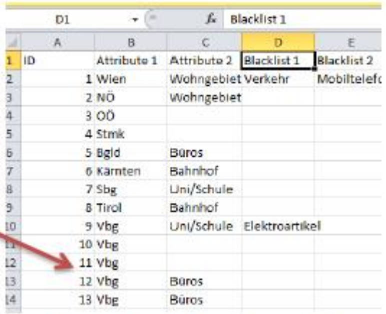
Titel

Akteur und Ziel



Screens auswählen

Als Verkäufer
will ich Screens eines Produkts über eine Kombination von Merkmalen zur Buchung auswählen
weil die einzelne Auswahl von Screens beim Verkaufsgespräch und der Besprechung von Varianten zu langsam ist.



D1 | Blacklist 1

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | ID | Attribute 1 | Attribute 2 | Blacklist 1 | Blacklist 2 |
| 2 | 1 | Wien | Wohngebiet | Verkehr | Mobiltelefo |
| 3 | 2 | NÖ | Wohngebiet | | |
| 4 | 3 | OÖ | | | |
| 5 | 4 | Stmk | | | |
| 6 | 5 | Bgld | Büros | | |
| 7 | 6 | Kärnten | Bahnhof | | |
| 8 | 7 | Sbg | Uni/Schule | | |
| 9 | 8 | Tirol | Bahnhof | | |
| 10 | 9 | Vbg | Uni/Schule | Elektroartikel | |
| 11 | 10 | Vbg | | | |
| 12 | 11 | Vbg | | | |
| 13 | 12 | Vbg | Büros | | |
| 14 | 13 | Vbg | Büros | | |

Bestehende Artefakte
Musterartefakte

Schlüsselbeispiele

Nutzenverstanden?

*Warum benötigt?*

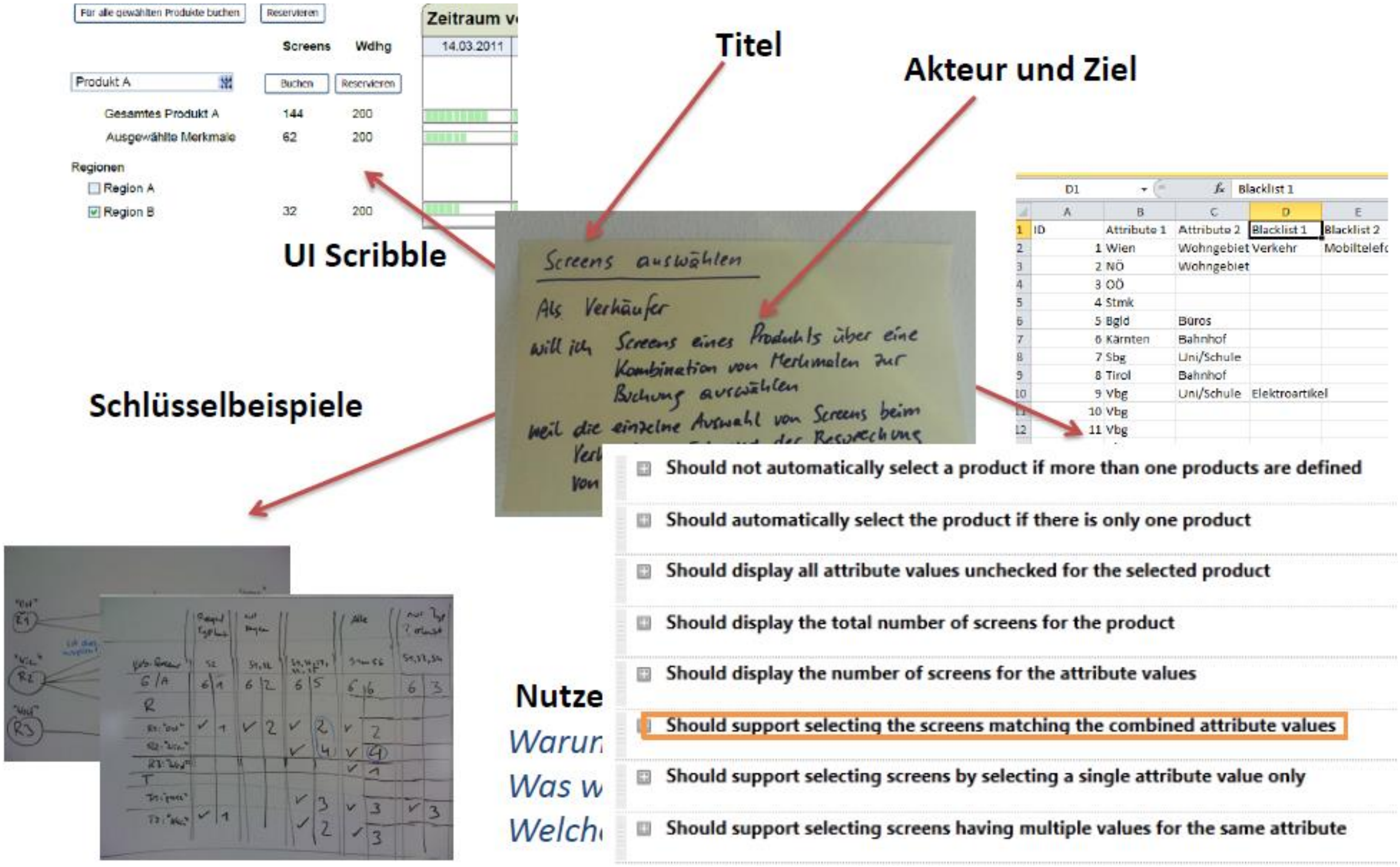
*Was wäre, wenn das Feature NICHT vorhanden ist?*

*Welcher Workaround wäre notwendig?*

Schätzung

© by Techtalk

# *Wie macht man es in der Praxis?*



Für alle gewählten Produkte buchen | Reservieren

Zeitraum v

| | Screens | Wdhg | |
|---|---|---|---|
| Produkt A | Buchen | Reservieren | 14.03.2011 |
| Gesamtes Produkt A | 144 | 200 | |
| Ausgewählte Merkmale | 62 | 200 | |
| Regionen | | | |
| □ Region A | | | |
| ☑ Region B | 32 | 200 | |

Screens auswählen

Als Verkäufer
will ich Screens eines Produkts über eine Kombination von Merkmalen zur Buchung auswählen
weil die einzelne Auswahl von Screens beim
Verk … der Besprechung
von

D1 | Blacklist 1

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | ID | Attribute 1 | Attribute 2 | Blacklist 1 | Blacklist 2 |
| 2 | 1 | Wien | Wohngebiet | Verkehr | Mobiltelefc |
| 3 | 2 | NÖ | Wohngebiet | | |
| 4 | 3 | OÖ | | | |
| 5 | 4 | Stmk | | | |
| 6 | 5 | Bgld | Büros | | |
| 7 | 6 | Kärnten | Bahnhof | | |
| 8 | 7 | Sbg | Uni/Schule | | |
| 9 | 8 | Tirol | Bahnhof | | |
| 10 | 9 | Vbg | Uni/Schule | Elektroartikel | |
| 11 | 10 | Vbg | | | |
| 12 | 11 | Vbg | | | |

⊞ Should not automatically select a product if more than one products are defined

⊞ Should automatically select the product if there is only one product

⊞ Should display all attribute values unchecked for the selected product

⊞ Should display the total number of screens for the product

⊞ Should display the number of screens for the attribute values

⊞ Should support selecting the screens matching the combined attribute values

⊞ Should support selecting screens by selecting a single attribute value only

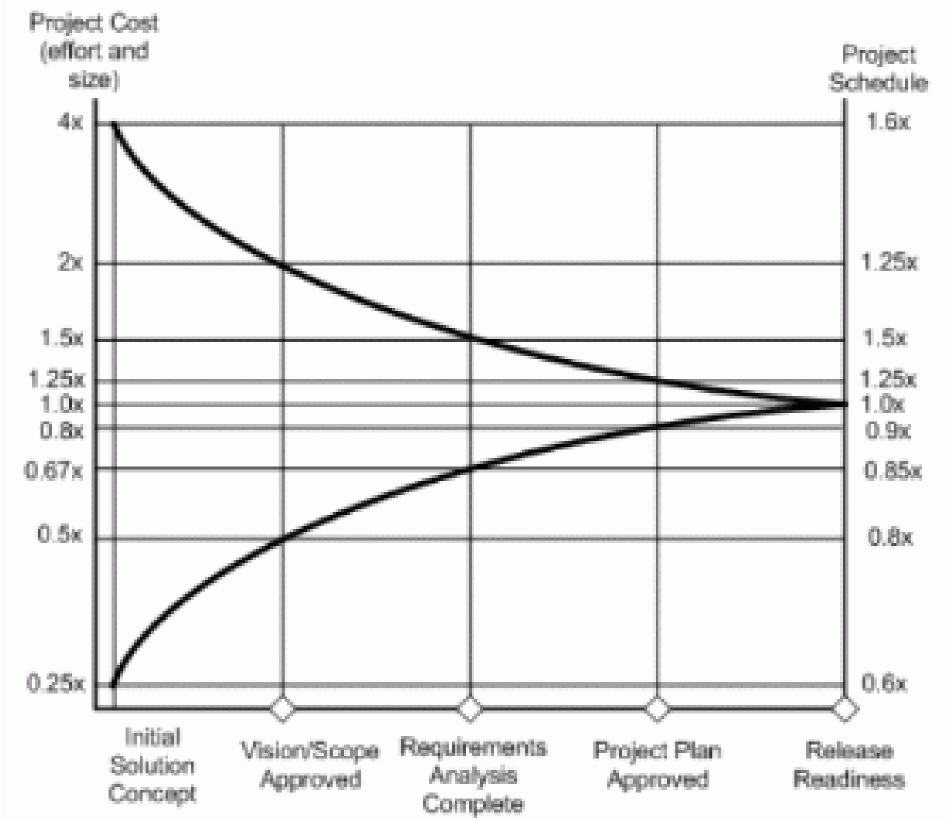
⊞ Should support selecting screens having multiple values for the same attribute

# *Schätzen – Wie geht man heran?*

Tipps

- Schätzen basiert typischerweise aus Erfahrungswerten aus der Vergangenheit
- Vergleichen Sie
  - Feature mit bereits umgesetzten Features (aus aktuellem Projekt, aus vergangenen Projekten)
  - Vergleichen Sie durch Zählen (Maskenelemente (Reiter, Felder, Schritte etc.)
  - Vergleichen Sie Komplexität (Ist Feature X komplexer als Feature y)
- Generell:
  - Verwenden Sie einfache Methoden – nicht zu Komplizierte
  - Komplexe Methoden (Function Points etc. benötigen viel Erfahrung)
  - Trend geht zu einfachen Schätzmethoden (bspw. Story Points)
  - Schätzen ist Teamarbeit!
- Schätzwerte ohne Statistiken (Konfidenzintervalle) sagt nicht viel aus!
- Lernen Sie aus Erfahrung: **Evaluieren Sie am Ende der Iteration Geschätzte Aufwände mit Ist-Aufwänden (Bsp: Burn Down Chart)**

# *Cone of Uncertainty (MC Connell)*



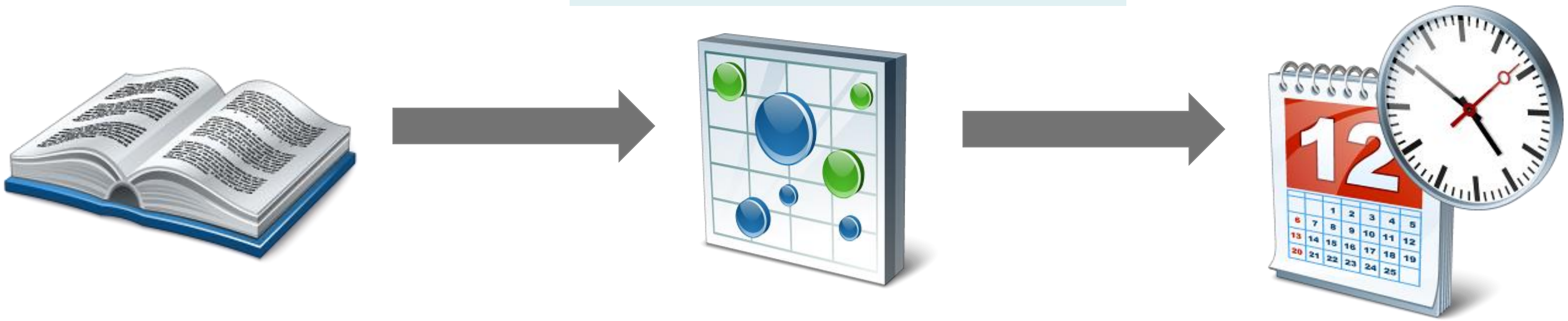
# *User Stories – Schätzen*

User Stories

Hinweis: einfache Werte ausreichend (Fibonacci)

Work Items / ToDos

Trend: Wird oft nicht mehr gemacht!



Priorisierung

- *Was ist in Iteration*

Story Points

- *Vergleich der Größe*
- *Vergleich der Komplexität*

Aufwand

- *In h (PT)*
- *Kleine Tasks*

Was steckt dahinter

- Team schätzt selbst und committed sich zu Umfang
- Adaptiv: Team lernt, wieviel in eine Timebox (Iteration) passt
- Methode: „Schnell“ - Planning Poker

# *Was ist die Idee hinter Story Points*

**Keine Zeiteinheiten sondern  relative Größenordnungen**

- Warum?
  - Ideal vs. Echte Zeit (Meetings, Unterbrechungen, etc)
  - Fiktive Einheiten (keine Zeit!)
  - Es geht schnell (Planning Poker)
  - Vergleich zu Referenzen
    - Bsp: Ein Logon Screen dauert 2 SPs (aus Erfahrung), Feature X hat mehr Komplexität und dauert x SPs

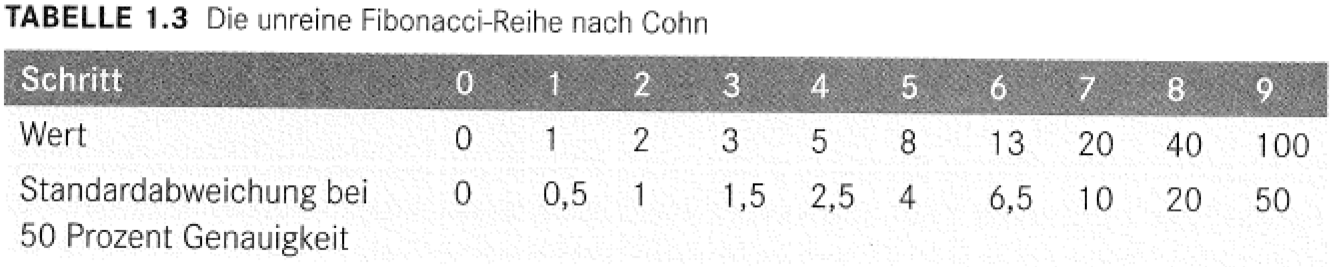
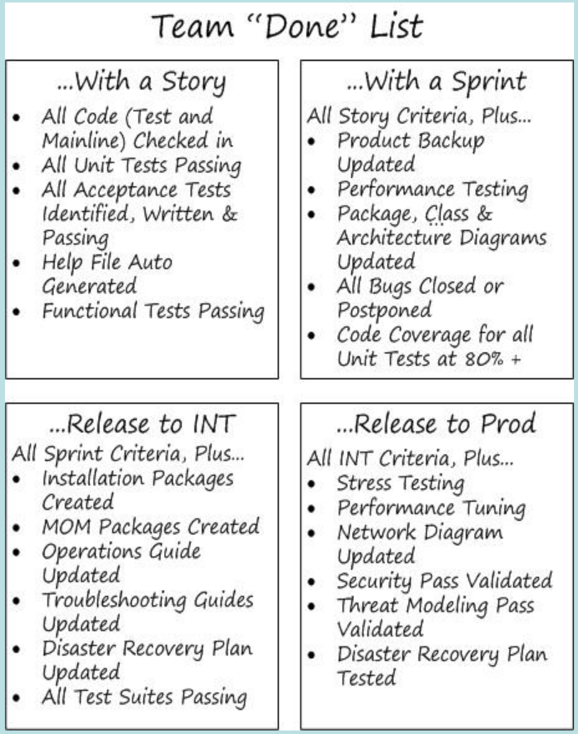


**TABELLE 1.3** Die unreine Fibonacci-Reihe nach Cohn

| Schritt | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Wert | 0 | 1 | 2 | 3 | 5 | 8 | 13 | 20 | 40 | 100 |
| Standardabweichung bei 50 Prozent Genauigkeit | 0 | 0,5 | 1 | 1,5 | 2,5 | 4 | 6,5 | 10 | 20 | 50 |

# *Einschub – Definition of Done*

- Coding != Done



## Team "Done" List

### ...With a Story

- All Code (Test and Mainline) Checked in
- All Unit Tests Passing
- All Acceptance Tests Identified, Written & Passing
- Help File Auto Generated
- Functional Tests Passing

### ...With a Sprint

All Story Criteria, Plus...

- Product Backup Updated
- Performance Testing
- Package, Class & Architecture Diagrams Updated
- All Bugs Closed or Postponed
- Code Coverage for all Unit Tests at 80% +

### ...Release to INT

All Sprint Criteria, Plus...

- Installation Packages Created
- MOM Packages Created
- Operations Guide Updated
- Troubleshooting Guides Updated
- Disaster Recovery Plan Updated
- All Test Suites Passing

### ...Release to Prod

All INT Criteria, Plus...

- Stress Testing
- Performance Tuning
- Network Diagram Updated
- Security Pass Validated
- Threat Modeling Pass Validated
- Disaster Recovery Plan Tested

© by Mitch Lacey (Link:http://mitch.lacey.com)

Achtung:

- Nur Teamanteil (Scrum) für Umsetzung!
- Req. Erhebung etc fehlt

# *Einschub: Wie schätzt man SW-Projekte?*

- Je nach Projektart, -größe & Vorgehensmodell etwas unterschiedlich
- Generell immer ähnliche Schätzpositionen
  - Umsetzungsaufwand je Feature (bspw. in Storypoints)
    (DoD -> was ist drinnen - bspw. Entwicklertests)
  - QS (Acceptance Testing, Testautomatisierung, Testfälle)
  - Projektmanagement (bzw. PO)
  - Requirements Engineering / UI Design
  - Puffer für Risiken (Unsicherheiten)
  - Aufschläge für Gewährleistung (bei Fixpreis)

- Oft: Umsetzung als Basis und mit prozentuellen Aufschlägen bspw für RE / QS etc
- Gegenrechnung über zeitliches Teamgebirge (Personen bzw. FTE)
- Achtung: Verkaufspreis nicht immer 1:1 zu Aufwand
  (Sales Sicht – um welchen Preis kann ich ein Projekt verkaufen)

# *StoryPoint zu Aufwand*

- Velocity (SCRUM) ist Durchsatz in einer Iteration

Wie viel haben wir uns vorgenommen
vs.
Wieviel haben wir geschafft

- Story Point zu Aufwand

Aufwand = SP * Velocity

# Planning poker

- An iterative approach to estimating
- Steps
  - Each estimator is given a deck of cards, each card has a valid estimate written on it
  - Customer/Product owner reads a story and it's discussed briefly
  - Each estimator selects a card that's his or her estimate
  - Cards are turned over so all can see them
  - Discuss differences (especially outliers)
  - Re-estimate until estimates converge

# *Übung: Erhebung von User Stories*

- 6 Gruppen
- 2 User Stories samt Akzeptanzkriterien (2-3)
- Schätzen der 2 User Stories (Umsetzung)
  - Wichtig: Brutto: Design, Umsetzung inkl. Testing, Doku, etc.
  - Methode: Planning Poker Game
- Zeit 45‘
- Präsentation

# *Requirements (4)*

- Startseite
  - Hat zentralen Activity-Stream (personalisiert)
  - Unternehmens-Newsbereich
    - Titel, Inhalt, Foto (später)
    - Online von/bis
    - Editieren durch Portaladmins
    - Lesen können alle
  - Dokumentenbiliothek
    - Editieren durch Portaladmins
    - Lesen/Download alle
  - Userseite (Personen des Unternehmens)
  - Services (personalisierbar ) -später

# *Referenzen*

- *Kent Beck, Martin Fowler*
  **“Planning Extreme Programming”**
  Addison-Wesley, 2001
- Mike Cohn

  **“ User Stories Applied”**

  Addison-Wesley, 2004
- Interesting **slide deck** by Mike Cohn
  http://www.mountaingoatsoftware.com/presentations/119-introduction-to-user-stories
- Mike Cohn
  **„Agile Estimating and Planning“**
  Prentice-Hall, 2005

# Practical Software Engineering

*Wie kommt man zu einer Software Architektur*

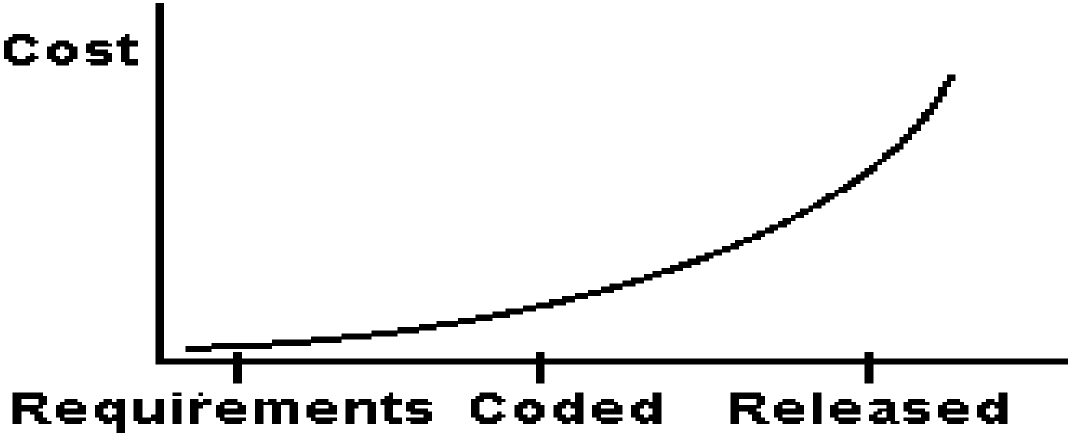
FH JOANNEUM
Internettechnik
Software Design
WS 2018

# *Agenda*

- Wie kommt man zu einer passenden Software Architektur
  - Ein praktischer Guide in Kurzform (basierend auf dem Arc42 Ansatz)

# *Warum macht SW-Architektur Sinn?*

- Über spät erkannte Fehler:



Als Entwickler tun Sie es implizit mit dem Unterschied:

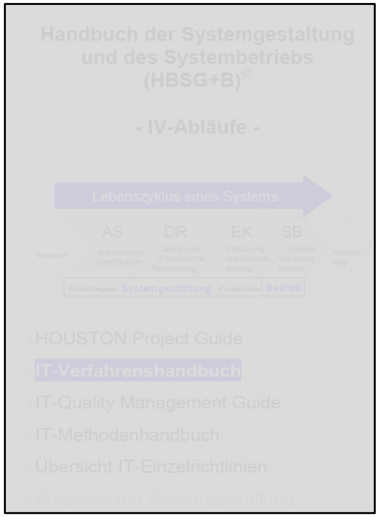
SW-Architektur ist „Design im Großen“

# *Notwendige Voraussetzungen*

- Kennen der technischen Rahmenbedingungen
  - Fachliche Requirements (Must)
  - Non Functional Requirements
  - Stakeholder (Für Architektur – technische)
  - Technische Requirements des Kunden
    - Enterprise Strategie
    - Technologie-Stacks
    - Betriebsinfrastrukturen
    - Security Anforderungen
    - etc

# *Bspe für techn. Kunden Requ.*

Prozesse



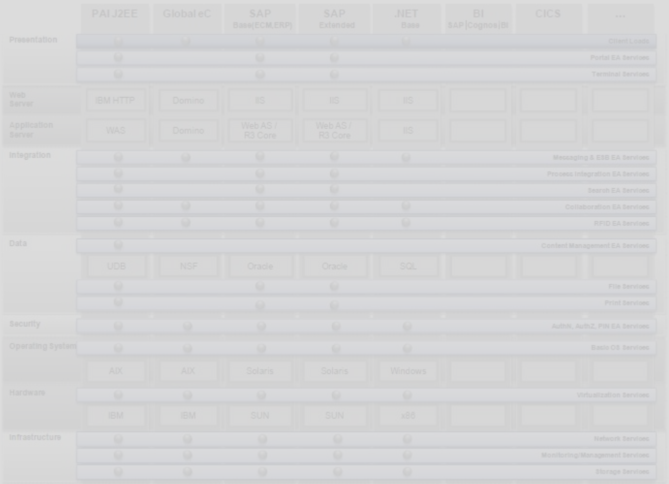
**Handbuch der Systemgestaltung und des Systembetriebs (HBSG+B)©**

**- IV-Abläufe -**

Lebenszyklus eines Systems
Initiation
AS
Anforderungs-spezifikation
DR
Design und IV-technische Realisierung
EK
Einführung und Konsoli-dierung
SB
System-betreuung / betrieb
Abschal-tung
Projektbeginn
Systemgestaltung
Projektende
Betrieb

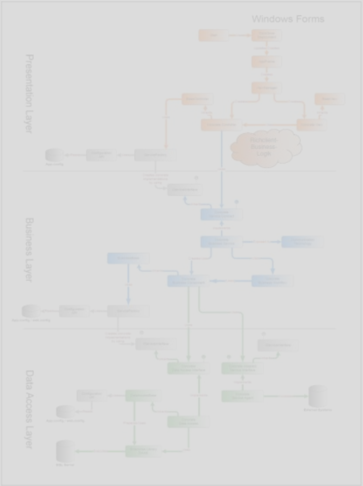
- HOUSTON Project Guide
- **IT-Verfahrenshandbuch**
- IT-Quality Management Guide
- IT-Methodenhandbuch
- Übersicht IT-Einzelrichtlinien
- Prozesse der Systemgestaltung

Technologie Stacks



| | PAI J2EE | Global eC | SAP<br>Base(ECM,ERP) | SAP<br>Extended | .NET<br>Base | BI<br>SAP\|Cognos\|BI | CICS | ... | |
|---|---|---|---|---|---|---|---|---|---|
| Presentation | ● | ● | ● | ● | ● | | | | Client Loads |
| | ● | | ● | ● | | | | | Portal EA Services |
| | ● | | ● | ● | | | | | Terminal Services |
| Web Server | IBM HTTP | Domino | IIS | IIS | IIS | | | | |
| Application Server | WAS | Domino | Web AS / R3 Core | Web AS / R3 Core | IIS | | | | |
| Integration | ● | ● | ● | ● | ● | | | | Messaging & ESB EA Services |
| | ● | | ● | ● | | | | | Process Integration EA Services |
| | ● | | ● | ● | | | | | Search EA Services |
| | ● | ● | ● | ● | ● | | | | Collaboration EA Services |
| | ● | ● | ● | ● | ● | | | | RFID EA Services |
| Data | ● | | | | | | | | Content Management EA Services |
| | UDB | NSF | Oracle | Oracle | SQL | | | | |
| | ● | | ● | ● | | | | | File Services |
| | ● | | ● | ● | | | | | Print Services |
| Security | ● | ● | ● | ● | ● | | | | AuthN, AuthZ, PIN EA Services |
| Operating System | ● | ● | ● | ● | ● | | | | Basic OS Services |
| | AIX | AIX | Solaris | Solaris | Windows | | | | |
| Hardware | ● | ● | ● | ● | ● | | | | Virtualization Services |
| | IBM | IBM | SUN | SUN | x86 | | | | |
| Infrastructure | ● | ● | ● | ● | ● | | | | Network Services |
| | ● | ● | ● | ● | ● | | | | Monitoring/Management Services |
| | ● | ● | ● | ● | ● | | | | Storage Services |

Designvorgaben



# *Guideline*

- Systemkontext definieren
- Systemarchitektur definieren
- Technische Architektur definieren
- Fachliche Architektur festlegen

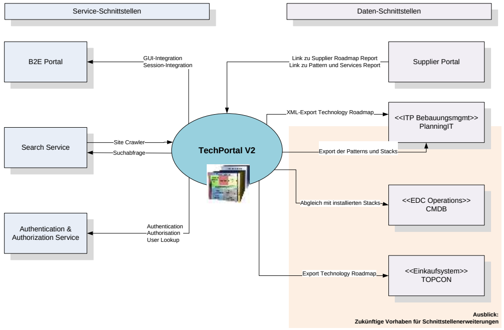
# 1: Systemkontext definieren

**Ziele:**

- Wie ist das System als Blackbox in seine Umgebung eingebettet?
- Welche wesentlichen Informationen gehen in das System hinein bzw. heraus?

**Methode:**

- UML Komponentendiagramm oder einfaches Diagramm


Service-Schnittstellen
Daten-Schnittstellen
B2E Portal
GUI-Integration
Session-Integration
Link zu Supplier Roadmap Report
Link zu Pattern und Services Report
Supplier Portal
XML-Export Technology Roadmap
<<ITP Bebauungsmgmt>> PlanningIT
Search Service
Site Crawler
Suchabfrage
TechPortal V2
Export der Patterns und Stacks
Abgleich mit installierten Stacks
<<EDC Operations>> CMDB
Authentication & Authorization Service
Authentication
Authorisation
User Lookup
Export Technology Roadmap
<<Einkaufsystem>> TOPCON
Ausblick:
Zukünftige Vorhaben für Schnittstellenerweiterungen

- Analogon aus XP: Methapher

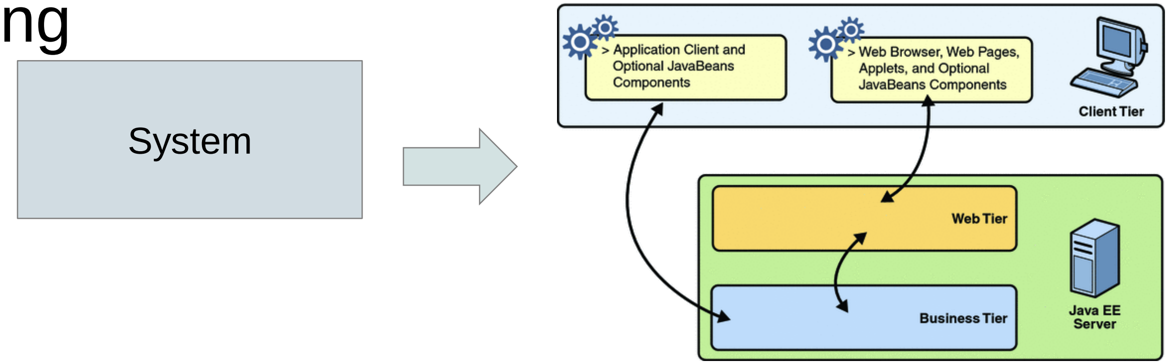
# *Systemkontextdiagramm*

- Was muss enthalten sein
  - System mit Namen
  - Nachbarsysteme
  - Schnittstellen (Best Practise: Typisierung der SST)

# 2: System-Architektur definieren

**Ziele**:

- Finden eines passenden Architekturpatterns für die Aufgabenstellung



System

> Application Client and Optional JavaBeans Components
> Web Browser, Web Pages, Applets, and Optional JavaBeans Components
Client Tier
Web Tier
Business Tier
Java EE Server

**Methode:**

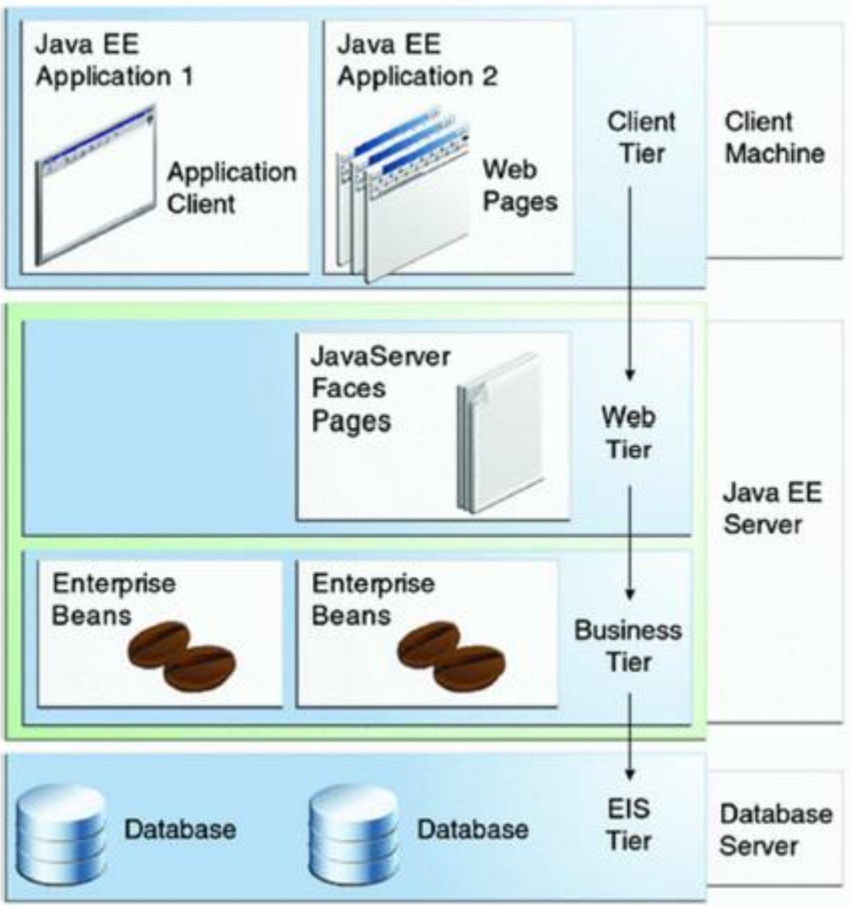
- Analyse der technischen Rahmenbedingungen (NFRs, Technologiestacks, etc)
- Divide & Conquer (Teilen in beherrschbare Einheiten)
- Rad nicht neu erfinden!

# Bsp: JEE System Architektur

*Example:*

**JEE Architecture**

**Distributed**
**Multitiered**
**Applications**



Adressiert:
Verteilungssicht /
Skalierung / NFRs

# *Exkurs: System-Architektur-Patterns*

Beispiele für Patterns

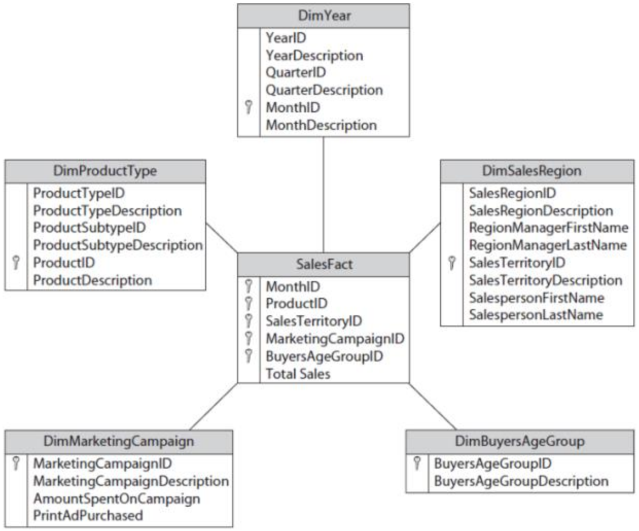
- OLTP (“klassisch”)
- OLAP / BI
- Cloud / Microservices

# *OLTP / OLAP*

- Online Transactional Processing (OLTP)
  - Transaktionsgetriebene Business Applikation
  - Typisch: Atomare User Transaktionen auf geringen Datenmengen
  - „Schnell“
- Online Analytical Processing (OLAP)
  - Datengetriebene Applikationen (Reporting, Analyse, Mining)
  - Hohe Datenmengen
  - “langsam”
  - Oft: parallel zu OLTP-Applikationen
  - Datenbank-lastig, basieren meist auf Produkten
  - Aktuell sehr im Trend!

# *OLAP - Datenmodellierung*

- Sternschema



DimYear
YearID
YearDescription
QuarterID
QuarterDescription
MonthID
MonthDescription

DimProductType
ProductTypeID
ProductTypeDescription
ProductSubtypeID
ProductSubtypeDescription
ProductID
ProductDescription

DimSalesRegion
SalesRegionID
SalesRegionDescription
RegionManagerFirstName
RegionManagerLastName
SalesTerritoryID
SalesTerritoryDescription
SalespersonFirstName
SalespersonLastName

SalesFact
MonthID
ProductID
SalesTerritoryID
MarketingCampaignID
BuyersAgeGroupID
Total Sales

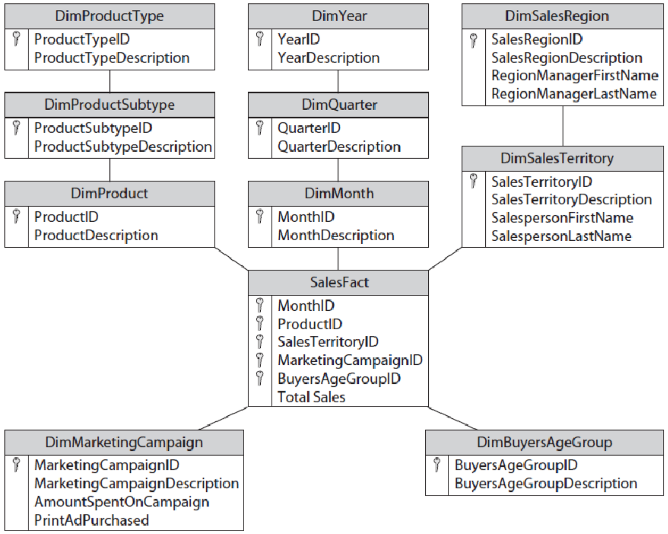
DimMarketingCampaign
MarketingCampaignID
MarketingCampaignDescription
AmountSpentOnCampaign
PrintAdPurchased

DimBuyersAgeGroup
BuyersAgeGroupID
BuyersAgeGroupDescription

- **Faktentabelle**: Denormalisierte, redundante Haltung von sog. Measures (=Kennzahlen)
- **Dimensionen**: Genaue Kategorisierung von Measures für weitere Analysen

Sternschema auch in transaktionalen Applikationen anwendbar!

- Snowflake Schema



DimProductType
ProductTypeID
ProductTypeDescription

DimYear
YearID
YearDescription

DimSalesRegion
SalesRegionID
SalesRegionDescription
RegionManagerFirstName
RegionManagerLastName

DimProductSubtype
ProductSubtypeID
ProductSubtypeDescription

DimQuarter
QuarterID
QuarterDescription

DimSalesTerritory
SalesTerritoryID
SalesTerritoryDescription
SalespersonFirstName
SalespersonLastName

DimProduct
ProductID
ProductDescription

DimMonth
MonthID
MonthDescription

SalesFact
MonthID
ProductID
SalesTerritoryID
MarketingCampaignID
BuyersAgeGroupID
Total Sales

DimMarketingCampaign
MarketingCampaignID
MarketingCampaignDescription
AmountSpentOnCampaign
PrintAdPurchased

DimBuyersAgeGroup
BuyersAgeGroupID
BuyersAgeGroupDescription

- **Hierarchien von Dimensionen**: Beziehungen von Dimensionen; jede Hierarachie hat eigene Tabelle, Höhere Aggregate werden wieder berechnet.

# *Cloud*

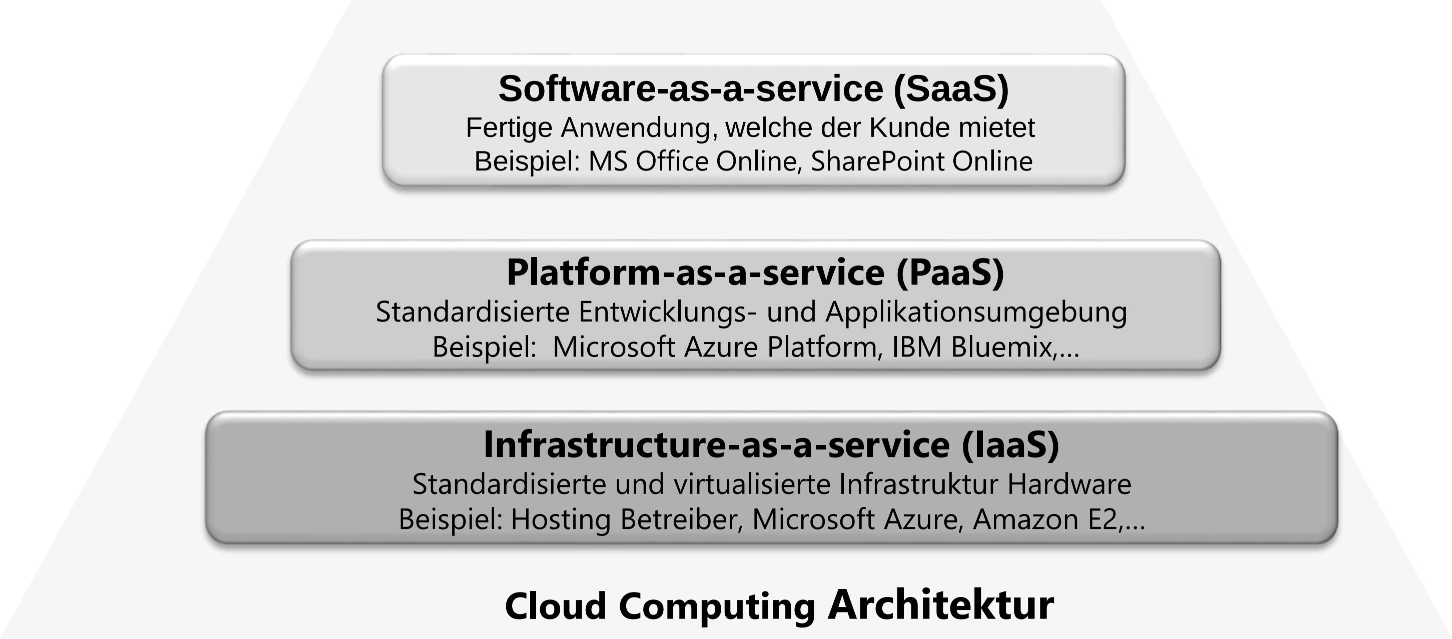
- Cloud = externes Hosting
- Cloud = Mietmodell (veränderbar)
- Cloud = hohe Skalierbarkeit
- Cloud = lose gekoppelte Architektur

Vom Hype zum Mainstream

- Hersteller pushen „Cloud First“ Ansatz
- Cloudpatterns halten Einzug in Applikationsarchitekturen
- Europa nicht ganz so euphorisch wie andere Kontinente

# Cloud - Begriffe

- Cloud = Mietmodell (veränderbar)



**Software-as-a-service (SaaS)**
Fertige Anwendung, welche der Kunde mietet
Beispiel: MS Office Online, SharePoint Online

**Platform-as-a-service (PaaS)**
Standardisierte Entwicklungs- und Applikationsumgebung
Beispiel: Microsoft Azure Platform, IBM Bluemix,...

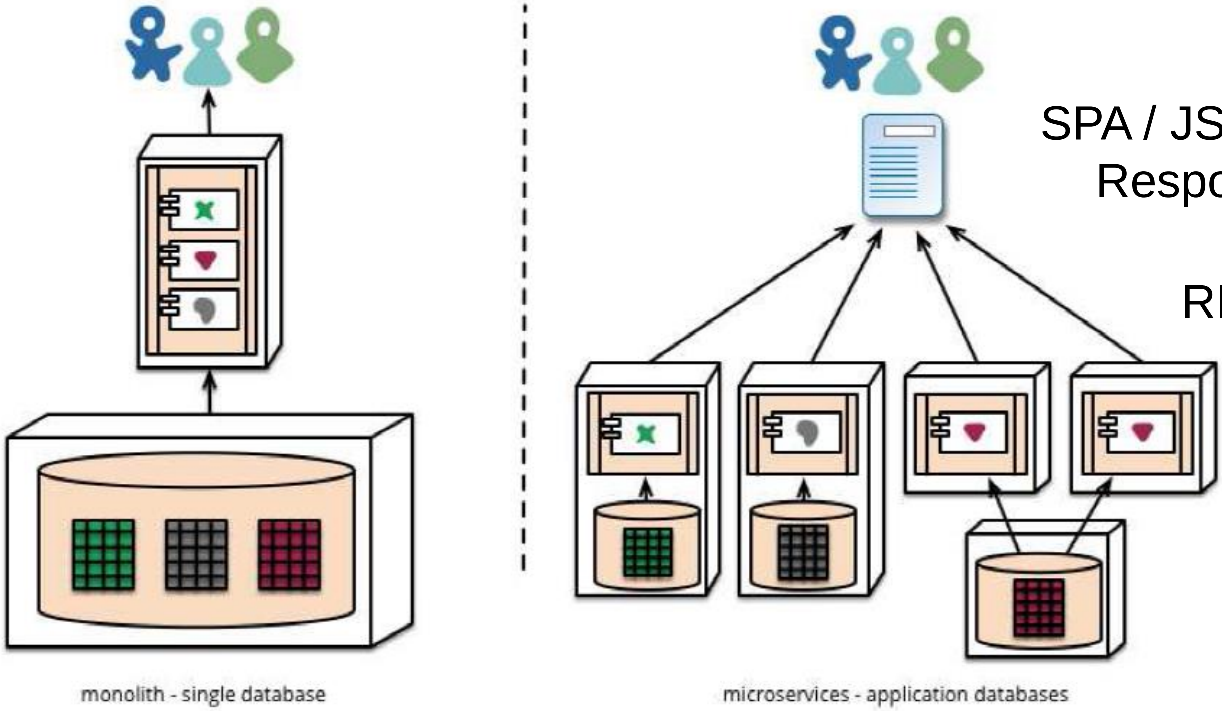
**Infrastructure-as-a-service (IaaS)**
Standardisierte und virtualisierte Infrastruktur Hardware
Beispiel: Hosting Betreiber, Microsoft Azure, Amazon E2,...

**Cloud Computing Architektur**

- **Public Cloud**: Beliebige Kunden nutzen Cloud Dienste eines Providers
- **Private Cloud**: Cloud für eine spezielle Organisation von einem Provider
- **Public-Private Cloud**: Private Cloud für spezielle Zwecke (bspw. NGOs, staatliche Betriebe)

# *Cloud Ansatz: Micro Services*

**Kernidee: Lose Kopplung  / Skalierbarkeit**


monolith - single database
microservices - application databases

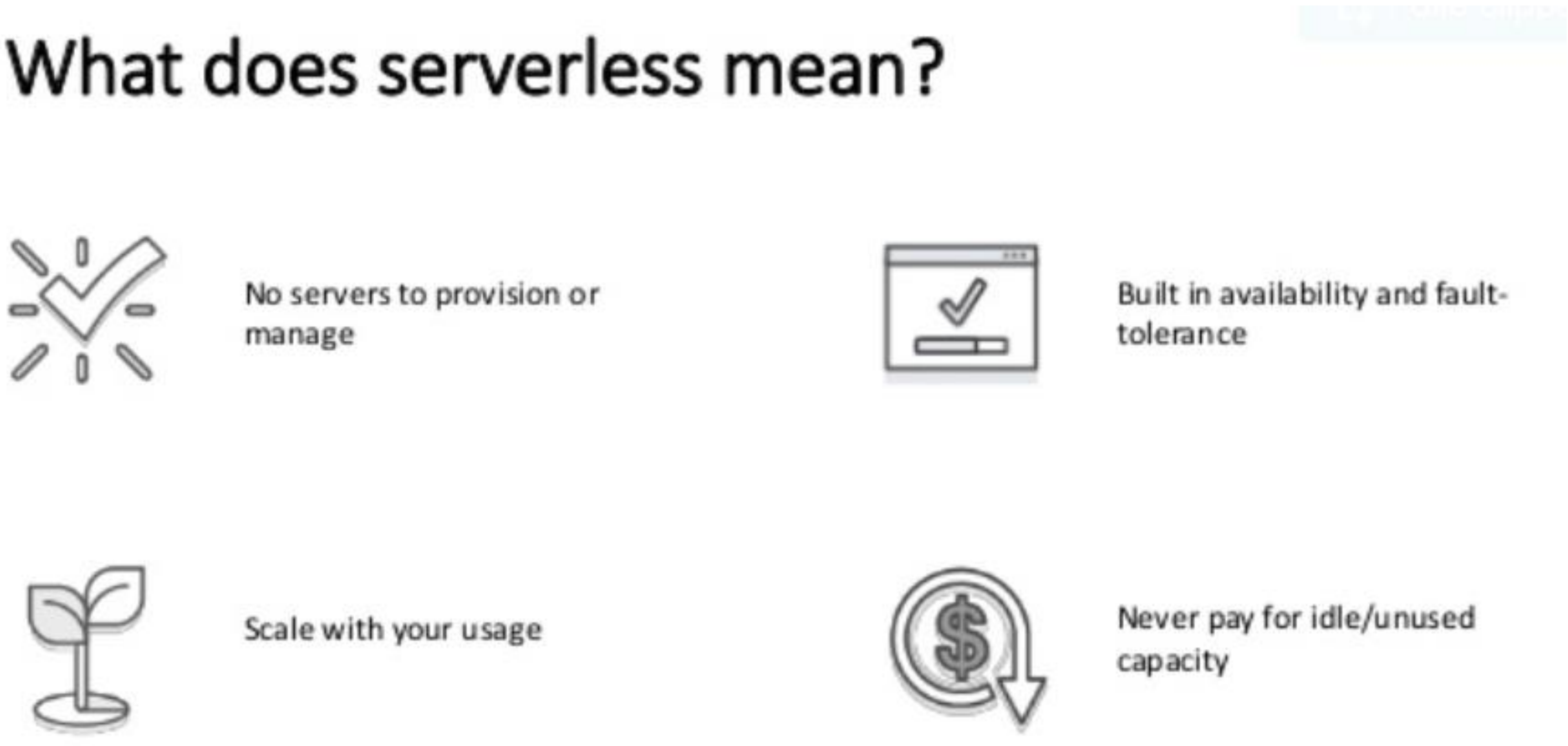
SPA / JS basierte
Responsive-UI

REST /JSON

Micro Services

**Achtung auf Komplexität – angemessen anwenden**

# *Serverless Architectures(1)*



## What does serverless mean?

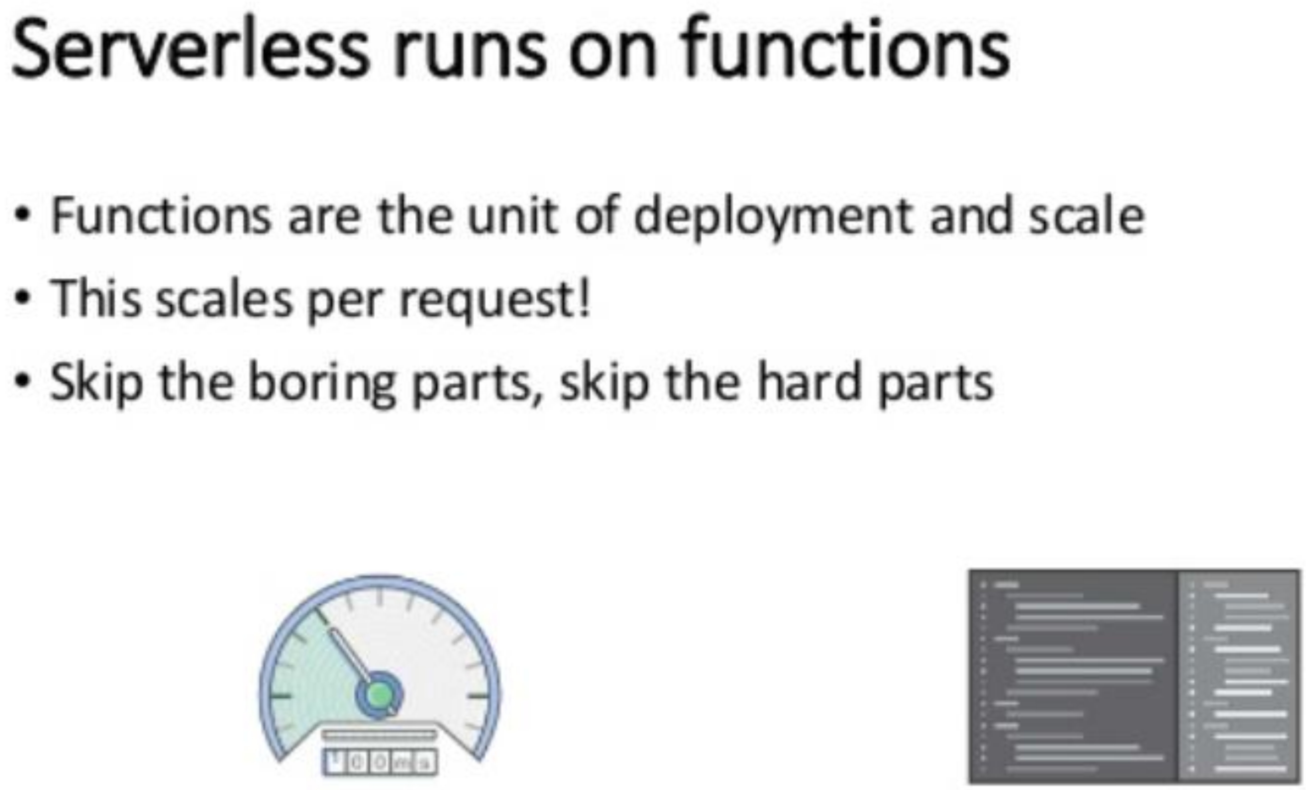
No servers to provision or manage

Built in availability and fault-tolerance

Scale with your usage

Never pay for idle/unused capacity

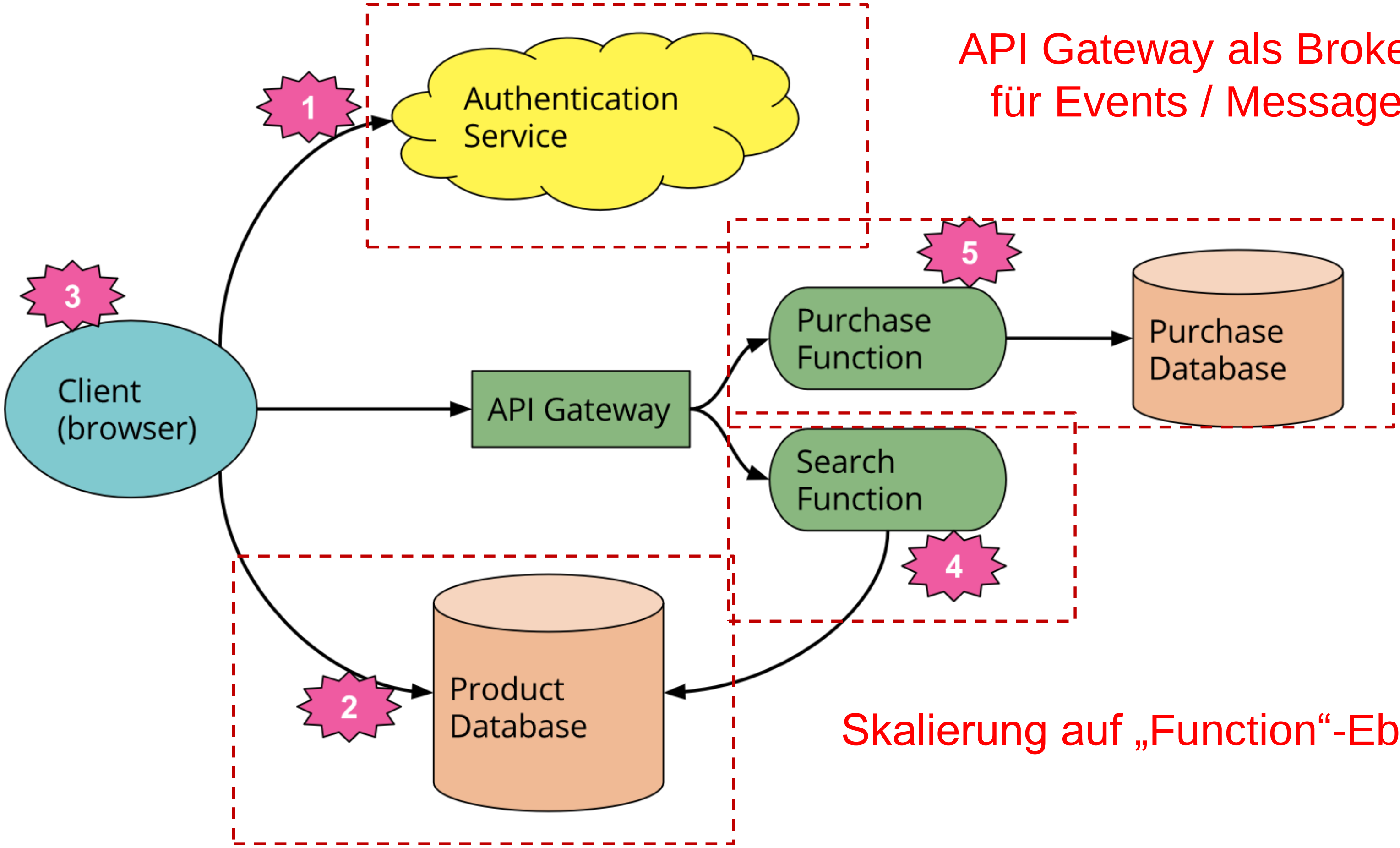
# *Serverless Architectures(2)*



## Serverless runs on functions

- Functions are the unit of deployment and scale
- This scales per request!
- Skip the boring parts, skip the hard parts

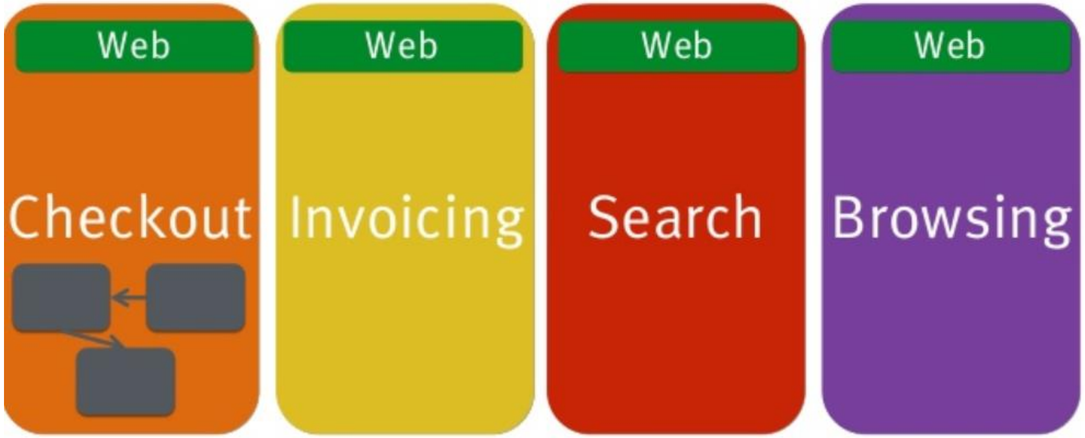
# Serverless Gedanke



API Gateway als Broker
für Events / Messages

Skalierung auf „Function“-Ebene

# *Self Contained Systems*



- Prinzip
  - Vertikale Systeme inkl. Web
  - UI ist immer Teil des Systems
  - Microservices-Backend
  - Redundante UIs
  - Wiederverwendete Asset-Bibliotheken
- Bekannte Vertreter
  - Google
  - Otto

# *3: Technische Architektur festlegen*

**Ziele:**

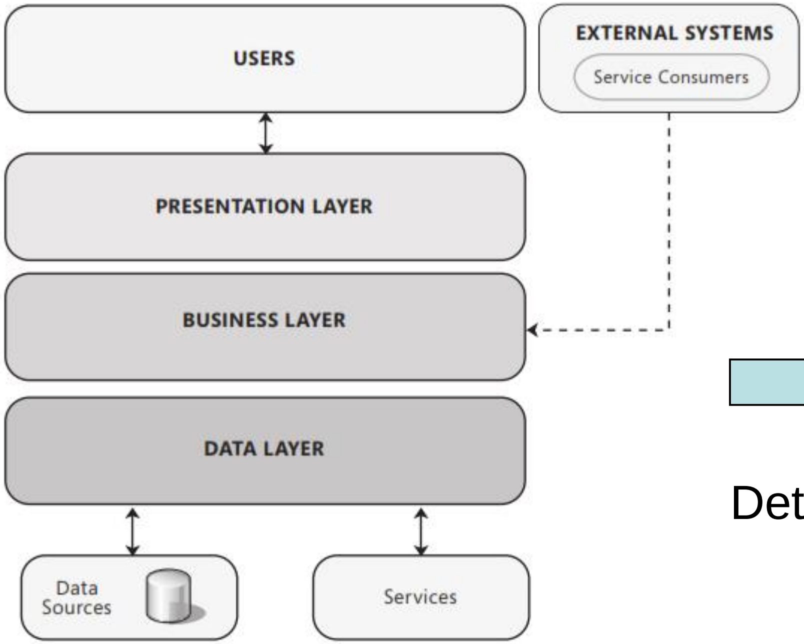
- Festlegen der technischen Architektur
- Definieren von Architekturaspekten

**Methode:**

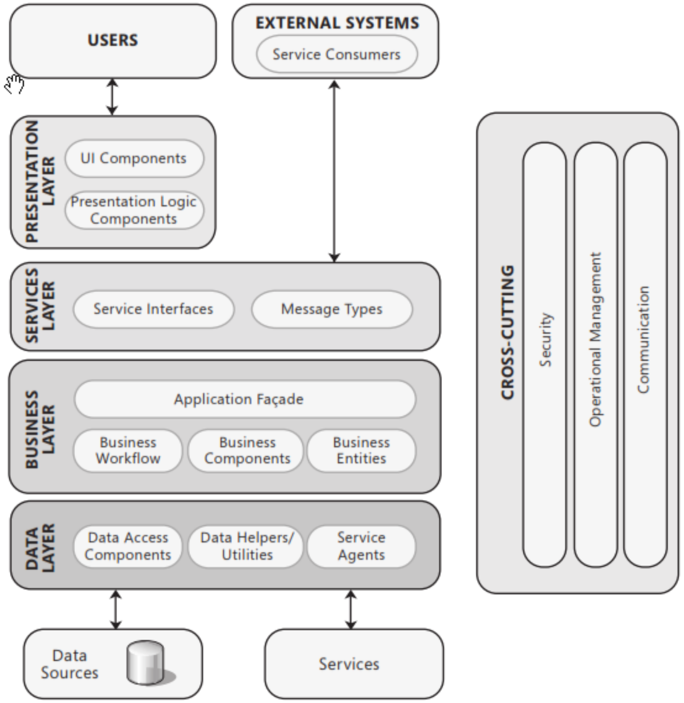
- Analyse der technischen und fachlichen Rahmenbedingungen
- Patterns studieren (Rad nicht neu erfinden)

# *Technische Architektur – Bsp.*

- Layerstruktur  (nach Microsoft Patterns & Practices)



Detaillierter



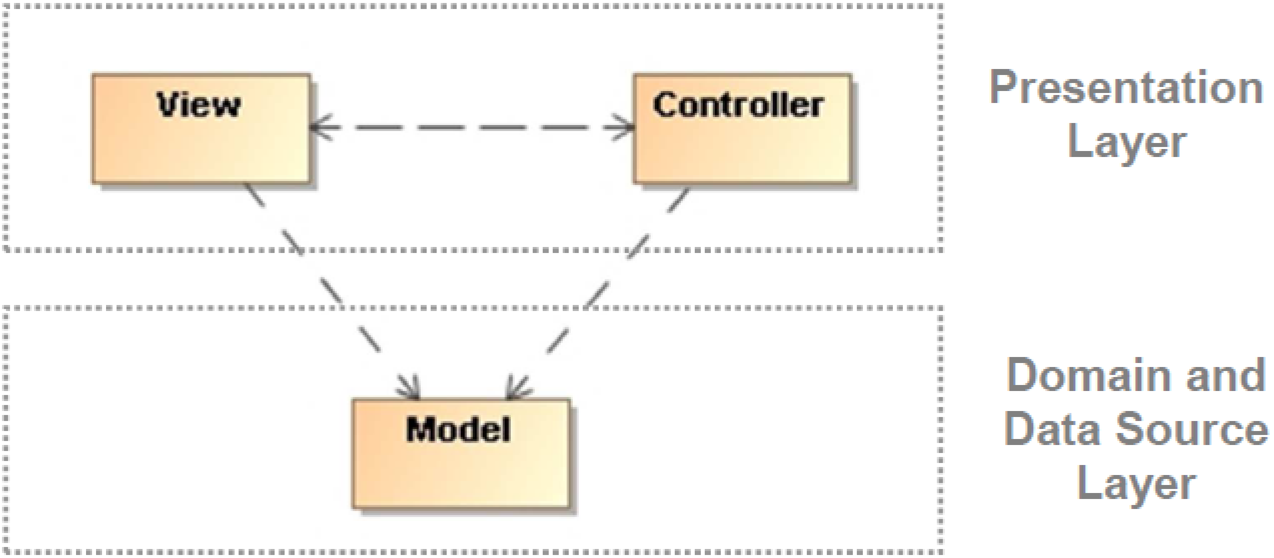
# *Typische Schichten & Patterns*

- Presentation Layer
- Service / Domain / Business Layer
- Data Access Layer

# *Presentation Layer (1)*

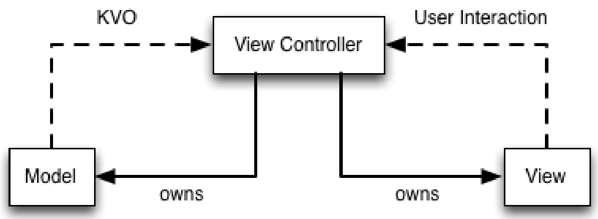
## Model-View-Controller

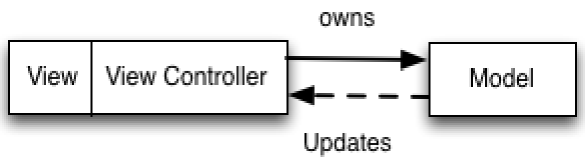
Structure:

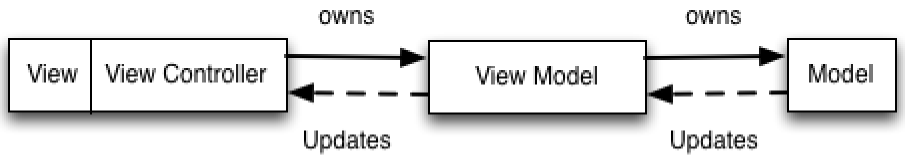


# Presentation Layer(2)

- MVVM (Weiterentwicklung des MVC Patterns)
  - Auslagern von Komplexität in ein View Modell (Testbarkeit)
    - (Two-Way) Databinding + Eventmechanismen als Bindeglieder
    - Bei modernen JS / AJAX basierten Uis hilfreich!



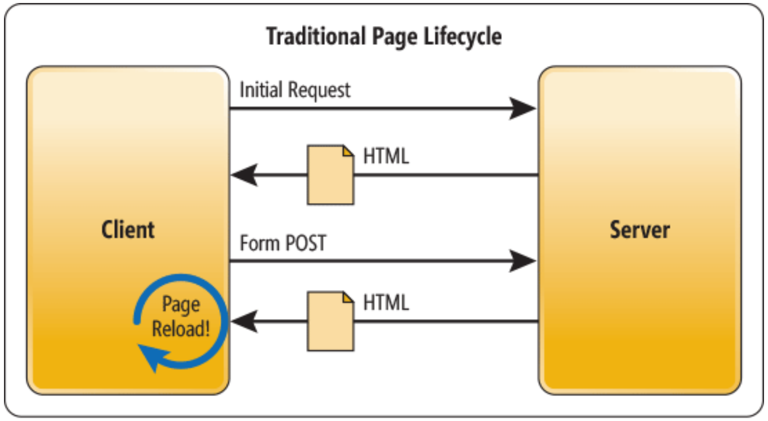


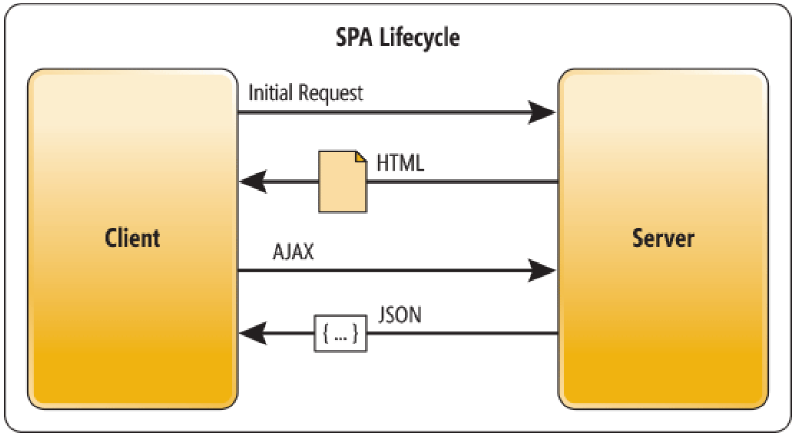


Link: http://www.objc.io/issue-13/mvvm.html

# *Presentation Layer (3)*

- Single Page Applications



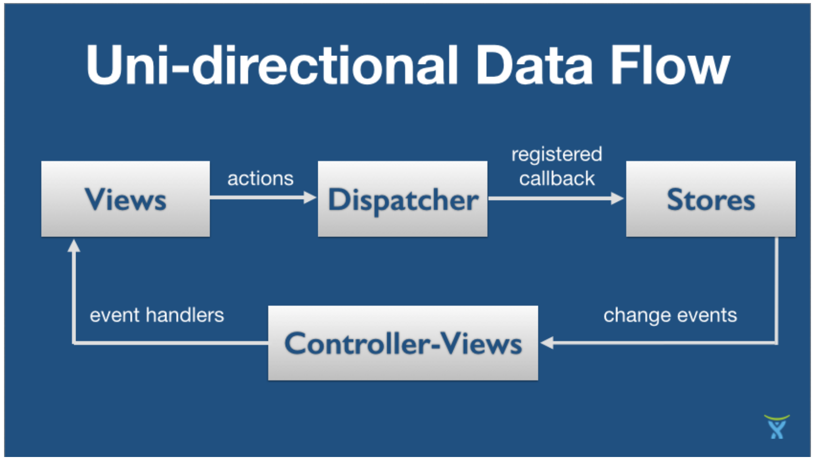


- Neue Art von Architektur (aktuell im Trend)
  - Technisch „eine Page“ am Client
  - Viel Logik auf den Client verlagert
  - Typisch: Ajax/ REST/ JSON mit Services
  - Achtung auf Usability / UI Design bei Business Applikationen (Mobile First)

# *Presentation Layer (4)*

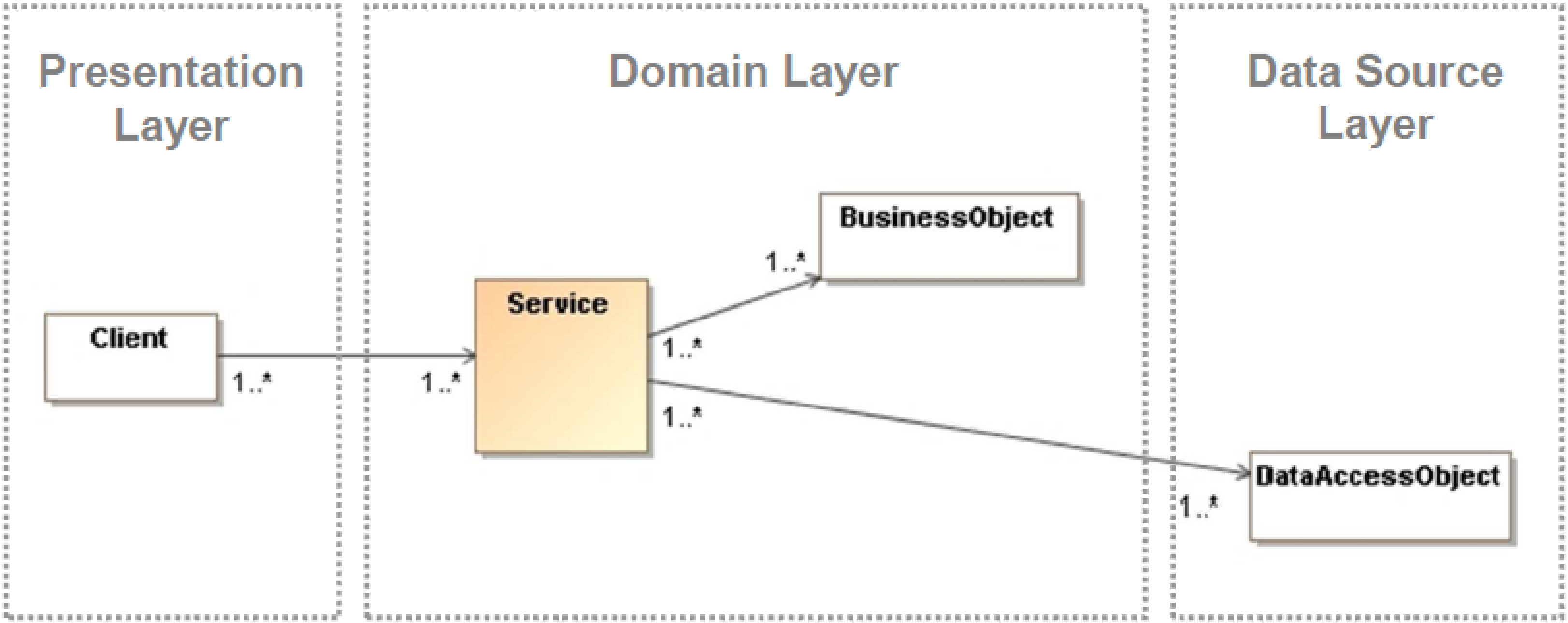
Der letzte Trend: Web-Komponenten + Unidirektionales Databinding

Bsp: Flux-Prinzip – (Facebook/Instagram)

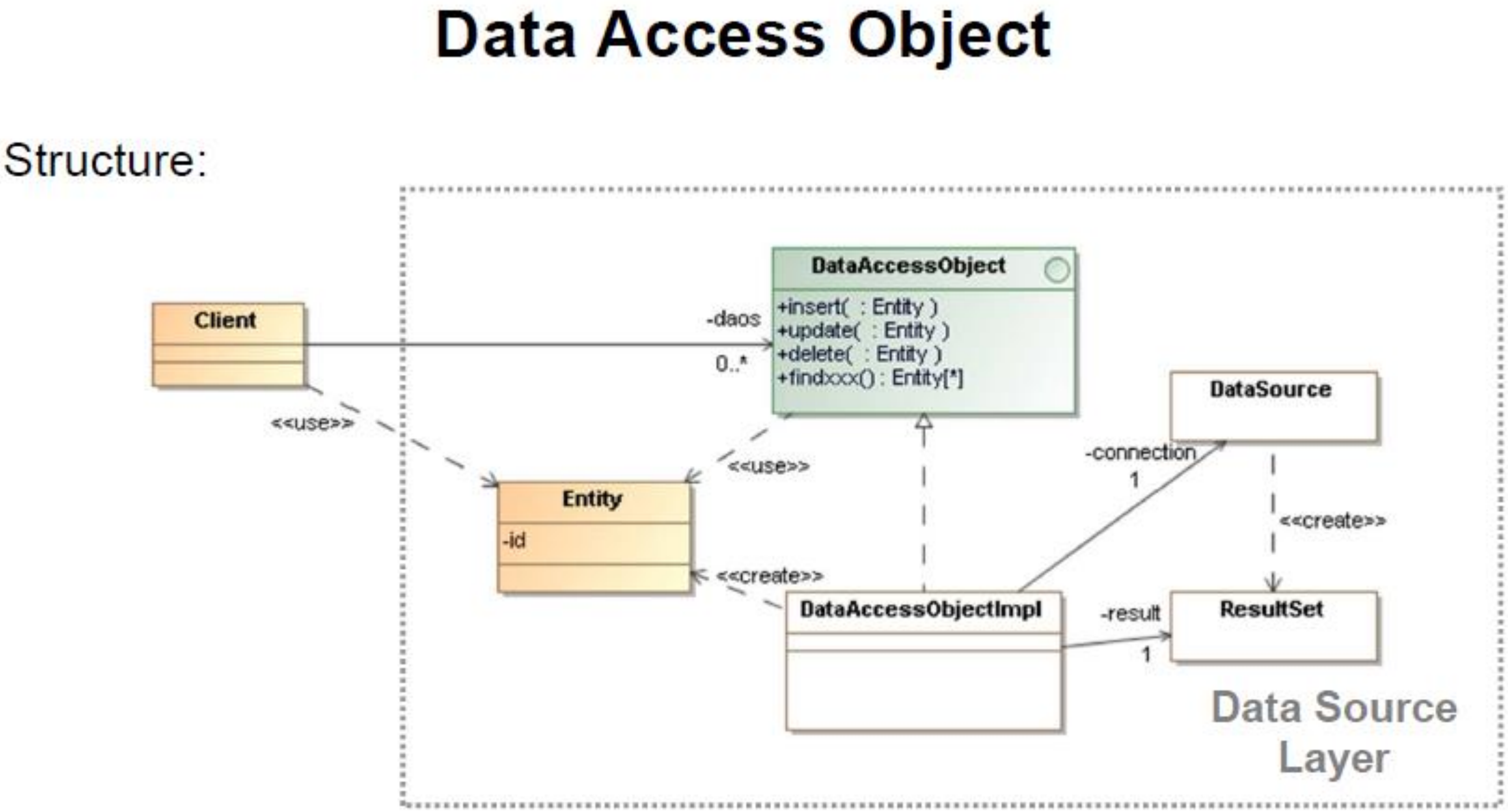


Bspe: React (Facebook) - (Unidirektional + Virtual DOM) (auch Angular2)

# *Service-  / Domainlayer*



# *Data Access Layer*



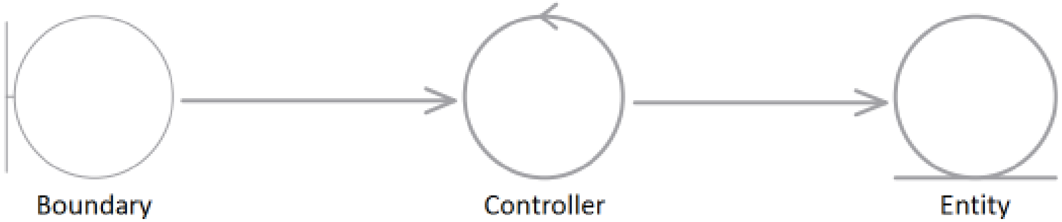
Data Access Object

Structure:

Achtung: Regeln definieren, was Servicelayer macht und was Dataaccess-Layer

# *Variante: Boundary Control Entity Pattern*

- Abgeleitet aus dem MVC Pattern für das Backend (Adam Bien)



| | | |
|---|---|---|
| Service | Komplettes Businesslogic | Datensicht |

Kernidee: Schneiden nach fachlichen Kriterien und nicht nach technischen Sichten

# *Typische Architektur Aspekte*

- Security (Authentifizierung, Autorisierung)
- Allg. Usability Regeln
- Internationalisierung
- Hilfesysteme …
- Sessionbehandlung
- **Transaktionsbehandlung**
- **Locking (Optimistisch, Pessimistisch)**
- **Einzelverarbeitung vs. Listenverarbeitung**
- Ausnahme-/ Fehlerbehandlung (Exceptionhandling)
- Logging, Protokollierung und Tracing (fachlich vs. technisch) …
- Verteilung (Deployment Artefakte)
- Konfigurierbarkeit
- **Reporting**
- …

Usability & Ergonomie

Technik

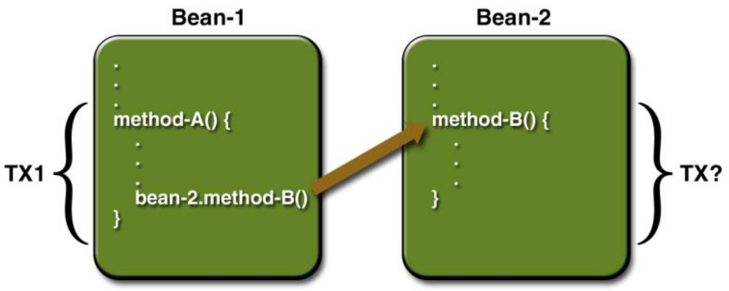
Betrieb

# *Bsp: A.-Aspekt Transaktionen*

**Ziele:**

- Festlegen der Transaktionsbehandlung für das System um den Anforderungen an fachliche Transaktionen zu genügen

**Methode:**



- Typische Festlegungen
  - Kontrolle nur im Businesslayer
  - Standardfall meist „Requires“
  - Bibliotheksauswahl (Bsp: JPA,…)
  - Schnittstelle zur programmatischen Kontrolle festlegen (Abstraktion von techn. Aspekten)
  - Richtlinien für das Team

# *Bsp: A-Aspekt Listenoperationen*

**Ziel**

- Erfüllen geforderter Antwortzeiten entsprechend der NFRs
- Massenverarbeitungen (Batchverarbeitung, Listen) oft nicht mit Standardmechanismen (O/R Mapping) erfüllbar
- Grenzen von O/R Mappern überwinden

**Methode**

- Typische Festlegungen
  - Einzelsatzverarbeitung (bspw. CRUD-Operationen) über Standardmechanismen der benutzten Umgebung (entitätsbezogene Datenobjekte, Composites)
  - Listenverarbeitung über optimierte Datenoperationen (bspw. flache Datenobjekte) oder optimierte Datenstrukturen (Tabellen)
  - Richtlinien für das Team

# *Bsp: A-Aspekt Locking*

**Ziel**

- Festlegen des Datensatz Lockings (Parallele Bearbeitung)

**Problem**

- Wird oft ignoriert und ist im Nachgang meist sehr teuer einzubauen

**Methode**

- Typische Festlegungen
  - Optimistisch (meist - kein Sperren) vs. Pessimistisch (Sperren des Datensatzes)
  - Beim Schreiben wird Version ausgewertet und bei Ungleichheit eine Fehlermeldung an den User.
  - Richtlinien für das Team

# *Fachliche Architektur festlegen*

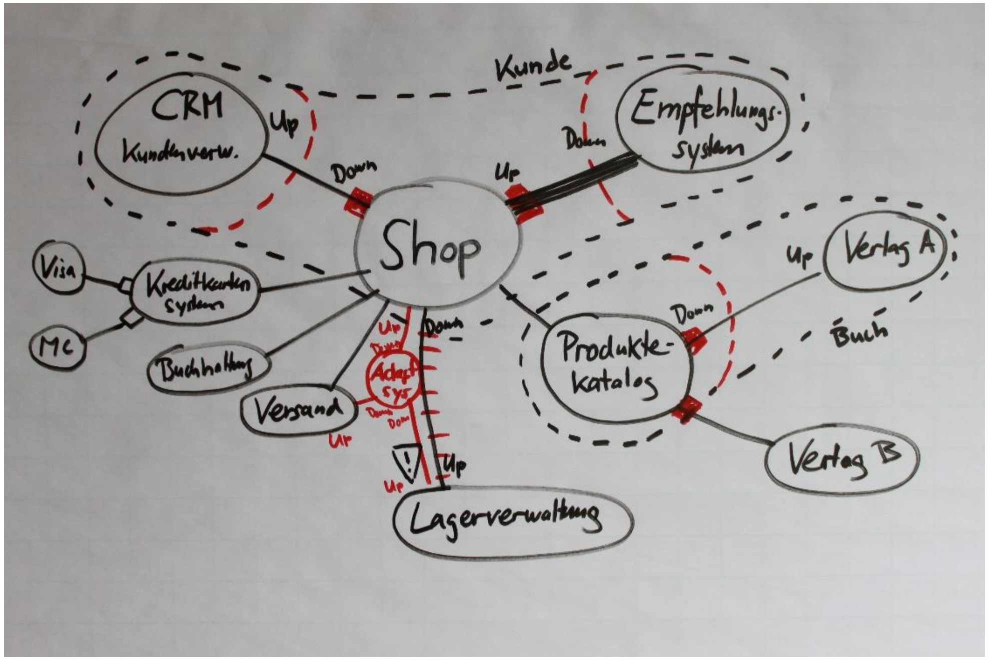
**Ziele:**

- Festlegen eines Domain Models
  - – Definition relevanter Business Objekte (Interfaces, Aufgaben, Regeln)
    - Wer hat die Hoheit über die Daten (Lesen /Schreiben)!

**Methode:**

- – Analyse der fachlichen Requirements
- – Hauptwörter in User Stories sind potentielle Business Objekte (User, Community, Stream, Services etc.)
- – Beachten: Klassen / Attribute / Relationen

# Domain Driven Design



Context Map

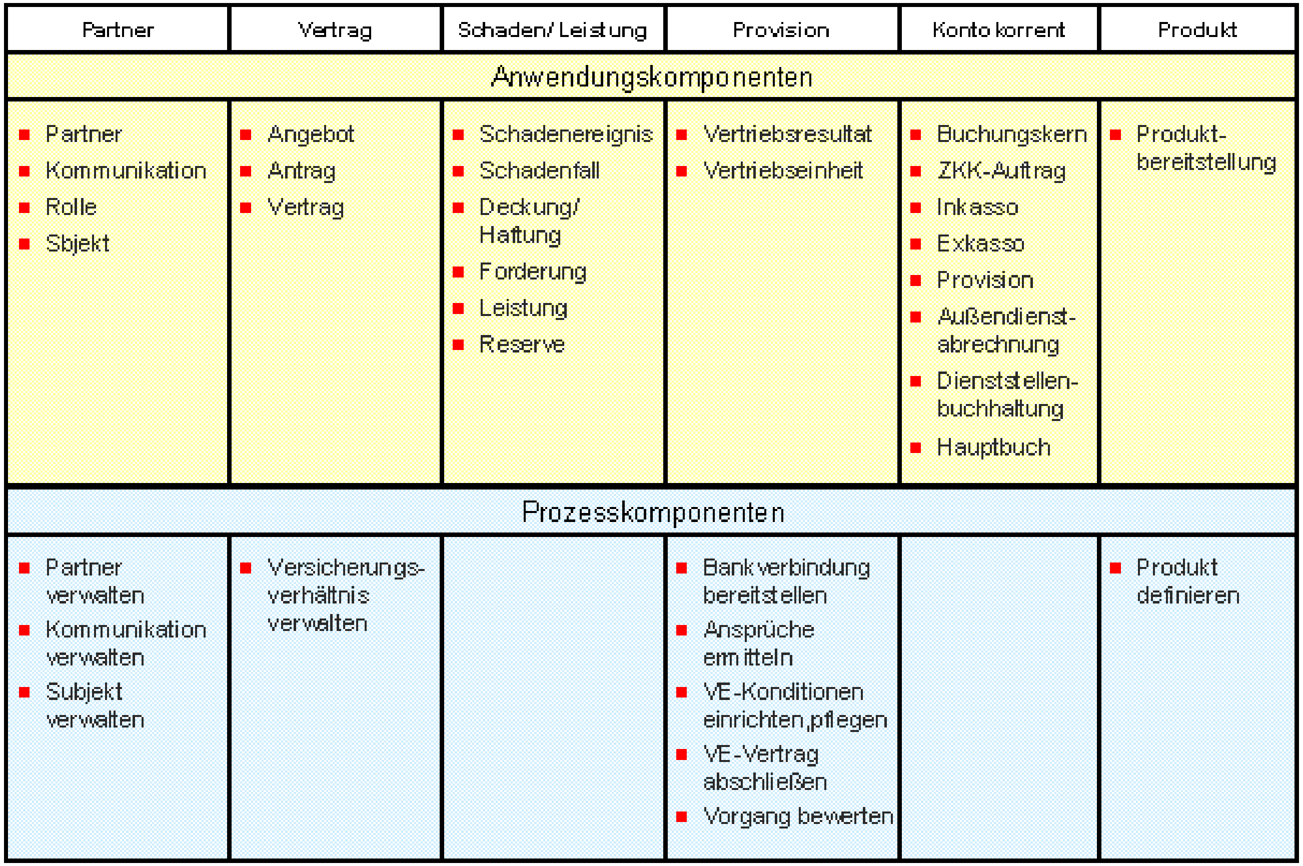
- Bounded Context (Gültigkeitsgrenzen der Domäne)

- Up: Master der Domäne
- Down: Übernimmt Domänen-strukturen

- Anti-Corruption Layer (Adapter als Entkopplung)

# Bsp: Beispiel für ein Domain Model

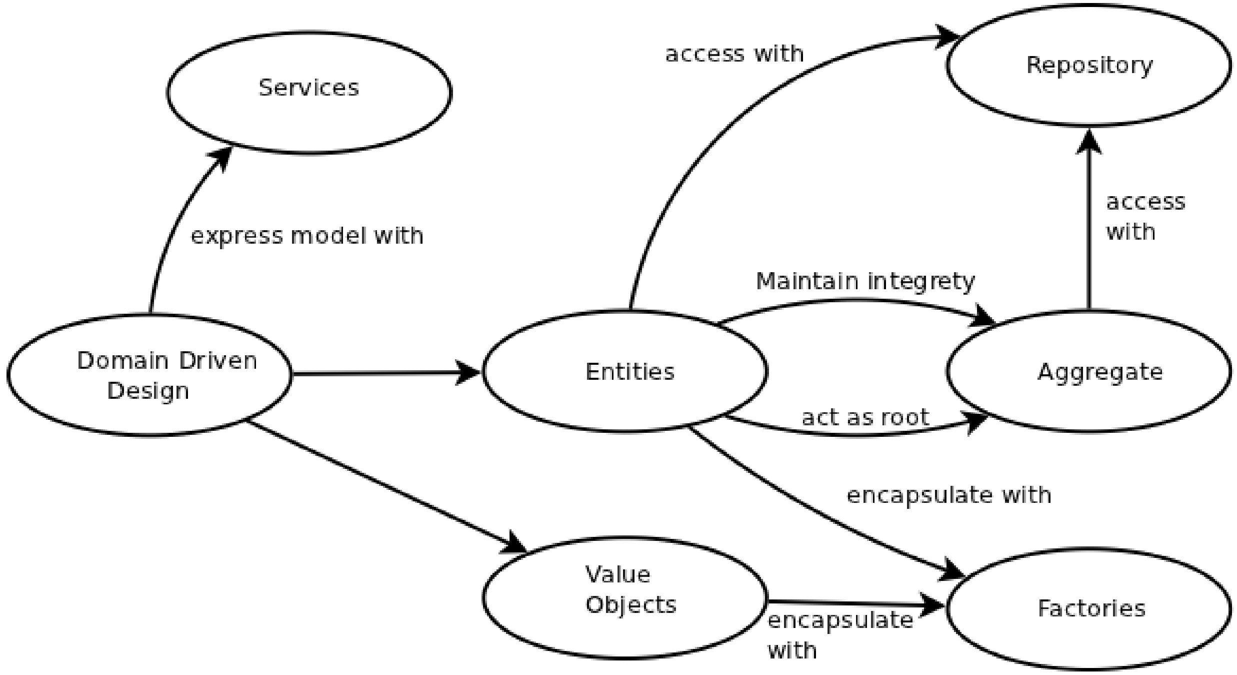
**Anmerkung:** Enthält nur Objects



| Partner | Vertrag | Schaden/ Leistung | Provision | Konto korrent | Produkt |
|---|---|---|---|---|---|
| Anwendungskomponenten | | | | | |
| ■ Partner<br>■ Kommunikation<br>■ Rolle<br>■ Sbjekt | ■ Angebot<br>■ Antrag<br>■ Vertrag | ■ Schadenereignis<br>■ Schadenfall<br>■ Deckung/ Haftung<br>■ Forderung<br>■ Leistung<br>■ Reserve | ■ Vertriebsresultat<br>■ Vertriebseinheit | ■ Buchungskern<br>■ ZKK-Auftrag<br>■ Inkasso<br>■ Exkasso<br>■ Provision<br>■ Außendienstabrechnung<br>■ Dienststellenbuchhaltung<br>■ Hauptbuch | ■ Produktbereitstellung |
| Prozesskomponenten | | | | | |
| ■ Partner verwalten<br>■ Kommunikation verwalten<br>■ Subjekt verwalten | ■ Versicherungsverhältnis verwalten | | ■ Bankverbindung bereitstellen<br>■ Ansprüche ermitteln<br>■ VE-Konditionen einrichten,pflegen<br>■ VE-Vertrag abschließen<br>■ Vorgang bewerten | | ■ Produkt definieren |

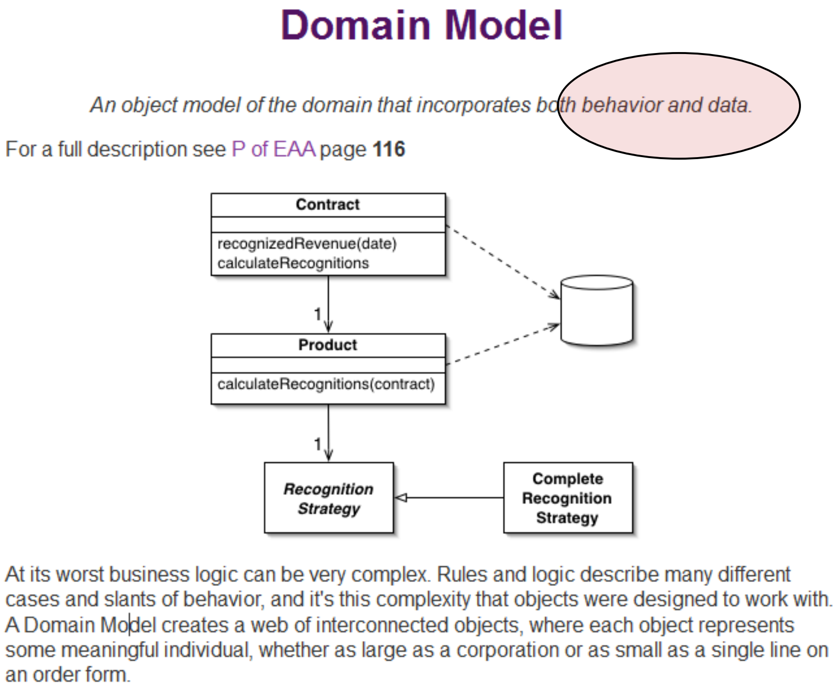
Quelle: http://www.gdv-online.de/vaa/vaafe_html/dokument/okompo.doc

# *Domain Driven Design*



# Beispiel für Domainmodel

- Fowler definiert in EAA:

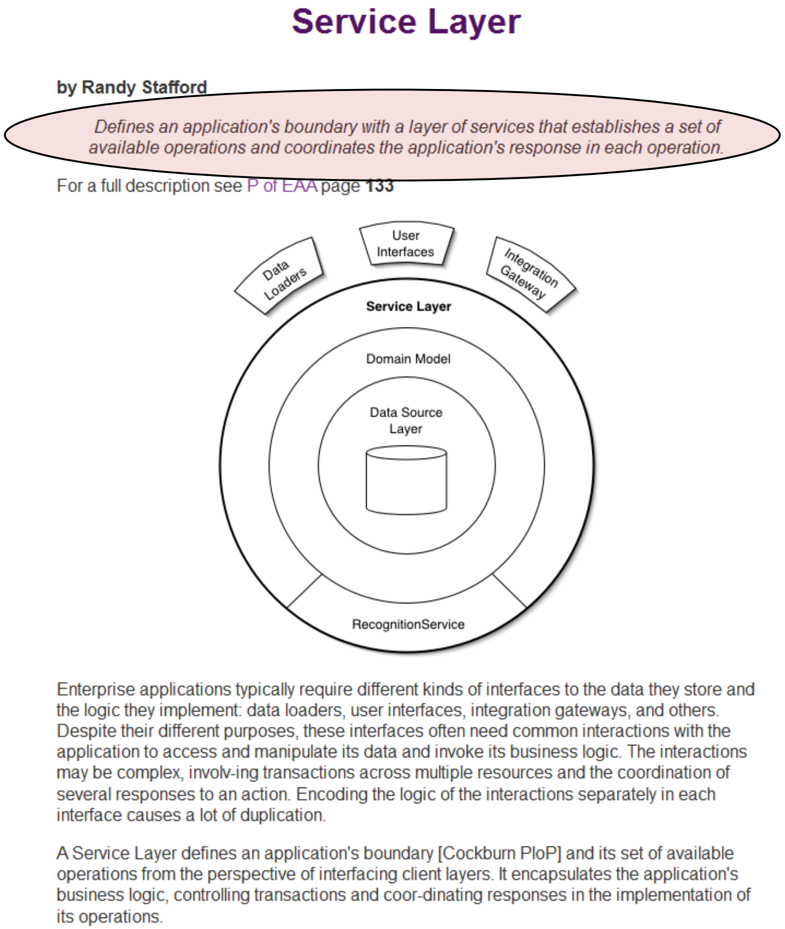


## Domain Model

*An object model of the domain that incorporates both behavior and data.*

For a full description see P of EAA page **116**

At its worst business logic can be very complex. Rules and logic describe many different cases and slants of behavior, and it's this complexity that objects were designed to work with. A Domain Model creates a web of interconnected objects, where each object represents some meaningful individual, whether as large as a corporation or as small as a single line on an order form.



## Service Layer

**by Randy Stafford**

*Defines an application's boundary with a layer of services that establishes a set of available operations and coordinates the application's response in each operation.*

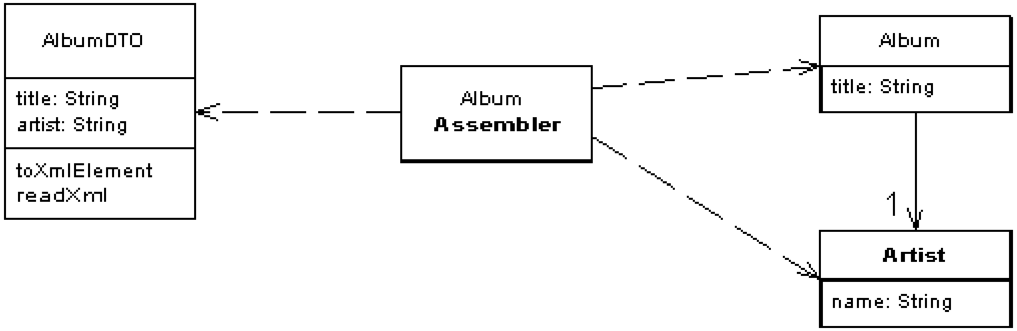
For a full description see P of EAA page **133**

Enterprise applications typically require different kinds of interfaces to the data they store and the logic they implement: data loaders, user interfaces, integration gateways, and others. Despite their different purposes, these interfaces often need common interactions with the application to access and manipulate its data and invoke its business logic. The interactions may be complex, involv-ing transactions across multiple resources and the coordination of several responses to an action. Encoding the logic of the interactions separately in each interface causes a lot of duplication.

A Service Layer defines an application's boundary [Cockburn PloP] and its set of available operations from the perspective of interfacing client layers. It encapsulates the application's business logic, controlling transactions and coor-dinating responses in the implementation of its operations.

# *Domainobjekte vs. DTOs*

- Idee: Optimierte Datentransferobjekte für die UI



Pragmatischer Tipp aus der Praxis

- Verwenden wo nötig

# *Exkurs – Datenmodellierung*

- Code First vs. DB-First
  - Code First ist aktuell im Trend
  - Achtung
    - Nicht nur Domainattribute festlegen (Attribute, Constraints,..)
    - Indizes nicht vergessen (Bsp: Suchen, komplexe Abfragen)

- Praktische Tipps bei Code-First
  - Visualisieren Sie dennoch ein ER- oder Objekt-Modell
  - Achten Sie auf Indizierung
  - Beachten Sie Update-Problematik im realen Umfeld wenn Release >1.0

# *Das alles klingt sehr aufwändig…*

… und wie passt das alles zu Agile?

## Aus dem Gelernten über „Agile Methoden“

- Kommunikation steht über Geschriebenem
- Keine explizite Architekturphase einziehen
  - Prototyping an realem Feature (value driven)
  - Feature muss in Iteration passen (kleine Einheiten)
- „Von allem ein bisschen was in jeder Iteration“
- „Jetzige Iteration im Detail mit Weitblick“
- KISS Prinzip -> XP: „You ain‘t Gona need it“ -> Refactoring

**Achtung: Nicht alle Entscheidungen sind reversibel (Kosten)!**

# *Vorgaben für Ihre Architektur*

- Presentationlayer mit JSF basiernd auf MVC
- Servicelayer mit Domainmodell
- Rest Service für User-Informatione aus Directory
- Data Accesslayer

# *Referenzen*

- Martin Fowler:
  **„Patterns of Enterprise Application Architecture”**, 2002, Addison Wesley

- Architektur Aspekte : ARC 42
  (Template selbst zu schwergewichtig für agil)

# Practical SE

## Continuous Integration

FH JOANNEUM

Software Design

Egon Teiniker

Version: 1.2.1

# *Continuous Integration*

## Outline

- Overview
- Steps in a Continuous Integration Scenario
- Practices of Continuous Integration
- Benefits of Continuous Integration

# Continuous Integration

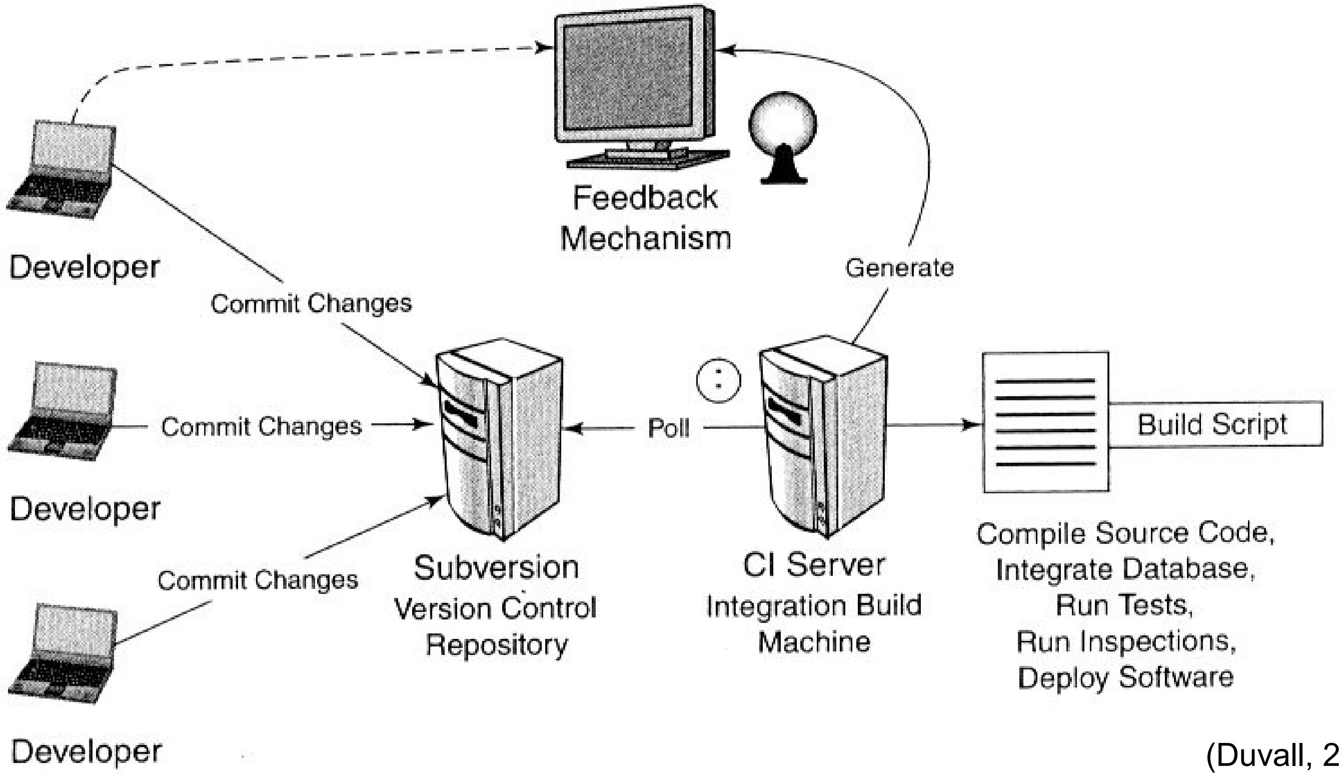
## Overview

**Continuous Integration (CI)** is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily – leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible.

Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.

# *Continuous Integration*

## Steps in a CI Scenario



(Duvall, 2007)

# *Continuous Integration*

## Steps in a CI Scenario

1. First, a developer commits code to the **version control repository**.
2. Soon after a commit occurs, the CI server detects that changes have occurred in the version control repository, so the **CI server retrieves the latest copy of the code** from the repository and **executes a build script**.
3. The CI server generates **feedback** (HTML reports, emails) for project members.
4. The **CI server continues to poll for changes** in the version control repository.

# *Continuous Integration*

## Practices of Continuous Integration

- Maintain a Single Source Repository.
  - Software projects involve lots of files that need to be orchestrated together to build a product. Keeping track of all of these is a major effort.
  - The current open source repository of choice is **Subversion**. (The older open-source tool CVS is still widely used)
  - Make sure it is the **well known place** for everyone to go get source code.
  - **Everything you need to do a build** should be in there including: test scripts, properties files, database schema, install scripts, and third party libraries.

## Practices of Continuous Integration

- Automate the Build.
  - Getting the sources turned into a running system can often be a **complicated process** involving compilation, moving files around, loading schemas into the databases, and so on.
  - Like most tasks in this part of software development it can be automated - and as a result **should be automated**.
  - Anyone should be able to bring in a virgin machine, check the sources out of the repository, issue **a single command**, and have a running system on their machine.
  - On a Java project we're okay with having developers build in their IDE, but the master build uses **Ant** to ensure it can be run on the CI server.

## Practices of Continuous Integration

- Make Your Build Self-Testing.
  - A good way to **catch bugs more quickly and efficiently** is to include automated tests in the build process.
  - For self-testing code you need a **suite of automated tests** that can check a large part of the code base for bugs.
  - The result of running the test suite should indicate if any tests failed. For a build to be self-testing **the failure of a test should cause the build to fail**.
  - **xUnit** tools are certainly the starting point for making your code self-testing.
  - Imperfect tests, run frequently, are much better than perfect tests that are never written at all.

## Practices of Continuous Integration

- Everyone Commits Every Day.
  - Integration is primarily about **communication**. Integration allows developers to tell other developers about the changes they have made.
  - With developers committing every few hours a **conflict can be detected within a few hours of it occurring**, at that point not much has happened and it's easy to resolve. Conflicts that stay undetected for weeks can be very hard to resolve.
  - **Every developer should commit to the repository every day**. In practice it's often useful if developers commit more frequently than that.

## Practices of Continuous Integration

- Every Commit Should Build the Mainline on an Integration Machine.
  - You shouldn't go home until the mainline build has passed with any commits you've added late in the day.
  - Many organizations do regular builds on a timed schedule, such as every night. The whole point of continuous integration is to find problems as soon as you can. **Nightly builds** mean that bugs lie undetected for a whole day before anyone discovers them.
  - A key part of doing a continuous build is that if the mainline build fails, it needs to **be fixed right away**.

## Practices of Continuous Integration

- Keep the Build Fast.
  - The whole point of Continuous Integration is to **provide rapid feedback**. Nothing sucks the blood of a CI activity more than a build that takes a long time.
  - For enterprise applications, **the usual bottleneck is testing** - particularly tests that involve external services such as a database.
  - Probably the most crucial step is to start working on setting up a **staged build**. The **commit build** is the build that's needed when someone commits to the mainline. The second-stage build is a **secondary build** which runs when it can, picking up the executable from the latest good commit build for further testing.

## Practices of Continuous Integration

- Test in a Clone of the Production Environment.
  - You want to set up your test environment to be as **exact a mimic of your production environment** as possible. Use the same database software, with the same versions, use the same version of operating system.
  - In reality there are limits. Some production environments may be prohibitively **expensive to duplicate**.
  - There is a growing interest in using **virtualization** to make it easy to put together test environments.

# *Continuous Integration*

## Practices of Continuous Integration

- Make it Easy for Anyone to Get the Latest Executable.
  - Anyone involved with a software project should be able to **get the latest executable and be able to run it**: for demonstrations, exploratory testing, or just to see what changed this week.
  - Make sure there's a well **known place** where people can find the latest executable.
  - It may be useful to put several executables in such a store. For the very latest you should put **the latest executable to pass the commit tests**.
  - If you are following a process with well defined iterations, it's usually wise to also put **the end of iteration builds** there too.

## Practices of Continuous Integration

- Everyone can see what’s happening.
  - Continuous Integration is all about communication, so you want to ensure that **everyone can easily see the state of the system** and the changes that have been made to it.
  - One of the most important things to communicate is **the state of the mainline build**.
  - CI servers' **web pages** can carry more information than this, of course.

# *Continuous Integration*

## Practices of Continuous Integration

- Automate Deployment.
  - It's important to have **scripts** that will allow you to deploy the application into any environment easily.
  - Automatic deployment helps both **speed up the process** and **reduce errors**.
  - If you deploy into production one extra automated capability you should consider is **automated rollback**.

# *Continuous Integration*

## Benefits of Continuous Integration

- The greatest and most wide ranging benefit of Continuous Integration is **reduced risk**.
- At all times **you know where you are**, what works, what doesn't, the outstanding bugs you have in your system.
- Continuous Integrations doesn't get rid of **bugs**, but it does make them dramatically **easier to find and remove**.
- As a result projects with Continuous Integration tend to have dramatically **less bugs, both in production and in process**.
- If you have continuous integration, it removes one of the biggest barriers to **frequent deployment**.

## Continuous Integration Roadmap

- Get everything you need into **source control**.
- One of the first steps is to **get the build automated**.
- Introduce some **automated testing** into you build.
- Try to **speed up the commit build**.

If you are starting a new project, begin with Continuous Integration from the beginning.

# *References*

- *Martin Fowler*
  **Continuous Integration**
  http://martinfowler.com/articles
  Last significant update: May 2006

- *Paul M. Duvall*
  **Continuous Integration**
  **Improving Software Quality and Reducing Risk**
  Addison-Wesley, 2007

# *Outline*

## CI Anti-Patterns

- Infrequent check-in
- Bottleneck commits
- IDE-only build
- Works on my machine
- Broken build
- Bloated build
- Myopic environment
- Polluted environment
- Slow machine
- Continuous ignorance
- Scheduled builds
- Minimal or spam feedback

# *CI Anti-Patterns*

## **Infrequent Check-in**

*Anti-pattern:*

Source files stay checked out of a repository for long periods of time due to the amount of changes required for tasking.

*Solution:*

**Commit smaller chunks of code frequently**.

# *CI Anti-Patterns*

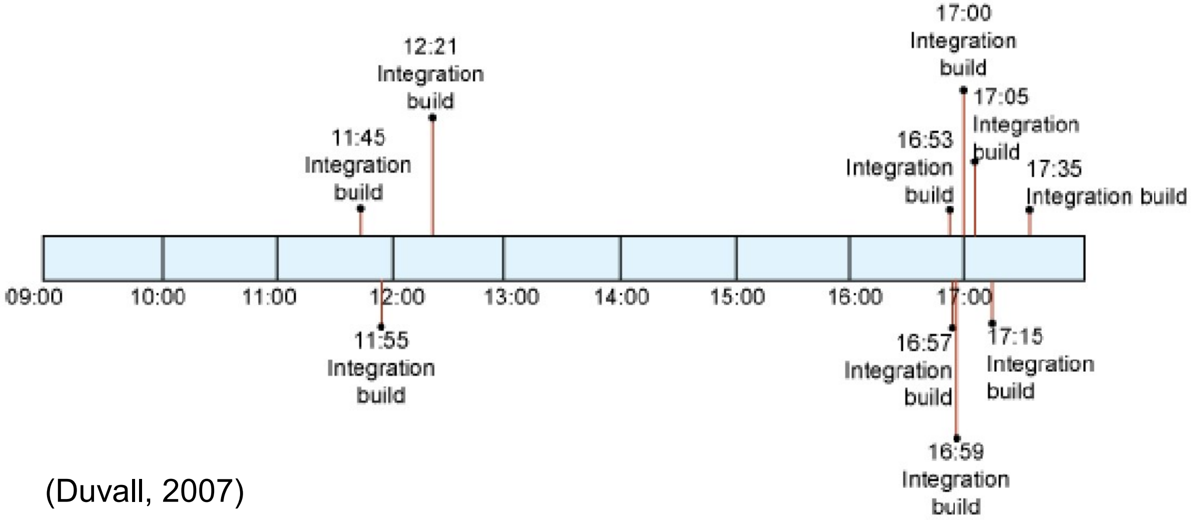
## Bottleneck Commits

*Anti-pattern:*

Developers commit code changes prior to leaving for the day, causing integration build errors and preventing team members from going home at a decent time.

*Solution:*

**Check-in code frequently throughout the day**.



(Duvall, 2007)

# *CI Anti-Patterns*

## IDE-only Build

*Anti-pattern:*

You run a private build using an IDE (that works locally) on your workstation only to discover things don't work in another environment.

*Solution:*

**Create a build script** and commit it into a version control repository. Run this same build script with every change.

# *CI Anti-Patterns*

## Works on My Machine

*Anti-pattern:*

You run a private build that works on your machine, only to discover later that the changes don't work in other environments.

*Solution:*

The **team uses an integration build machine** that runs a build with every change committed to a version control repository.

# *CI Anti-Patterns*

## Broken Build

*Anti-pattern:*

Builds stay broken for long periods of time, thus preventing developers from checking out functioning code.

*Solution:*

Developers are immediately notified upon a build breakage and make it a **top priority to fix a broken build**.

# *CI Anti-Patterns*

## Bloated Build

*Anti-pattern:*

Throwing everything into the commit build process, such as running every type of automated inspection tool or running load tests such that feedback is delayed.

*Solution:*

A **build pipeline** enables running different types of builds.

# *CI Anti-Patterns*

## Myopic Environment

*Anti-pattern:*

Assuming that because a build works in one environment, it'll work in any environment.

*Solution:*

Define build behavior in build targets; moreover, externalize environment-dependent data into **.properties files**.

# *CI Anti-Patterns*

## Polluted Environment

*Anti-pattern:*

An incremental build is run to save time; however, an older artifact (from a previous build) produces a false positive (or false negative) build.

*Solution:*

**Clean previously built artifacts** prior to running a build. Baseline server and configuration information.

# *CI Anti-Patterns*

## Slow Machine

*Anti-pattern:*

A workstation with limited resources is used as the build machine, leading to lengthy build times.

*Solution:*

The build machine has **optimal disk speed, processor, and RAM resources** for speedy builds.

# *CI Anti-Patterns*

## Continuous Ignorance

*Anti-pattern:*

When a build was successful, everyone assumes the resulting software is working fine. In reality, the limited build consists of compilation and a few unit tests.

*Solution:*

**A full integration build** is run with every change to the version control repository.

# *CI Anti-Patterns*

## Scheduled Builds

*Anti-pattern:*

Builds are run daily, weekly, or on some other schedule, but not with every change.

*Solution:*

**A build is run with every change** applied to a source code repository.

# *CI Anti-Patterns*

## Minimal Feedback

*Anti-pattern:*

Teams choose not to send build status notifications to team members; thus, people aren't aware a build has failed.

*Solution:*

**Use various feedback mechanisms** to relate build status information.

# *CI Anti-Patterns*

## Spam Feedback

*Anti-pattern:*

Team members quickly become inundated with build status e-mails (success and failure and everything in between) to the point where they start to ignore messages.

*Solution:*

**Feedback is succinctly targeted** so that people don't receive irrelevant information.

# *FAQs*

## Continuous Integration

- Explain the term Continuous Integration (CI).
- Describe the steps in a typical continuous integration scenario (including sketch).
- Describe the important CI practices.
- Describe discussed advantages of using CI.
- Describe known CI anti patterns.

# References

- *Paul Duvall*
  **Automation for the people:**
  **Continuous Integration anti-patterns**
  IBM Developer Works, 2007

- *Paul Duvall*
  **Automation for the people:**
  **Continuous Integration anti-patterns, Part 2**
  IBM Developer Works, 2008

# *Practical SE*

## *Agile Testing*

FH JOANNEUM

Software Design

Egon Teiniker

Version: 1.0.1

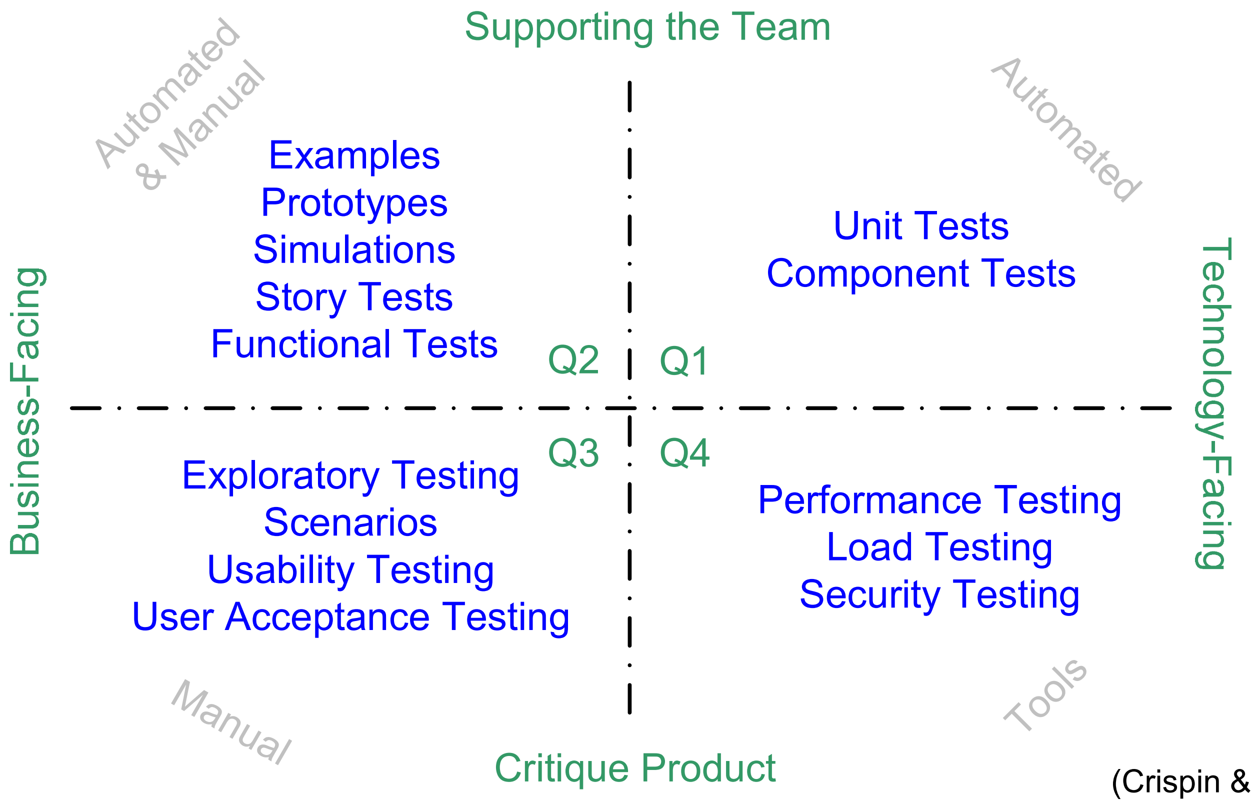
# *Agile Testing*

## Outline

- Agile Testing Quadrants
- Roadmap to Test Automation
- Test Automation Difficulty

# *Agile Testing*

## Agile Testing Quadrants



Supporting the Team

Automated & Manual

Examples
Prototypes
Simulations
Story Tests
Functional Tests

Q2

Q1

Automated

Unit Tests
Component Tests

Business-Facing

Technology-Facing

Q3

Q4

Exploratory Testing
Scenarios
Usability Testing
User Acceptance Testing

Performance Testing
Load Testing
Security Testing

Manual

Tools

Critique Product

(Crispin & Gregory, 2009)

## Agile Testing Quadrants

We can divide the testing activities into four quadrants:

**Q1: Test-driven development**

- Unit tests verify functionality of a small subset of the system (methods and objects).
- Component tests verify the behaviour of a larger part of the system (group of classes).
- Unit and component tests are automated and written in the same programming language as the application.
- Unit and component tests are implemented by the **team** (the tester supports the team by defining test cases).

## Agile Testing Quadrants

**Q2: Functional tests**

- Functional tests which are customer tests define external quality and the features that the customer (PO) wants. These tests also drive development, but at a higher level.
- These automated tests run directly on the business logic in the production code without having to go through a presentation layer.
- Functional tests are implemented by the **team** based on examples and prototypes and test cases provided from the PO and the tester.

## Agile Testing Quadrants

**Q3: GUI tests**

- These business-facing tests exercise the working software to see if it doesn’t quit meet expectations. We try to emulate the way a real user would work the application.
- This is mainly manual testing that only a human can do (use automated scripts to setup the data we need).
- GUI tests are performed by the **tester**.

## Agile Testing Quadrants

**Q4: Non-functional tests**

- These tests are intended to critique product characteristics such as performance, robustness, and security.
- Creating and running these tests might require the use of specialized tools and additional expertise.
- This job is done by the **tester** (usually the tester needs help from the team in technological issues).

## Test Automation Difficulty

The following common kinds of tests are listed in order of difficulty, from easiest to most difficult:

**Simple entity objects:**

- Simple classes with no dependencies.
- Complex classes with dependencies.

**Service objects:**

- Individual components.
- The entire business logic layer.
- Customer tests via Service Facade.

## Test Automation Difficulty

**"Hard-to-test" code:**

- User interface logic.
- Database logic.
- Multi-threaded software.

**Object-oriented legacy software**
(software built without any tests)

**Non-object-oriented legacy software.**

The irony is that many teams start automatic testing by trying to implement tests onto an existing application…

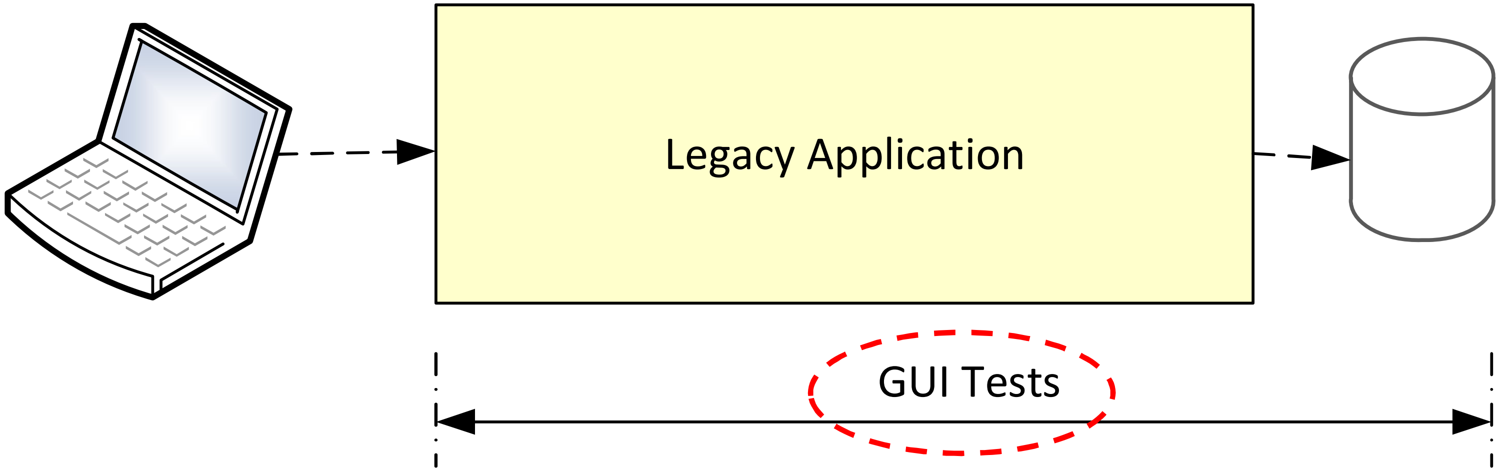
# *Legacy Architecture Testing*

## Outline

- Automate GUI Tests
- Add API and Functional Tests
- Add Interfaces and Unit Tests

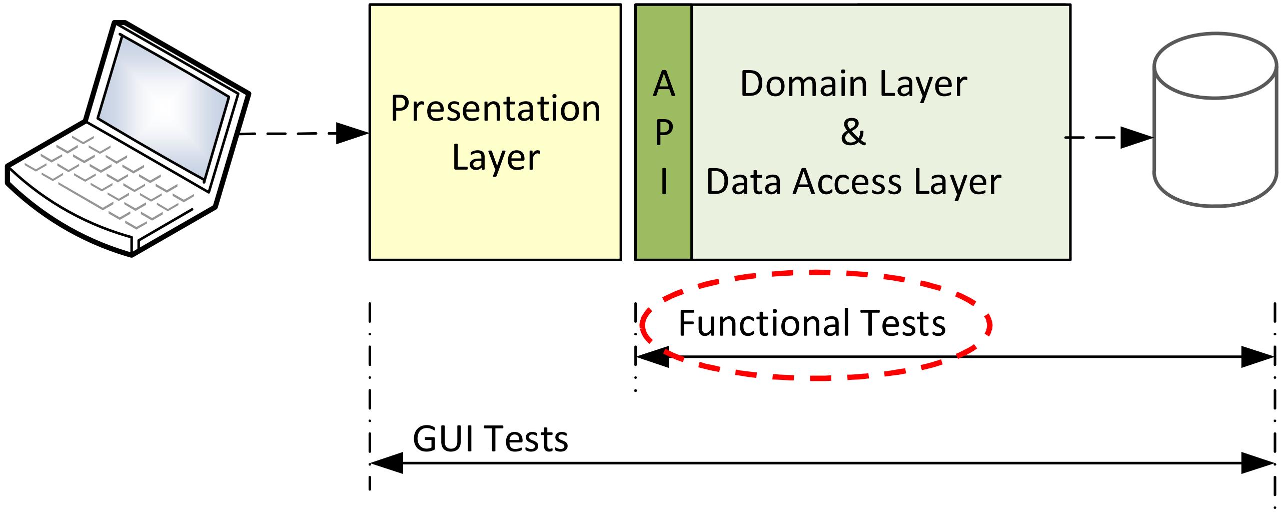
# *Legacy Architecture Testing*

## Step 1: Automate GUI Tests



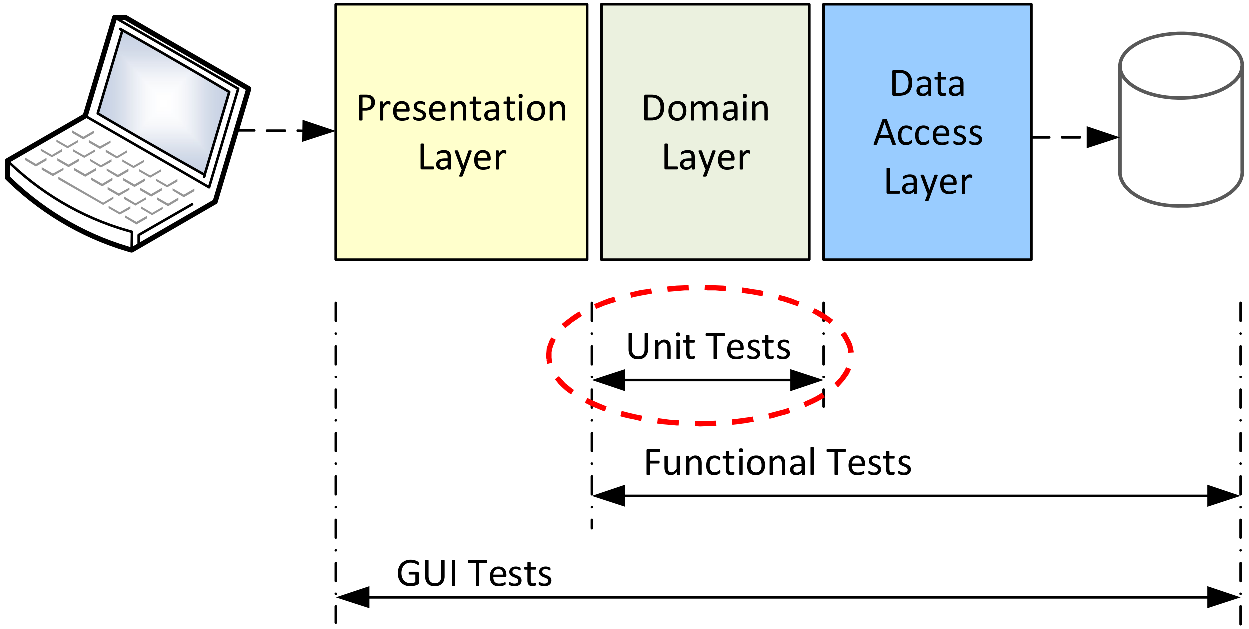
# *Legacy Architecture Testing*

## Step 2: Add API and Functional Tests



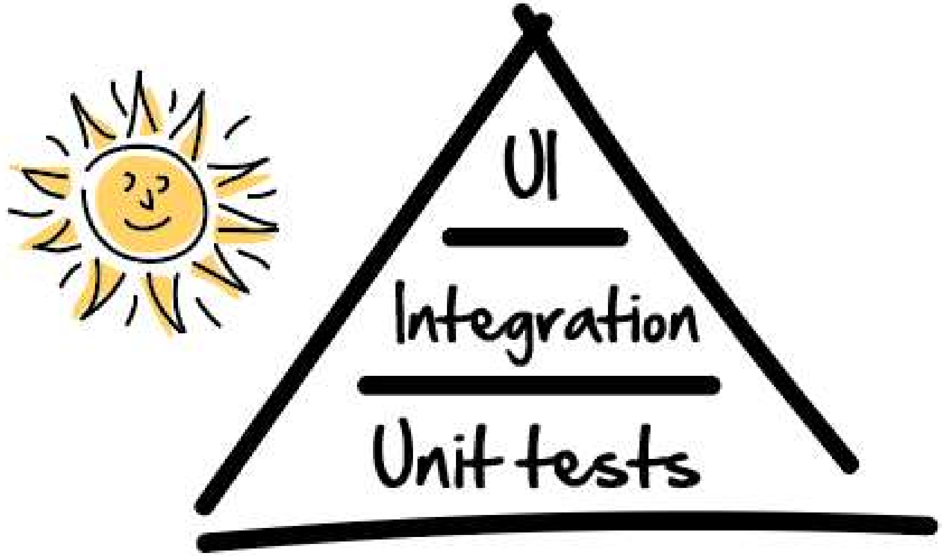
# *Legacy Architecture Testing*

## Step 3: Add Interfaces and Unit Tests



# *Legacy Architecture Testing*

## Remember: The Test Pyramid



(http://www.agilenutshell.com/episodes/41-testing-pyramid)

# References

- *Ken Schwaber*
  **Agile Project Management with Scrum**
  Microsoft Press, 2004

- *Lisa Crispin, Janet Gregory*
  **Agile Testing**
  Addison Wesley 2009

- *Gerard Meszaros*
  **xUnit Test Patterns**
  Addison-Wesley, 2007

# *Practical Software Engineering*

## *Web Application Security*

FH JOANNEUM

Software Design

Egon Teiniker

Version: 1.0.0

# *HTTPS Overview*

## Outline

- Introduction
- HTTPS Overview
- Server Certificates
- Site Certificate Validation

# *HTTPS Overview*

## Introduction

The Web requires a secure form of HTTP which provides:

- **Server authentication**
  Clients know they are talking to the real server.
- **Client authentication**
  Servers know they are talking to the real user.
- **Integrity**
  Clients and servers are safe from their data being changed.
- **Encryption**
  Clients and servers talk privately without fear of eavesdropping.

# *HTTPS Overview*

## Introduction

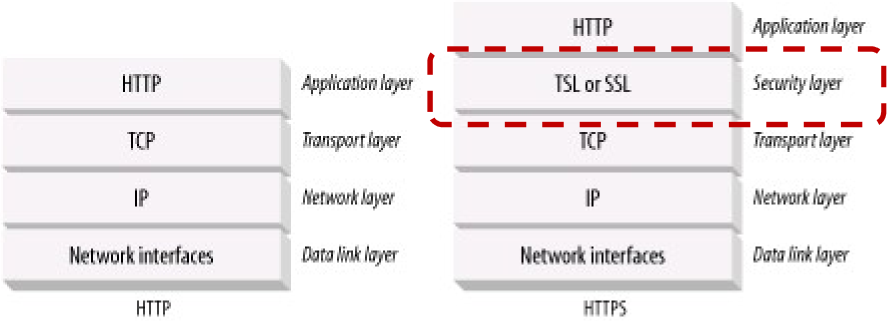
A secure form of HTTP which provides: (continued)

- **Efficiency**
  An algorithm fast enough for inexpensive clients and servers to use.
- **Ubiquity**
  Protocols are supported by nearly all clients and servers.
- **Administrative scalability**
  Instant secure communication for anyone, anywhere.
- **Adaptability**
  Supports the best known security methods of the day.

# *HTTPS Overview*

## Introduction

When using HTTPS, all the HTTP request and response data is encrypted before being sent across the network.



HTTPS works by providing a transport-level cryptographic security layer – using either the **Secure Sockets Layer (SSL)** or its successor, **Transport Layer Security (TLS)** – underneath HTTP.

# *HTTPS Overview*

## Introduction

HTTPS combines the HTTP protocol with a powerful set of **symmetric, asymmetric, and certificate-based cryptographic techniques**.

- If the URL has an **http** scheme, the client opens a connection to the server on **port 80** and sends its HTTP commands.
- If the URL has an **https** scheme, the client opens a connection to the server on **port 443** and “handshakes” with the server.

# *HTTPS Overview*

## SSL Handshake

Before sending encrypted HTTP messages, the client and the server need to do an SSL handshake, where they:

- Exchange protocol version numbers
- Select a cipher that each side knows
- Authenticate the identity of each side
- Generate temporary session keys to encrypt the channel

# *HTTPS Overview*

## Server Certificates

**SSL supports mutual authentication**, carrying server certificates to clients and carrying client certificates back to servers.

Client certificates are not commonly used for browsing, but **secure HTTPS transactions always require server certificates**.

The server certificate is an **X.509 v3** – derived certificate showing to organization’s name, address, server DNS domain name, and other information.

# *HTTPS Overview*

## Site Certificate Validation

Most modern browsers do some simple sanity checks on certificates based on an algorithm proposed by Netscape.

These checks includes the following steps:

- **Date check**
  The browser checks the certificate's start and end dates to ensure the certificate is still valid.
- **Signer trust check**
  Every certificate is signed by some certificate authority (CA), who vouches for the server. Browsers ship with a list of signing authorities that are trusted.

# *HTTPS Overview*

## Site Certificate Validation

These checks includes the following steps: (continued)

- **Signature check**
  The browser checks the certificate's integrity by applying the signing authority's public key to the signature and comparing it to the checksum.
- **Site identity check**
  Most browsers try to verify that the domain name in the certificate matches the domain name of the server they talked to.

# *Authentication*

## Outline

- Introduction
- Authentication Technologies
- Secure Password Storage

# *Authentication*

## Introduction

Authentication is the process of establishing and verifying an identity.

It is a way for an application **to confirm that you are who you say you are**. This is the most fundamental element to application security.

In order for a user to provide his identity, he must provide **credentials**.

# *Authentication*

## Introduction

We know three different methods of proving an identity:

- **Something you know**
  *Example: Passwords or security questions*
  *(e.g. if you forgot your password)*
- **Something you have**
  *Example: Hardware tokens*
- **Something you are**
  *Example: Biometrics like fingerprint readers, retina scanners.*

# *Authentication*

## Introduction

In the typical case, a user supplies his **username** and **password**, and the application must verify that these items are correct.

Authentication is the front line of defense against unauthorized access. In real-world applications authentication often is the **weakest link**, which enables an attacker to gain unauthorized access.

Many of the most common authentication vulnerabilities are no-brainers. Anyone can type dictionary words into a login form in attempt to guess valid passwords…

## Authentication Technologies

Web developers can use a wide range of technologies to implement authentication mechanisms:

- **HTML forms-based authentication**
  By far the most common authentication mechanism used by Web applications are HTML forms to capture a username and password and submit these to the application (>90%).
- **Multifactor mechanisms**
  In more security-critical Web applications (e.g. online banking) the forms-based mechanism is often expanded into multiple stages, requiring the user to submit additional credentials, such as a **PIN** or characters from a secret word.

## Authentication Technologies

- **Client SSL certificates and/or smartcards**
  Some Web applications use client-side SSL certificates or cryptographic mechanisms implemented within smartcards. They are typically used only in security-critical contexts where an application’s user base is small (e.g. **VPNs** for remote office workers).
- **HTTP basic and digest authentication**
- **Windows-integrated authentication**
  The HTTP-based authentication mechanisms (basic, digest) and NTLM or Kerberos are rarely used on the Internet.
  They are more common in intranet environments.

## Secure Password Storage

- **Hashing a Password**
  1. Generate a random salt
  2. Concatenate salt + password
  3. Hash (salt + password)
  4. Concatenate (salt + hash(salt + password)) = hash value

  In step 3, we can run the hash algorithm 100.000 times to **increase the calculation time**.

  Using this procedure, we get **different hash values for the same password** because the salt is generated randomly.

## Secure Password Storage

- **Checking a Password**
  1. Extract salt from hash
  2. Concatenate salt + password
  3. Hash (salt + password)
  4. Compare the both hash values

  We have to calculate the hash value (using the same salt)
  for the entered password before we can compare the values.

# *Session Management*

## Outline

- Introduction
- Session Management Technologies

# *Session Management*

## Introduction

HTTP was designed to be stateless.

Modern Web applications need to keep track of what users are currently connected to them (**session**) and the data associated with those sessions (**state**).

The session management mechanism provides a rich source of potential vulnerabilities from jumping into other user’s sessions to hijacking an administrator’s session to compromise the entire application.

# *Session Management*

## Session Management Technologies

The most convenient way to keep track of a session is for a server to issue each client a **session token**- a unique number that identifies that client.

**With each request the client sends back this token**, which the server can then use to associate with existing state information for that particular client.

In most cases, applications use **HTTP cookies** as the transmission mechanism for passing these session tokens between the server and client.

# *Session Management*

## Session Management Technologies

Some security-critical applications use **alternative techniques to manage state**.

- **HTTP Authentication**

  Using HTTP authentication (basic, digest, NTLM), the client component interacts with the authentication mechanism directly via the browser, using HTTP headers, and not via application-specific code.

  After the user enters his credentials into a browser dialog, **the browser resubmits these credentials with every subsequent request** to the same server.

# *Session Management*

## Session Management Technologies

- **Sessionless State Mechanisms**

  Some applications **transmit all data required to manage that state via the client**, usually in a cookie or a hidden field (like ASP.NET ViewState).

  To be secure, the data transmitted via the client must be properly protected.

  The application may also include an expiration time within the state object's data to perform the equivalent of session timeouts.

# *Authorization*

## Outline

- Introduction
- Access Control Technologies

# *Authorization*

## Introduction

Authorization is logically build on top of authentication
and session management.

An application needs a way to decide whether it should permit a given request to perform its attempted action or access the resources it is requesting – we call that **authorization**.

Authorization vulnerabilities are conceptually simple:
The application lets an attacker do something
he or she shouldn't be able to.

## Authorization Technologies

In the context of JEE, the **Java Authentication and Authorization Service (JAAS)** is used. The component containers are responsible for providing application security.

A container provides two types of security:

- **Declarative security** defines an application component’s security requirements by means of deployment descriptors and / or annotations. A deployment descriptor is an external file and can be modified without the need to recompile the source code:
  - Enterprise JavaBeans deployment descriptor: META-INF/ejb-jar.xml
  - Web application deployment descriptor: WEB-INF/web.xml

# *Authorization*

## Authorization Technologies

A container provides two types of security: (continued)

- **Programmatic** security is used when security checks are embedded within an application code. It can be used when declarative security alone is not sufficient to express the security model of an application.

  *Examples:* Java EE security API

  - isUserInRole() for Servlets and JSP
  - isCallerInRole() for EJB

# Authorization

## Authorization Technologies

As an alternative to JAAS, **Spring Security** ships with a robust authorization module which can serve the needs of small to medium-sized implementations reasonably well.

*Examples:* Common built-in expressions

hasRole([role]) Returns true if the current principal has the specified role.

hasAnyRole([role1,role2]) Returns true if the current principal has any of the supplied roles (given as a comma-separated list of strings)

isAnonymous() Returns true if the current principal is an anonymous user

# *References*

- *OWASP*
  **Authentication Cheat Sheet**
  **Password Storage Cheat Sheet**
  **Transport Layer Protection Cheat Sheet**
  https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series

- *Dafydd Stuttard, Marcus Pinto*
  **The Web Application Hacker's Handbook**
  Chapter 6: Attacking Authentication
  Wiley, 2nd Edition, 2011

# References

- *OWASP*
  **Session Management Cheat Sheet**
  `https://www.owasp.org/index.php/Session_Management_Cheat_Sheet`

- *Dafydd Stuttard, Marcus Pinto*
  **The Web Application Hacker's Handbook**
  Chapter 7: Attacking Session Management
  Wiley, 2nd Edition, 2011

# *Practical Software Engineering*

Testen von Webapplikationen mit Selenium

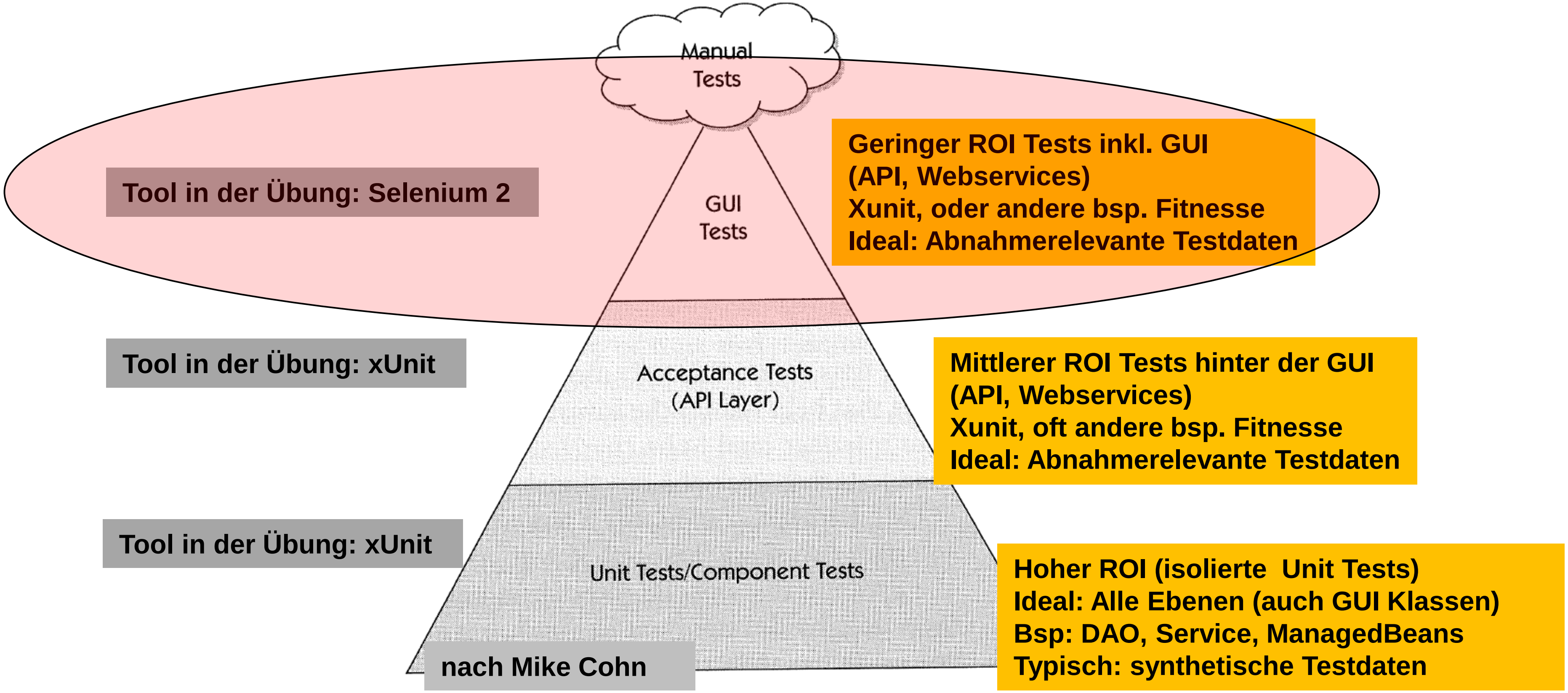
FH JOANNEUM
Internettechnik
Software Design
WS 2018

# *Agenda*

## **Testen von Webapplikationen mit Selenium**

- Intro Selenium
- Selenium 3.0 (Webdriver) mit Demo

# *Zur Erinnerung: Testpyramide*



**Tool in der Übung: Selenium 2**

**Geringer ROI Tests inkl. GUI**
**(API, Webservices)**
**Xunit, oder andere bsp. Fitnesse**
**Ideal: Abnahmerelevante Testdaten**

**Tool in der Übung: xUnit**

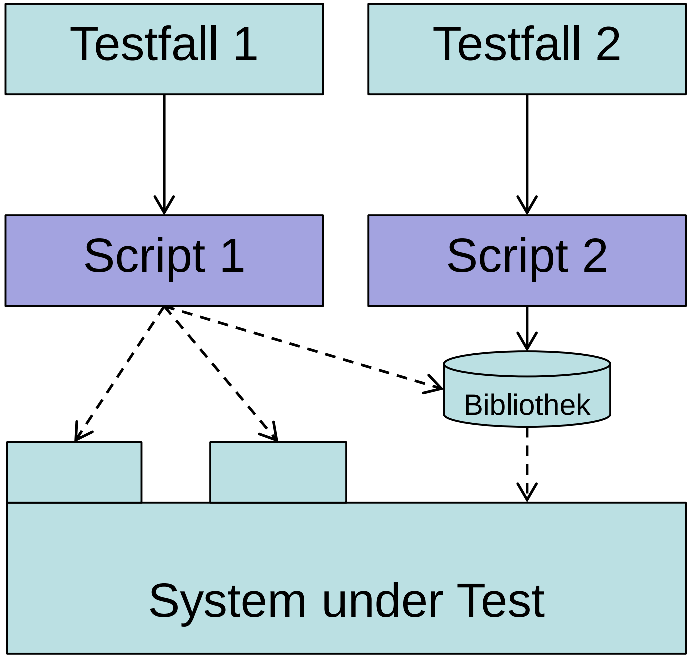
**Mittlerer ROI Tests hinter der GUI**
**(API, Webservices)**
**Xunit, oft andere bsp. Fitnesse**
**Ideal: Abnahmerelevante Testdaten**

**Tool in der Übung: xUnit**

**Hoher ROI (isolierte Unit Tests)**
**Ideal: Alle Ebenen (auch GUI Klassen)**
**Bsp: DAO, Service, ManagedBeans**
**Typisch: synthetische Testdaten**

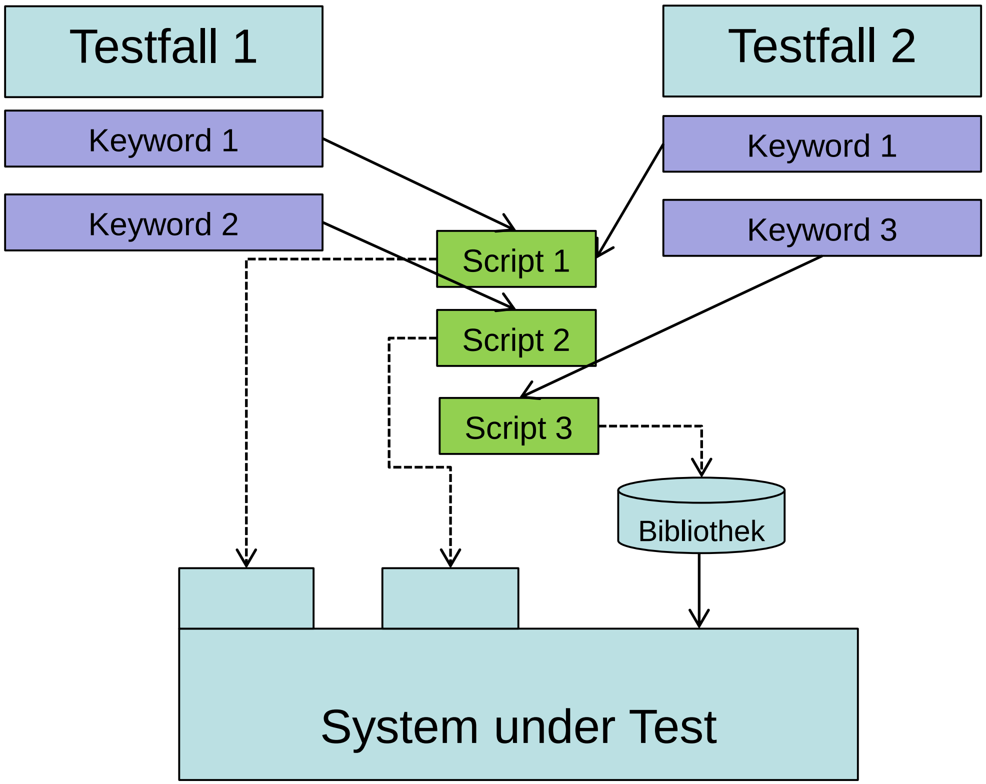
**nach Mike Cohn**

# *Klassisches Testmodell*



- Jeder Testfall ein Skript
- Wiederholung in Bibliotheken ausgelagert
- Probleme
  - Viele Skripts
  - Redundante Skripts
  - Wartbarkeit

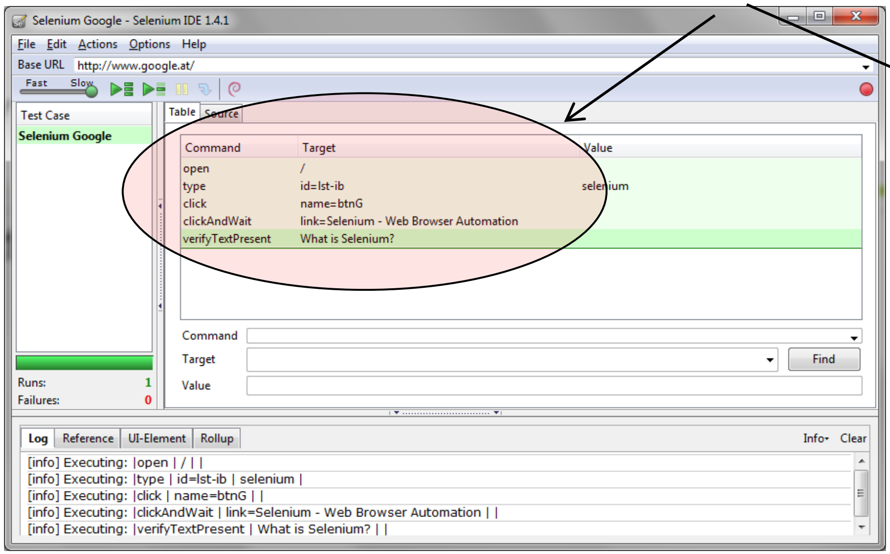
# *Key Word Driven Testing*



- Testfälle zerlegt in Keywords (=Abstrahierung)
- Keywords = Wiederverwendbare Schritte
- Steuerung von Tests mittels Keywords
- Ein Skript je Keyword
- Synoyme
  - „Action word Testing“
  - „Table driven Testing“

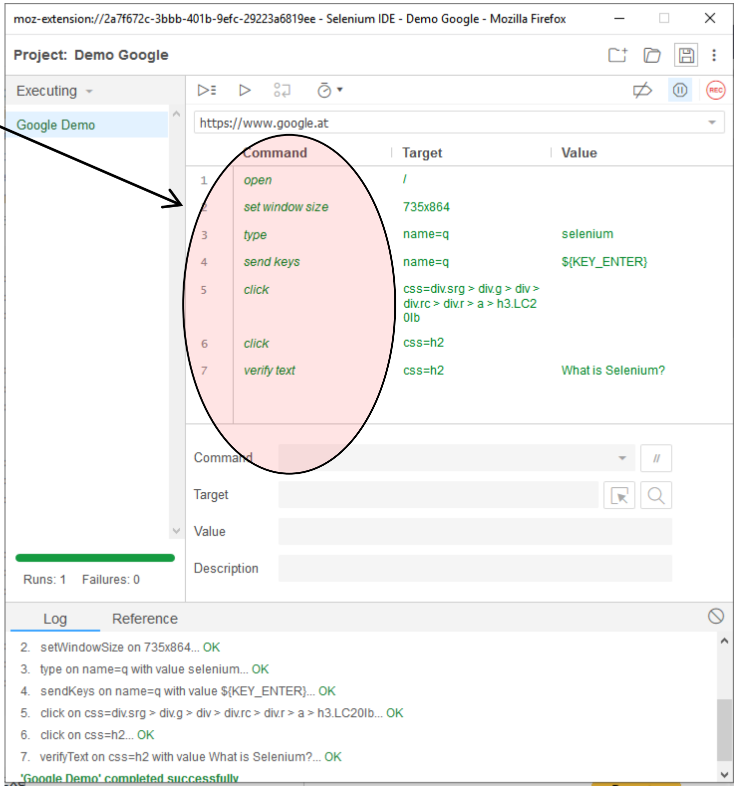
# Key Word Driven Testing -Bsp

Keywords


Selenium Google - Selenium IDE 1.4.1
File Edit Actions Options Help
Base URL http://www.google.at/
Fast Slow
Test Case
Selenium Google
Table Source
Command
Target
Value
Find
Runs: 1
Failures: 0
Log Reference UI-Element Rollup
Info▾ Clear

| Command | Target | Value |
|---|---|---|
| open | / | |
| type | id=lst-ib | selenium |
| click | name=btnG | |
| clickAndWait | link=Selenium - Web Browser Automation | |
| verifyTextPresent | What is Selenium? | |

```
[info] Executing: |open | / | |
[info] Executing: |type | id=lst-ib | selenium |
[info] Executing: |click | name=btnG | |
[info] Executing: |clickAndWait | link=Selenium - Web Browser Automation | |
[info] Executing: |verifyTextPresent | What is Selenium? | |
```

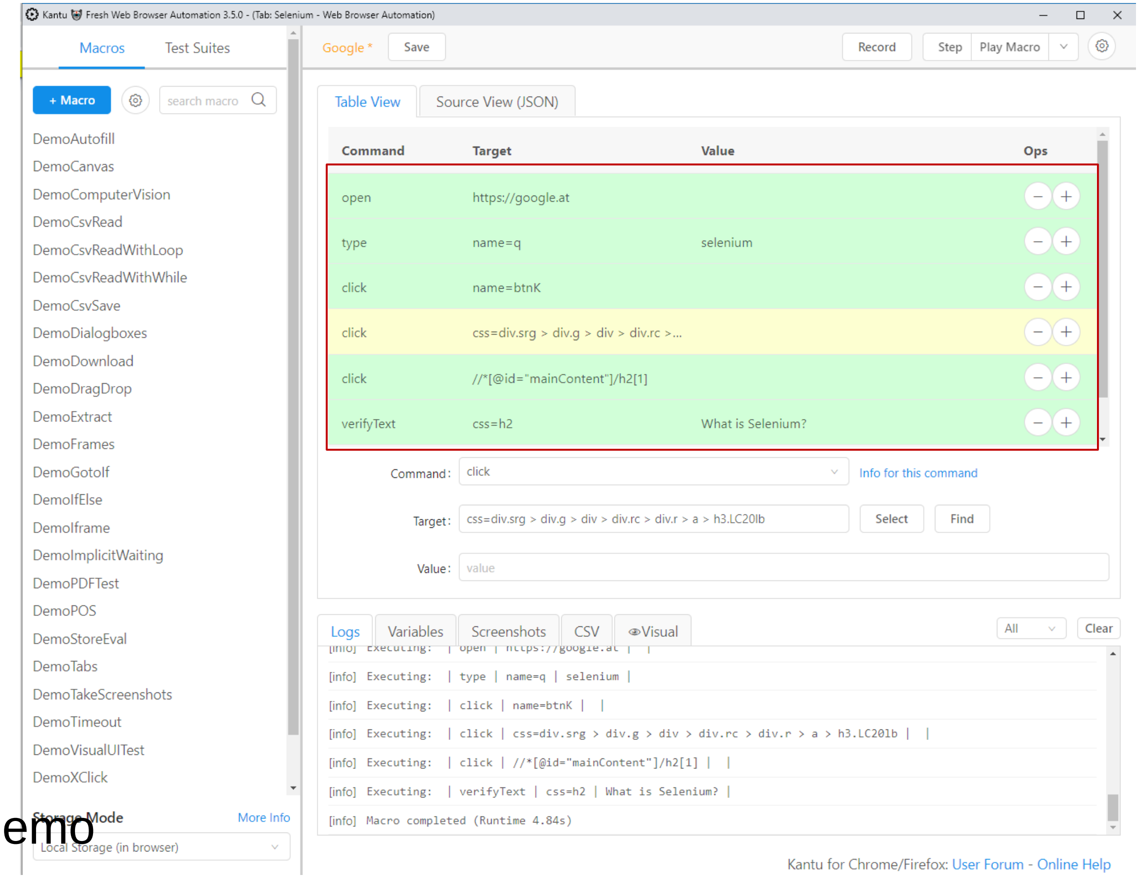


| | Command | Target | Value |
|---|---|---|---|
| 1 | open | / | |
| 2 | set window size | 735x864 | |
| 3 | type | name=q | selenium |
| 4 | send keys | name=q | ${KEY_ENTER} |
| 5 | click | css=div.srg > div.g > div > div.rc > div.r > a > h3.LC20lb | |
| 6 | click | css=h2 | |
| 7 | verify text | css=h2 | What is Selenium? |

```
2.  setWindowSize on 735x864... OK
3.  type on name=q with value selenium... OK
4.  sendKeys on name=q with value ${KEY_ENTER}... OK
5.  click on css=div.srg > div.g > div > div.rc > div.r > a > h3.LC20lb... OK
6.  click on css=h2... OK
7.  verifyText on css=h2 with value What is Selenium?... OK
'Google Demo' completed successfully
```

Achtung: Klassische Selenium IDE geht nicht mehr in Firefox ab Version 55
Neue IDE (rechts) als Chrome / Firefox Extension – noch nicht final
Selenium IDE installieren: https://addons.mozilla.org/de/firefox/addon/selenium-ide/

# *Kantu – Selenium IDE für Chrome*


Kantu Fresh Web Browser Automation 3.5.0 - (Tab: Selenium - Web Browser Automation)
Macros
Test Suites
+ Macro
search macro
Google *
Save
Record
Step
Play Macro
Table View
Source View (JSON)
Logs
Variables
Screenshots
CSV
Visual
All
Clear
Storage Mode
More Info
Local Storage (in browser)
Kantu for Chrome/Firefox: User Forum - Online Help

DemoAutofill
DemoCanvas
DemoComputerVision
DemoCsvRead
DemoCsvReadWithLoop
DemoCsvReadWithWhile
DemoCsvSave
DemoDialogboxes
DemoDownload
DemoDragDrop
DemoExtract
DemoFrames
DemoGotoIf
DemoIfElse
DemoIframe
DemoImplicitWaiting
DemoPDFTest
DemoPOS
DemoStoreEval
DemoTabs
DemoTakeScreenshots
DemoTimeout
DemoVisualUITest
DemoXClick

| Command | Target | Value | Ops |
|---|---|---|---|
| open | https://google.at | | − + |
| type | name=q | selenium | − + |
| click | name=btnK | | − + |
| click | css=div.srg > div.g > div > div.rc >... | | − + |
| click | //*[@id="mainContent"]/h2[1] | | − + |
| verifyText | css=h2 | What is Selenium? | − + |

Command: click — Info for this command

Target: css=div.srg > div.g > div > div.rc > div.r > a > h3.LC20lb — Select — Find

Value: value

```
[info]  Executing:  | open | https://google.at |  |
[info]  Executing:  | type | name=q | selenium |
[info]  Executing:  | click | name=btnK |  |
[info]  Executing:  | click | css=div.srg > div.g > div > div.rc > div.r > a > h3.LC20lb |  |
[info]  Executing:  | click | //*[@id="mainContent"]/h2[1] |  |
[info]  Executing:  | verifyText | css=h2 | What is Selenium? |
[info]  Macro completed (Runtime 4.84s)
```

- Demo

Achtung: Nur bedingt einsetzbar – viel Handarbeit

# *Was ist Selenium?*

- Toolset zum Automatisierten Testen von Webapplikationen (Url: http://seleniumhq.org)
  - Basiert auf Keyword Paradigma
  - Schwerpunkte
    - Funktionales Testen (Testquadrant Q2 bzw. Layer 3)
    - GUI Layer
    - Nicht für: Performance!
  - Open Source Tool von Thoughtworks
  - Starke Verbreitung
  - Historie
    - Selenium 1 ab 2004 entwickelt
    - Selenium 2 ab ca. 2006, Released Juli 2011
    - Selenium 3 Released 2016

# Selenium 1.0

Obosolet

**Selenium-IDE**

- Aufzeichnen und Abspielen von JS basierten Testcases in Firefox Addon
- Testcases exportierbar in verschiedene Sprachen (HTML, Java, C#,PHP…)

**Selenium-RC (Remote Control)**

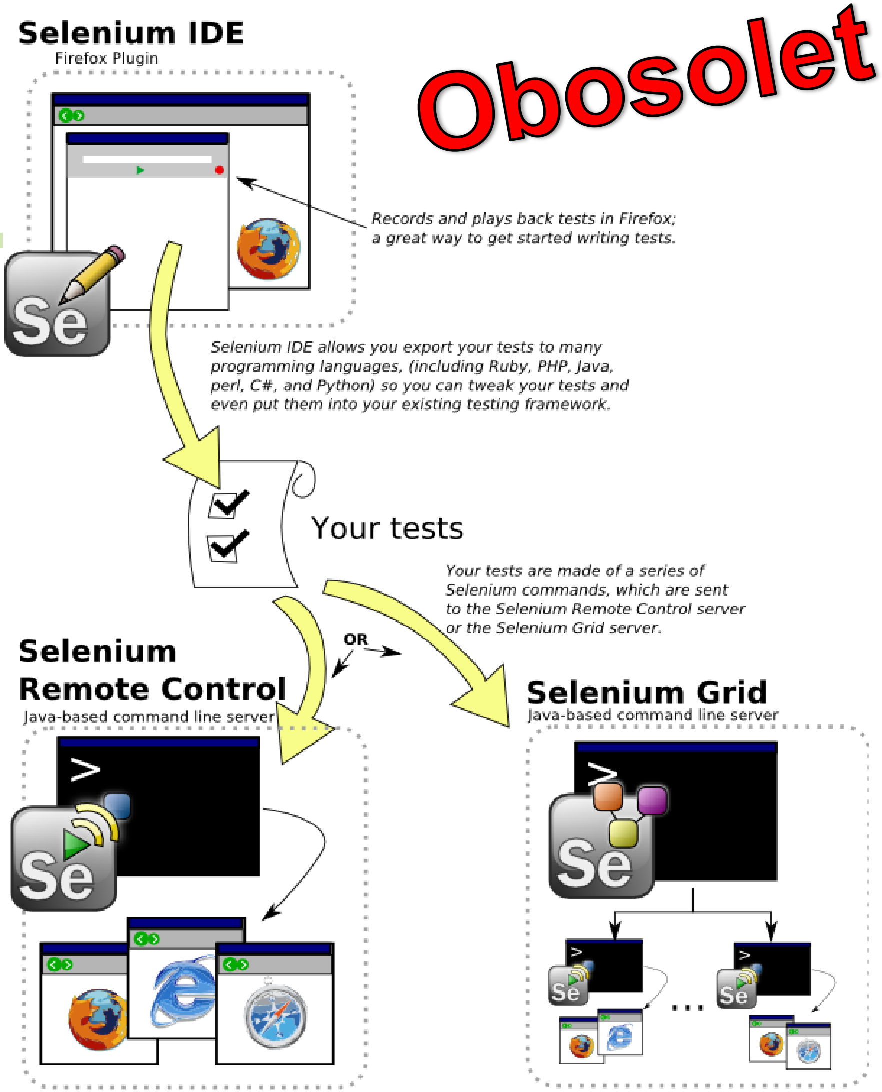
- API zum Programmatischen Anpassen der Testcases und Controllogik

**Selenium-Grid**

- Scale Out Erweiterung für Multi-Umgebungstests

**Nachteile von Sel. 1.0:**

- **JS basiert (Engine abhängig)**
- **Braucht Selenium RC**



*Records and plays back tests in Firefox; a great way to get started writing tests.*

*Selenium IDE allows you export your tests to many programming languages, (including Ruby, PHP, Java, perl, C#, and Python) so you can tweak your tests and even put them into your existing testing framework.*

*Your tests are made of a series of Selenium commands, which are sent to the Selenium Remote Control server or the Selenium Grid server.*

*Selenium Remote Control starts up browsers (one at a time) and then runs commands you pass along from your tests*

*Selenium Grid coordinates multiple Selenium Remote Control servers so you can run tests on lots of platforms at the same time, which saves lots of time and allows wider testing.*

# *Selenium 2 (= Webdriver)*

- Integration der Webdriver API
- Was ist Webdriver?
  - Neuimplementierung von Selenium
  - **Native API** statt Java Script (Webdriver steuert Browser nativ an)
  - Tests in Sprache der Software (Java, C#, Ruby, Python)
  - Neue, intuitivere OO API, “vereinfacht” das Testen
  - Unterstützte Browser (=WebDrivers)
    - Google Chrome 12.0.712.0+
    - Internet Explorer 6, 7, 8, 9,10,… - 32 and 64-bit where applicable
    - Firefox 3.0, 3.5, 3.6, 4.0, 5.0, 6, 7,…
    - Opera 11.5+
    - HtmlUnit 2.9
    - Android – 2.3+ for phones and tablets (devices & emulators)
    - iOS 3+ for phones (devices & emulators) and 3.2+ for tablets (devices & emulators
  - DIE Zukunft von Selenium
  - Abwärtskompatibel (Selenium Server -> nicht empfohlen)

# *Selenium 2*

- Nur noch 1 Tool

- Selenim 2.0 (Webdriver)
  - selenium-java-2.x.x.x.jar

# *Selenium 3 – Evolution*

- WebDriver users will just find bug fixes and a drop-in replacement for 2.x.
- Selenium Grid users will also find bug fixes and a simple update.
- The WebDriver APIs are now the only APIs actively supported by the Selenium project.
- The Selenium RC APIs have been moved to a “legacy” package.
- The original code powering Selenium RC has been replaced with something backed by WebDriver, which is also contained in the "legacy" package.
- By a quirk of timing, Mozilla have made changes to Firefox that mean that from Firefox 48 you must use their geckodriver to use that browser, regardless of whether you're using Selenium 2 or 3.

# *Gecko Webdriver für Mozilla*

- The other notable change has been that there is now a W3C specification for browser automation

- Mozilla has been a front-runner in implementing the W3C WebDriver protocol. On the plus side, this has exposed problems with the spec as it has evolved, but it also means that Firefox support is hard to track as their engineering efforts have been forward looking, rather than on supporting the current wire protocol used by Selenium WebDriver. For now, the best advice we can offer is for you to try the latest release of geckodriver and Selenium together.

# *Gecko Webdriver*

- Best practice:

- Firefox Driver Manager inkludieren
- Source: https://github.com/bonigarcia/webdrivermanager
- Anwendung siehe Bsp
- Relativ jung - es gibt noch ein paar Issues
- Bsp: Firefox Crash beim driver.quit https://github.com/mozilla/geckodriver/issues/339

# *Webdriver*

- Einfache API
  - Blocking API mit wenigen Ausnahmen
    - Bsp. Click
  - Blocking über Waits (Implicit / Explicit)
    - Implicit (WebDriverSetting): Allg. Wait für ElementLoads
    - Explicit (WaitDriverWait): Spezielles Wait im Code
  - Selektieren von Elementen
    - ByName/ByID: Das meiste geht
    - ByXPath: Alles im DOM ansprechbar -> empfohlen
      - Syntax: http://www.w3schools.com/xsl/xpath_syntax.asp
      - xPath in Firefox Quantum: Inspektor -> Element selektieren -> Rechte Maustaste -> Kopieren -> Xpath (nur absolut)
      - Ältere FF Versionen: Firebug Addon
      - Chrome: Ranorex Addon: https://www.ranorex.com/selocity/browser-extension.html

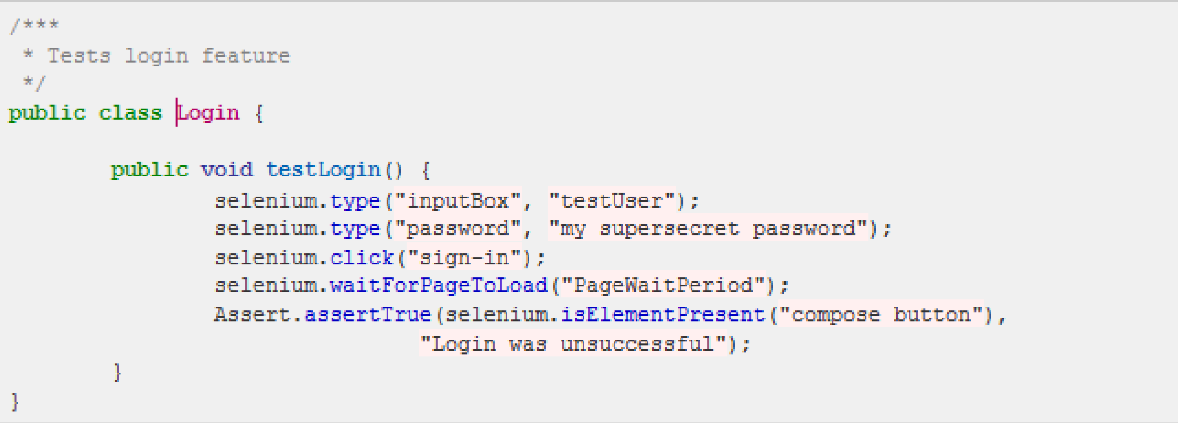
# *Beispiele*

- Selenium 2.0 (klassisch) (Bsp: Selemium-Classical)
  - Braucht Webdriver und Server
  - (Webdriver backed)
- Selenium 2.0 (Webdriver) (Bsp: Selemium-WebDriver)
  - Nur Webdriver
- Selenium 3.0 (PageObjectPattern)
  - Nur Webdriver
  - (Bsp: Selemium-WebdriverPageObject)

**Do it!**

# Page Object Pattern (1)

- Ein tradioneller Testcase (Keyword basiert)



```
/***
 * Tests login feature
 */
public class Login {

        public void testLogin() {
                selenium.type("inputBox", "testUser");
                selenium.type("password", "my supersecret password");
                selenium.click("sign-in");
                selenium.waitForPageToLoad("PageWaitPeriod");
                Assert.assertTrue(selenium.isElementPresent("compose button"),
                                "Login was unsuccessful");
        }
}
```

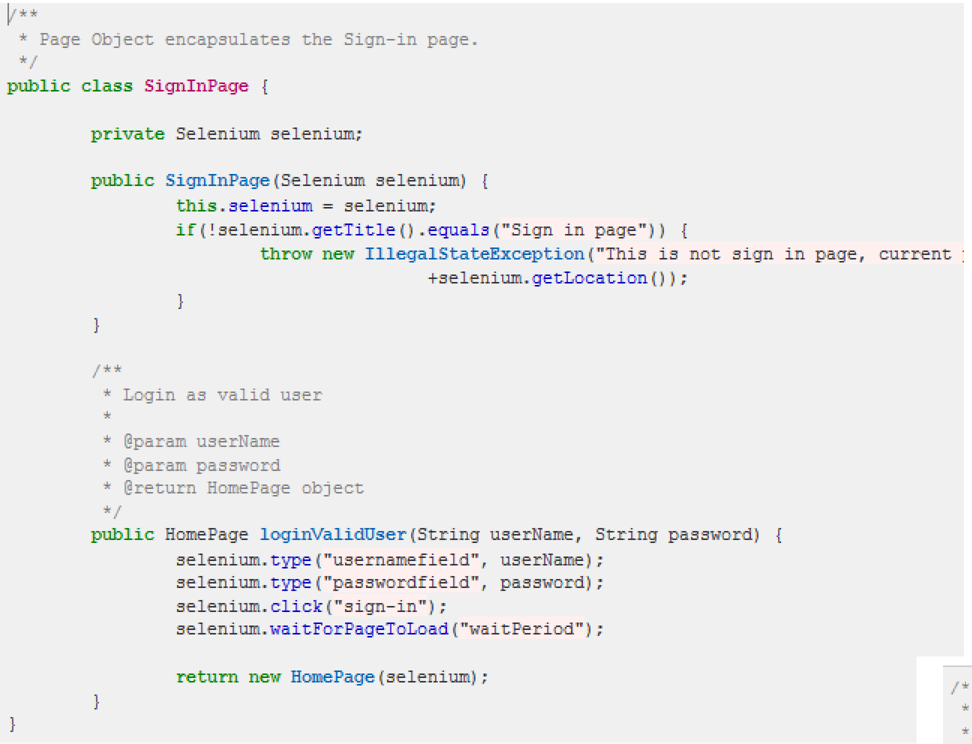
**Nachteile**

- Keine Trennung zwischen Testmethode und Lokatoren (UI Elemente)
- Lokatoren verstreut in Tests
- **Problem in Wartbarkeit**

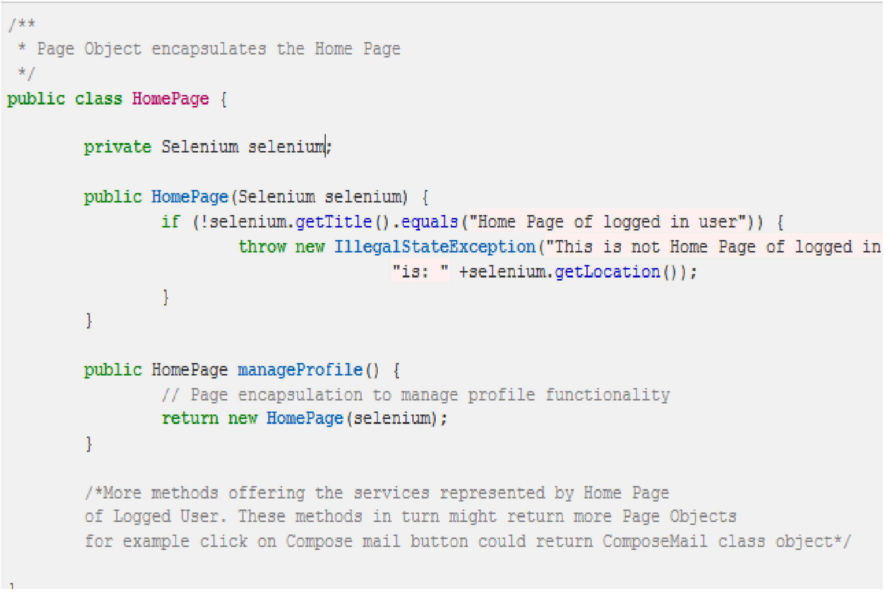
**Achtung : Selenium 1 Code**

# Page Object Pattern (2)

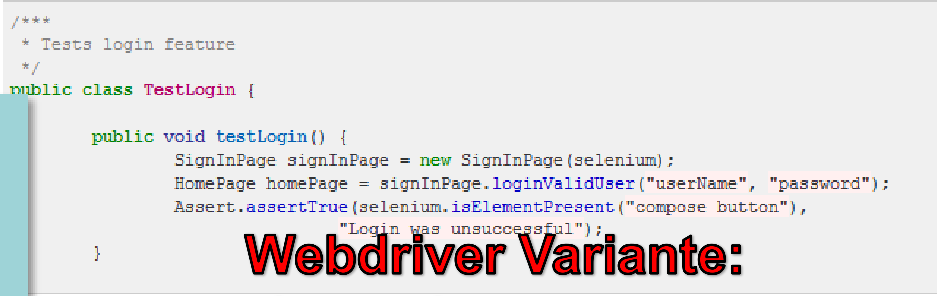
**Achtung :
Selenium 1 Code**



```java
/**
 * Page Object encapsulates the Sign-in page.
 */
public class SignInPage {

        private Selenium selenium;

        public SignInPage(Selenium selenium) {
                this.selenium = selenium;
                if(!selenium.getTitle().equals("Sign in page")) {
                        throw new IllegalStateException("This is not sign in page, current
                                        +selenium.getLocation());
                }
        }

        /**
         * Login as valid user
         *
         * @param userName
         * @param password
         * @return HomePage object
         */
        public HomePage loginValidUser(String userName, String password) {
                selenium.type("usernamefield", userName);
                selenium.type("passwordfield", password);
                selenium.click("sign-in");
                selenium.waitForPageToLoad("waitPeriod");

                return new HomePage(selenium);
        }
}
```



```java
/**
 * Page Object encapsulates the Home Page
 */
public class HomePage {

        private Selenium selenium;

        public HomePage(Selenium selenium) {
                if (!selenium.getTitle().equals("Home Page of logged in user")) {
                        throw new IllegalStateException("This is not Home Page of logged in
                                        "is: " +selenium.getLocation());
                }
        }

        public HomePage manageProfile() {
                // Page encapsulation to manage profile functionality
                return new HomePage(selenium);
        }

        /*More methods offering the services represented by Home Page
        of Logged User. These methods in turn might return more Page Objects
        for example click on Compose mail button could return ComposeMail class object*/
```



```java
/***
 * Tests login feature
 */
public class TestLogin {

        public void testLogin() {
                SignInPage signInPage = new SignInPage(selenium);
                HomePage homePage = signInPage.loginValidUser("userName", "password");
                Assert.assertTrue(selenium.isElementPresent("compose button"),
                                "Login was unsuccessful");
        }
```

**Webdriver Variante:
Siehe Codebsp**

Page Object Pattern

- Trennung zwischen Testmethode und Page Code
- Je Page eine Klasse mit Services / Operationen
- Return einer Operation ist ein PageObject
- **Einfachere Wartbarkeit (Kapselung in PageObject)**

# *Best Practices*

- Webdriver:
  - Doku lesen (Webdriver!)
    (nicht alles immer vollständig)
  - WIKI lesen, Beispiele googeln, API Javadoc hilft auch
  - Interessante Intro vom Webdriver-Erfinder:
    Simon Stewart (Google) Talk zu Webdriver, URL:
    http://www.youtube.com/watch?v=tGu1ud7hk5I
- Sonst
  - Mit einfachen Tests beginnen (nicht zu viel testen)
  - Nur Happy Path
  - PageObjectPattern anwenden!

# *Typische Stolperfallen*

- Browser startet nicht
  - Auf letzte Version von Selenium upgraden (Hohe Update Raten der Browser!)
  - Eigenes Profil definieren + Optionen setzen
- Basic Authentication
  - Eigenes Profil + Ausschalten von „Phishy-“Security dann: user:pwd@url
- Warten auf Pageload
  - Explizite Waits setzen
- JSF und eindeutige Ids (insbesondere bei JSF 1)
  - Wenn Problem: Manuelles setzen von Client Ids
  - Sonst: Eindeutiger Xpath probieren oder (übersetzter Text)
  - Blog: http://illegalargumentexception.blogspot.co.at/2009/10/jsf-working-with-component-identifiers.html
- Chromeprobleme: http://www.automationtestinghub.com/selenium-chromedriver/

# *Typische CI Build Chain*

1. Build
2. Isolierte Unittests
3. Deploy (falls 2 ok)
4. Integrationstests  (manchmal auch vor Deploy)
5. Selenium Tests (falls 4 ok)

# *Selenium vs. kommerzielle Tools*

- Bsp: Ranorex (Grazer Unternehmen)
  - Sehr erfolgreich!

- VT:
  - Mehr GUI / Inhalte oft ähnlich, „Xpath Mapper“
  - Schwerpunkt mehr für explizite Tester, weniger nah am „Entwickler“
- NT:
  - Extra Tool, nicht so integriert in die CI Chain

# *Testen von JS-basierten Clients*

- Strategien analog gezeigten Strategien

- Backend Services wie üblich (Unit Tests)
- Javascript Funktionalität
  - Verschiedene Unit Test-Tools: Jasmine / Mocha /Chai mit BDD Ansätzen
  - Variante Snapshot-Testing: Facebook & React: Jest
    - Snapshot Testing: Serialisierung von Webkomponenten & Diff
- Funktional Tests
  - Selenium

# *Practical Software Engineering*

Unit Testing Refined

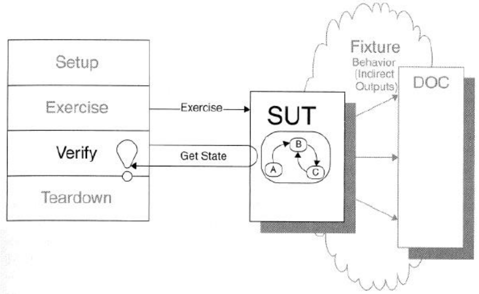
FH JOANNEUM
Internettechnik
Software Design
WS 2018

# *Agenda*

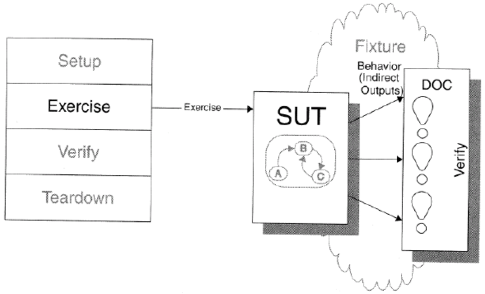
- Unit Testing Refined
  - Mocks / Stubs
  - Wie testet man sinnvoll auf Unitebene
  - Testbares Design
  - Wartbare Tests
  - Refactoring

# *Motivation Mocks*

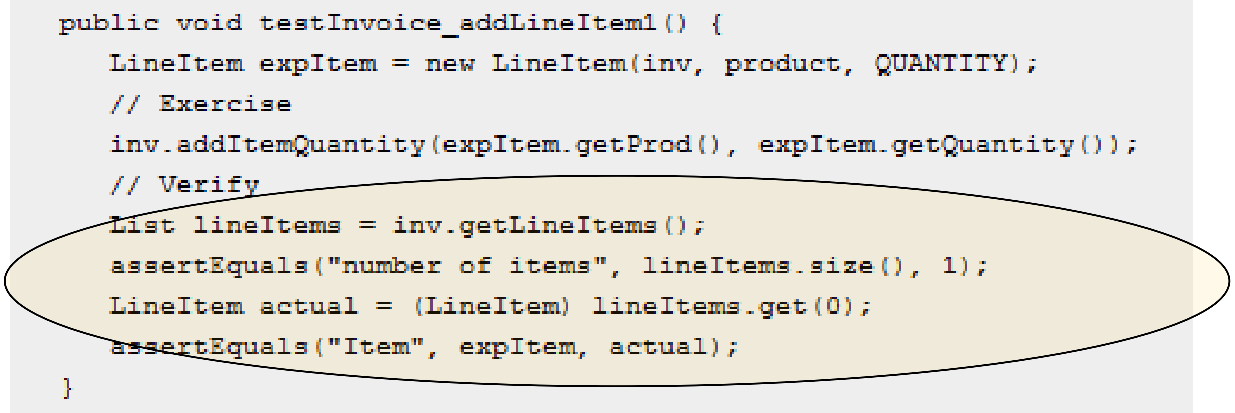
- 2 Arten von Verifikationen nach xUnit Patterns
  - State Verification
    - Zustand nach SUT Test wird geprüft



  - Behaviour Verification
    - Verhalten nach SUT Test wird geprüft

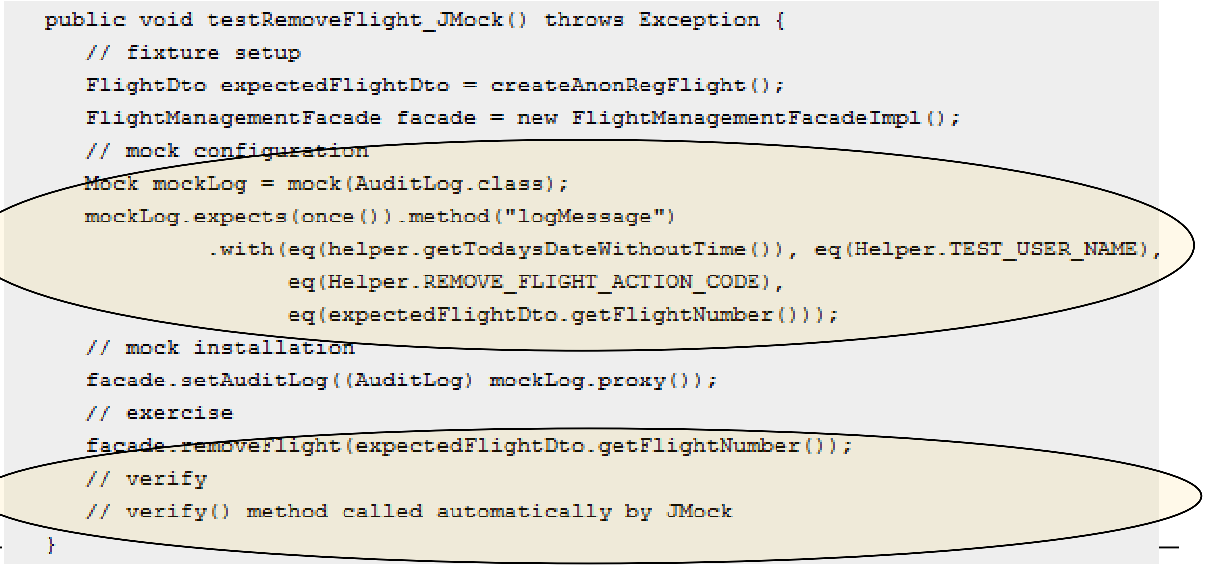


# *State Verification vs. Behaviour Verification*



```
public void testInvoice_addLineItem1() {
   LineItem expItem = new LineItem(inv, product, QUANTITY);
   // Exercise
   inv.addItemQuantity(expItem.getProd(), expItem.getQuantity());
   // Verify
   List lineItems = inv.getLineItems();
   assertEquals("number of items", lineItems.size(), 1);
   LineItem actual = (LineItem) lineItems.get(0);
   assertEquals("Item", expItem, actual);
}
```

State



```
public void testRemoveFlight_JMock() throws Exception {
   // fixture setup
   FlightDto expectedFlightDto = createAnonRegFlight();
   FlightManagementFacade facade = new FlightManagementFacadeImpl();
   // mock configuration
   Mock mockLog = mock(AuditLog.class);
   mockLog.expects(once()).method("logMessage")
            .with(eq(helper.getTodaysDateWithoutTime()), eq(Helper.TEST_USER_NAME),
                  eq(Helper.REMOVE_FLIGHT_ACTION_CODE),
                  eq(expectedFlightDto.getFlightNumber()));
   // mock installation
   facade.setAuditLog((AuditLog) mockLog.proxy());
   // exercise
   facade.removeFlight(expectedFlightDto.getFlightNumber());
   // verify
   // verify() method called automatically by JMock
}
```

Behaviour

# *Stubs vs. Mocks*

- Mock:
  - Ersetzt ein echtes Objekt in einem Test durch Fake
  - Definiert, ob Unit Test ok ist oder fehlerhaft
  - Anders ausgedrückt: **Mittels Mock wird Verfizierung durchgeführt**
- Stub
  - Ersetzt ein echtes Objekt in einem Test durch Fake
  - Test schlägt nie fehlt aufgrund eines Stubs
  - Anders ausgedrückt: **Mittels Stub wird keine Verfizierung durchgeführt**

# Grafische Darstellung - Stub

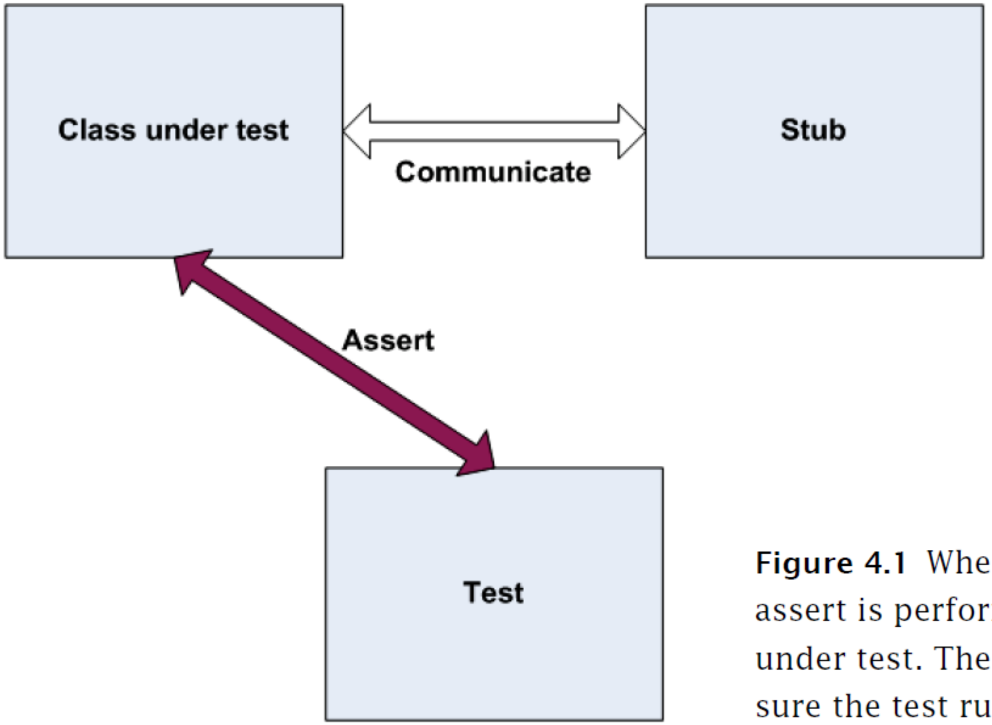


**Figure 4.1** When using a stub, the assert is performed on the class under test. The stub aids in making sure the test runs smoothly.

# *Grafische Darstellung - Mock*

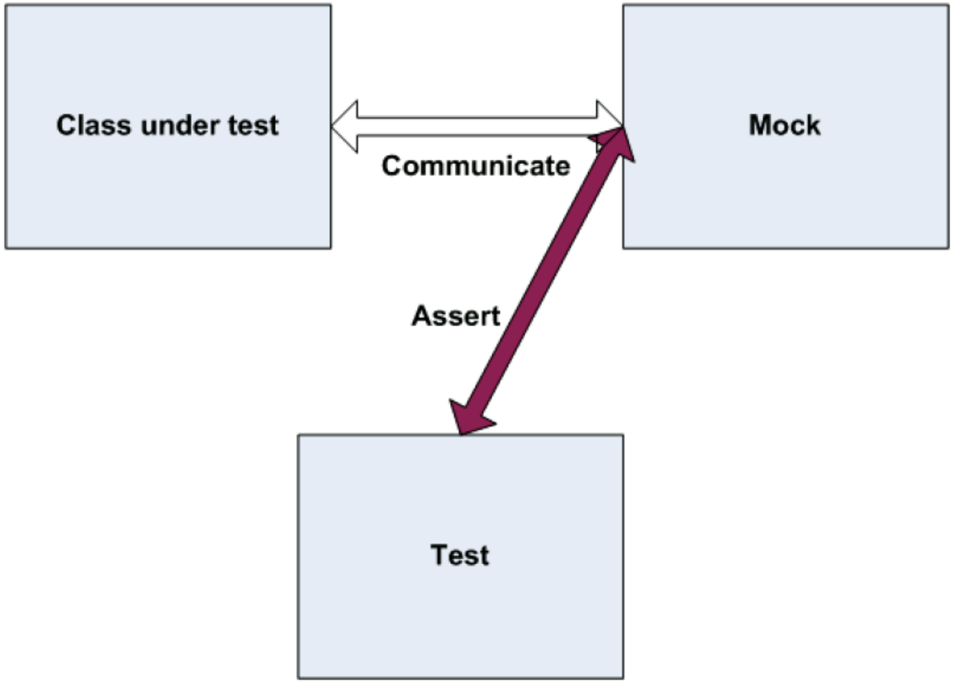


**Figure 4.2** The class under test communicates with the mock object, and all communication is recorded in the mock. The test uses the mock object to verify that the test passes.

# *Stubs vs. Mocks - Verwendung*

- Gute Unittests
  - Können viele Stubs haben
  - Sollten aber nur 1 Mock haben
- Hintergrund
  - Jeder Test sollte nur ein Testziel haben
  - Mehrere Mocks deuten auf mehrere Testziele hin
  - Mehrere Mocks: Tests splitten
  - Vergleiche OO Design:
    - 1 Methode sollte nur 1 Aufgabe aufgaben

# *Mock Frameworks*

- Stub wird im Prinzip erzeugt / aufgezeichnet wie ein Mock
- Wie unterscheidet sich dann ein Mock?
  - Aber: Verify wird nur auf Mock ausgeführt
  - Achtung: VerfiyAll verifiziert alle Mocks! -> verletzt Regel
- Achtung:
  - Es gibt Mockframeworks die zwischen Mocks und Stubs unterscheiden (Bsp. Mockito, EasyMock –Java, Rhinomocks - .NET)
  - Nicht alle Frameworks unterscheiden Stub / Mock
  - StrictMocks vs. Dynamic Mocks
    - StrictMocks: Nur aufgezeichnete Methoden können aufgerufen werden
    - Nonstrict / Dynamic Mocks: Auch nicht aufgezeichnete Methoden aufrufbar

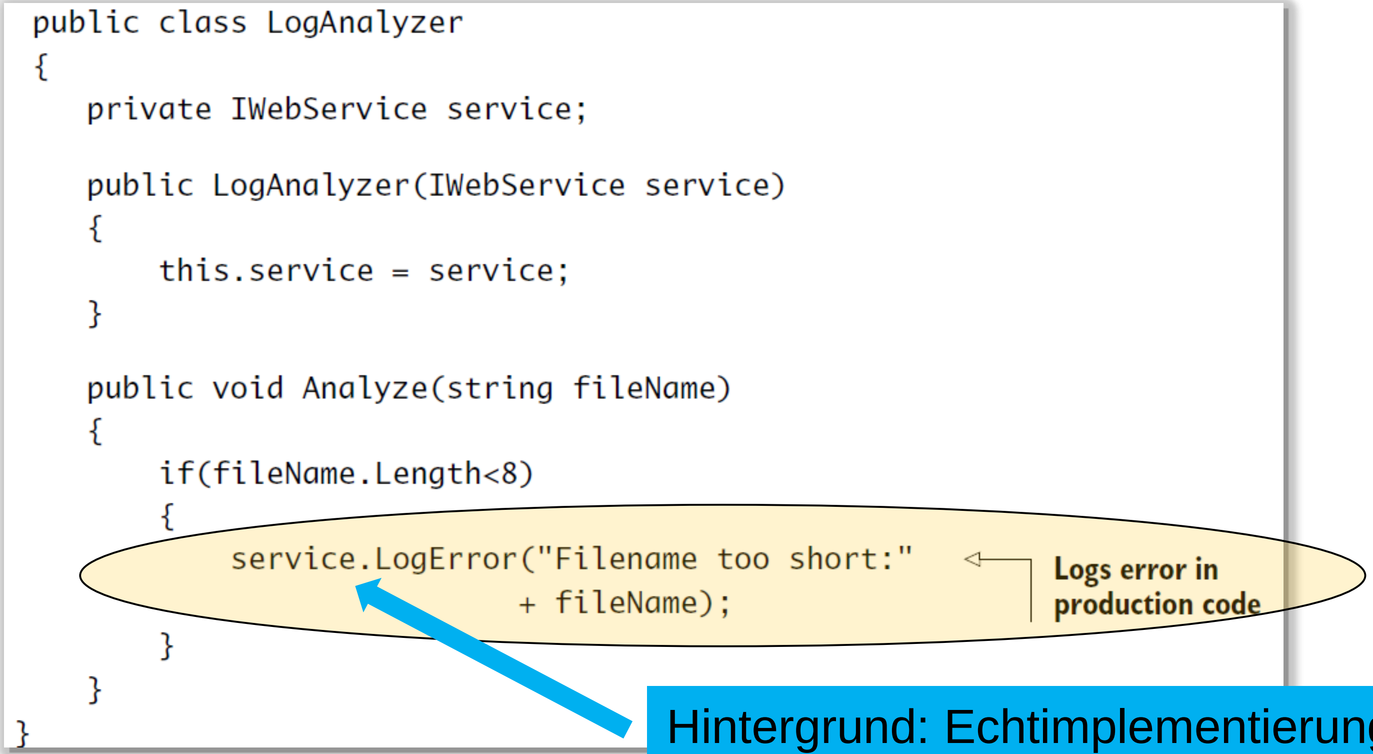
# *Demo Mocking Framework*

- Bsp: EasyMock

- Mittlerweile verbreiteter, da neuer: Mockito

# *Unittest Bsp – mit Mock*

- LogAnalyser.Analyse() soll getestet werden



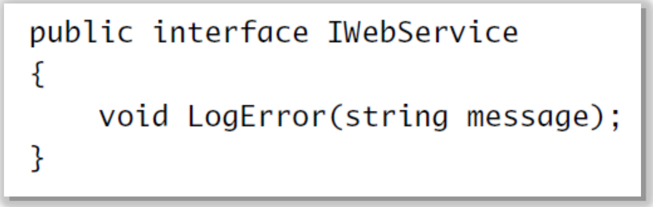
```
public class LogAnalyzer
{
   private IWebService service;

   public LogAnalyzer(IWebService service)
   {
       this.service = service;
   }

   public void Analyze(string fileName)
   {
       if(fileName.Length<8)
       {
           service.LogError("Filename too short:"
                           + fileName);
       }
   }
}
```

**Logs error in production code**

Hintergrund: Echtimplementierung loggt in das Filesystem -> soll isoliert werden

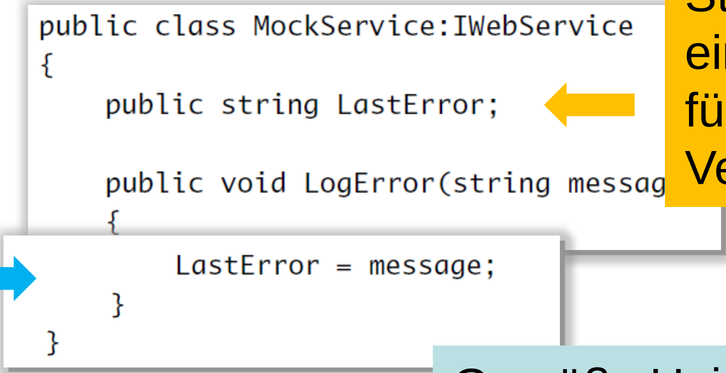
# *Ein manuelles Mock Beispiel*

## Ein Webservice



```
public interface IWebService
{
    void LogError(string message);
}
```

## Handgeschriebener Mock

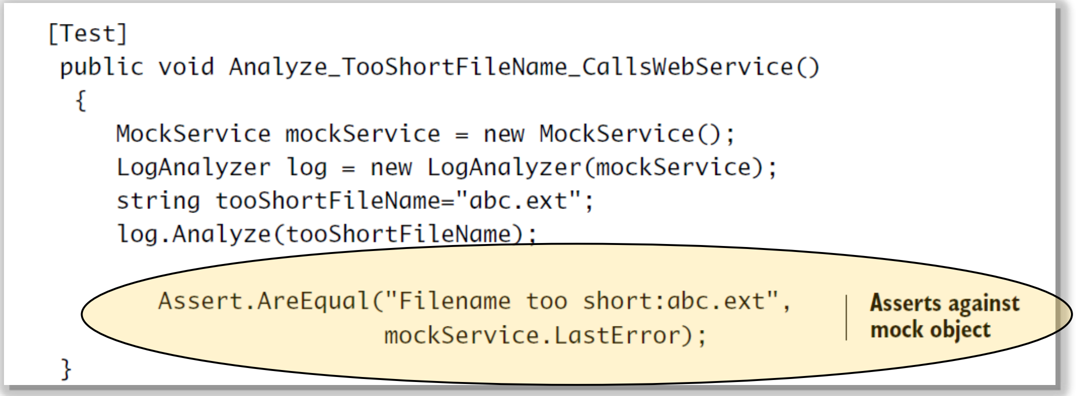


```
public class MockService:IWebService
{
    public string LastError;

    public void LogError(string messag
    {
        LastError = message;
    }
}
```

State eingeführt für State Verification

Stubtestlogik

## Der Test



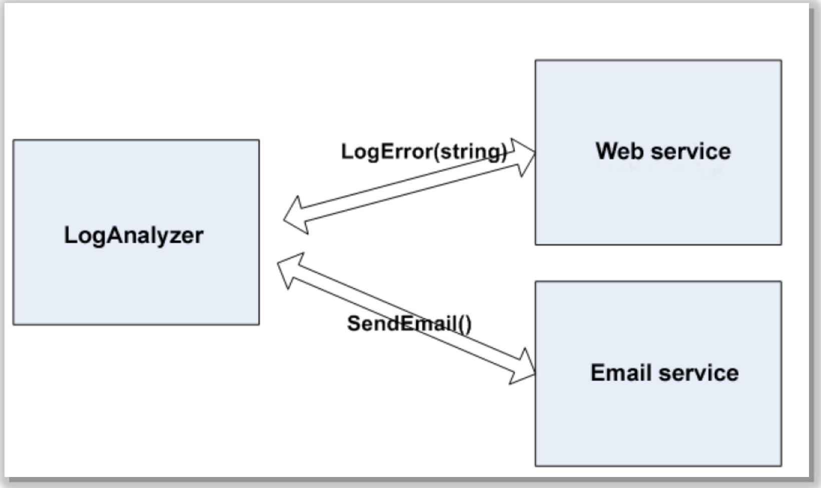
```
[Test]
 public void Analyze_TooShortFileName_CallsWebService()
 {
     MockService mockService = new MockService();
     LogAnalyzer log = new LogAnalyzer(mockService);
     string tooShortFileName="abc.ext";
     log.Analyze(tooShortFileName);

     Assert.AreEqual("Filename too short:abc.ext",
                     mockService.LastError);
 }
```

**Asserts against mock object**

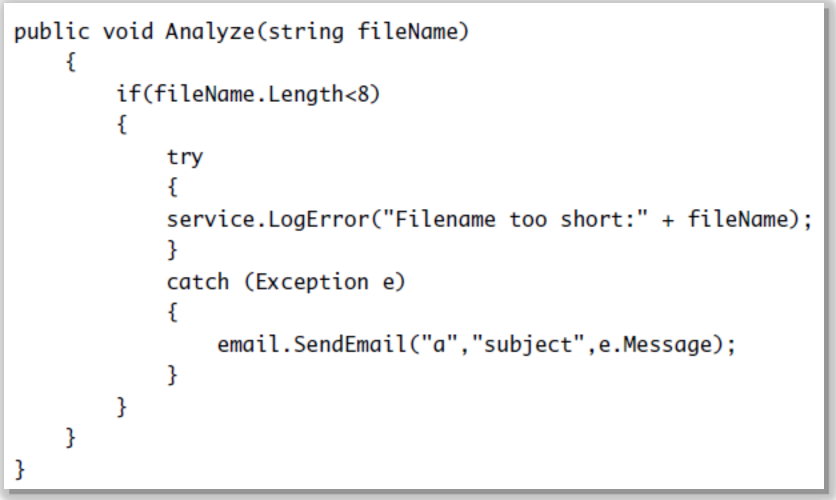
Gemäß xUnit Pattern ist das ein Testspy Warum: Zustand für Verify wird gespeichert

# *Mocks vs. Stubs - ein Bsp.*

- LogAnalyser schickt im Fehlerfall Mail an Administrator
- Ziel Unit Test: Wird Email korrekt versendet?



Die zu testende Methode

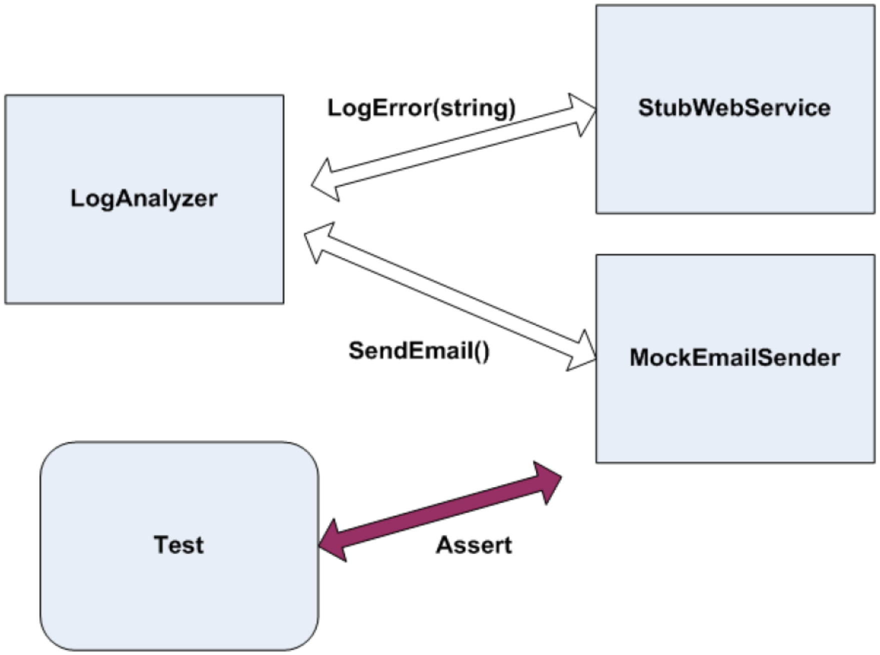


```
public void Analyze(string fileName)
    {
        if(fileName.Length<8)
        {
            try
            {
            service.LogError("Filename too short:" + fileName);
            }
            catch (Exception e)
            {
                email.SendEmail("a","subject",e.Message);
            }
        }
    }
}
```

  - Wie Testen?

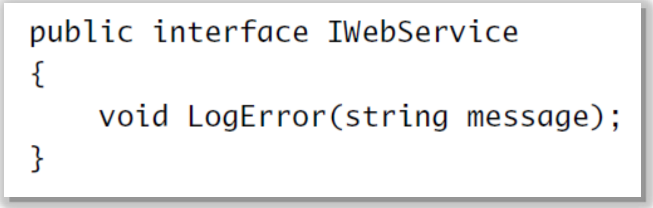
# *Testmethode: Mock und Stub*

- Webservice als Stub, EmailSender als Mock



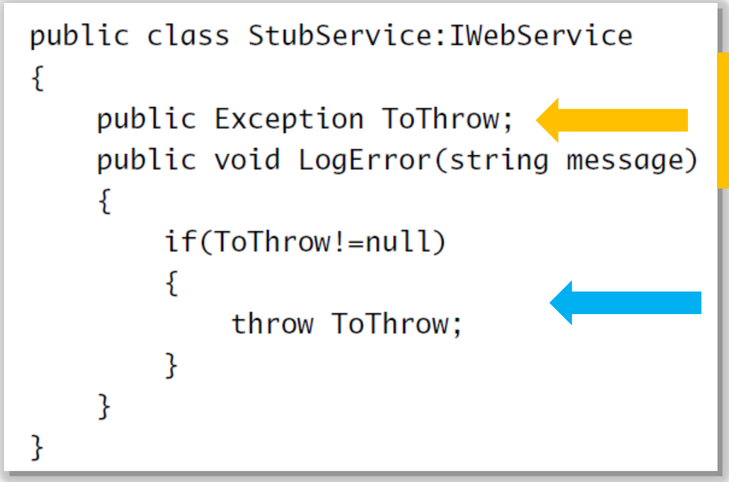
# *Webservice als Stub*

## Webservice



```
public interface IWebService
{
    void LogError(string message);
}
```

## Handgeschriebener Stub



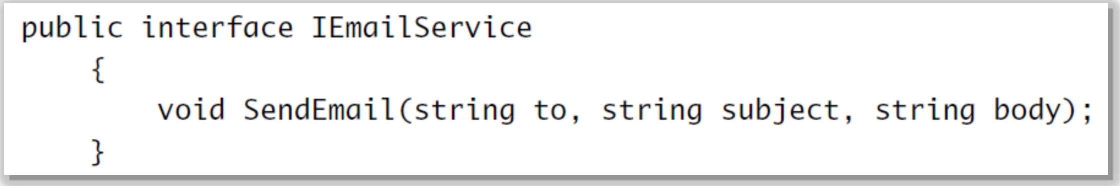
```
public class StubService:IWebService
{
    public Exception ToThrow;
    public void LogError(string message)
    {
        if(ToThrow!=null)
        {
            throw ToThrow;
        }
    }
}
```

Property Injection wg. Testbarkeit

Stubtestlogik

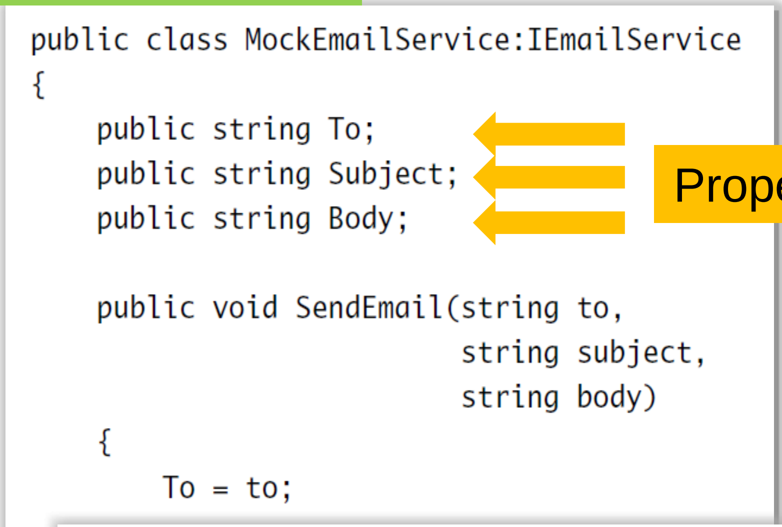
# Emailservice als Mock

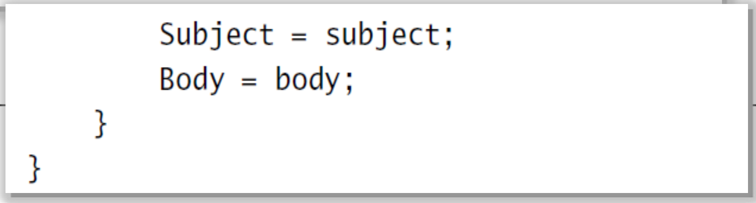
## Emailservice



```
public interface IEmailService
    {
        void SendEmail(string to, string subject, string body);
    }
```

## Handgeschriebener Mock

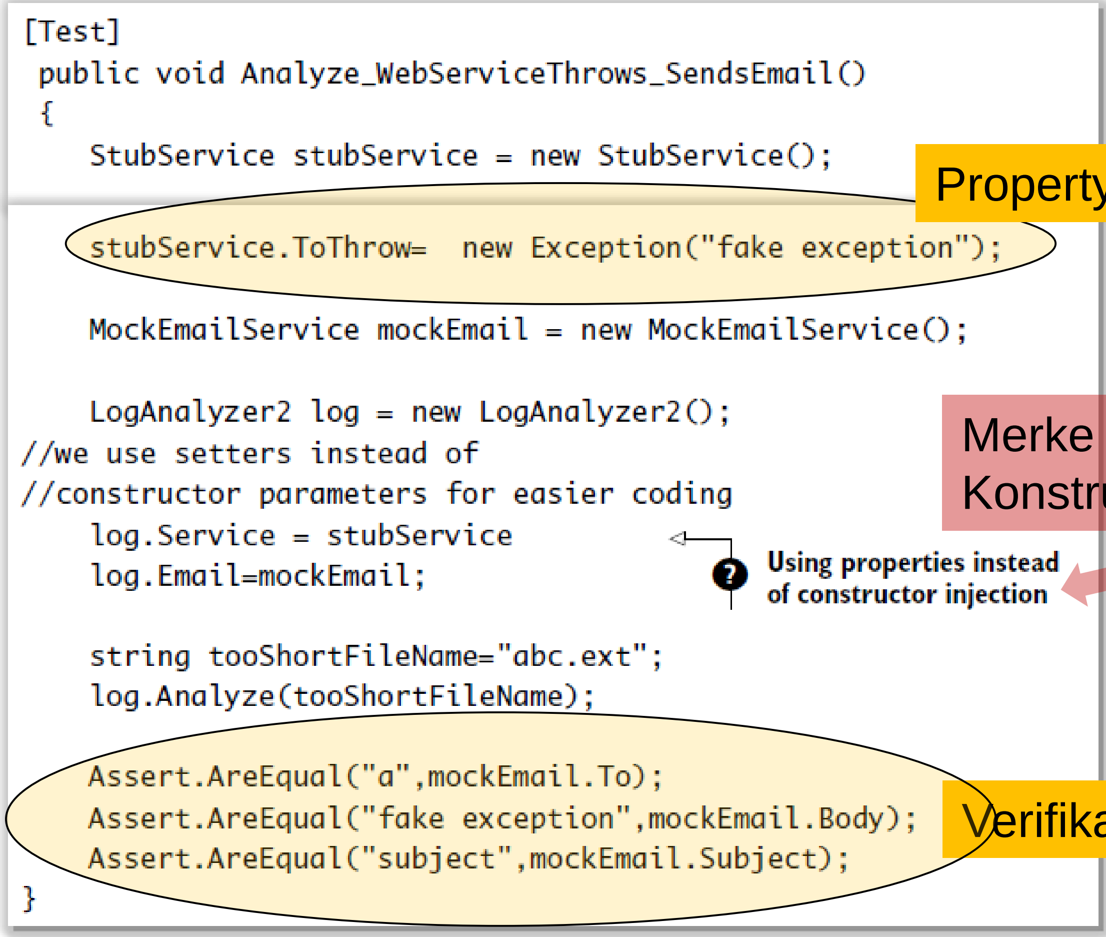




```
public class MockEmailService:IEmailService
{
    public string To;
    public string Subject;
    public string Body;

    public void SendEmail(string to,
                          string subject,
                          string body)
    {
        To = to;
        Subject = subject;
        Body = body;
    }
}
```

Property Injection

# Der Test dazu



```
[Test]
 public void Analyze_WebServiceThrows_SendsEmail()
 {
    StubService stubService = new StubService();

    stubService.ToThrow=  new Exception("fake exception");

    MockEmailService mockEmail = new MockEmailService();

    LogAnalyzer2 log = new LogAnalyzer2();
//we use setters instead of
//constructor parameters for easier coding
    log.Service = stubService
    log.Email=mockEmail;

    string tooShortFileName="abc.ext";
    log.Analyze(tooShortFileName);

    Assert.AreEqual("a",mockEmail.To);
    Assert.AreEqual("fake exception",mockEmail.Body);
    Assert.AreEqual("subject",mockEmail.Subject);
}
```

❶ Using properties instead of constructor injection

Property Injection für Verhalten

Merke Properties besser als Konstruktur für Testbarkeit

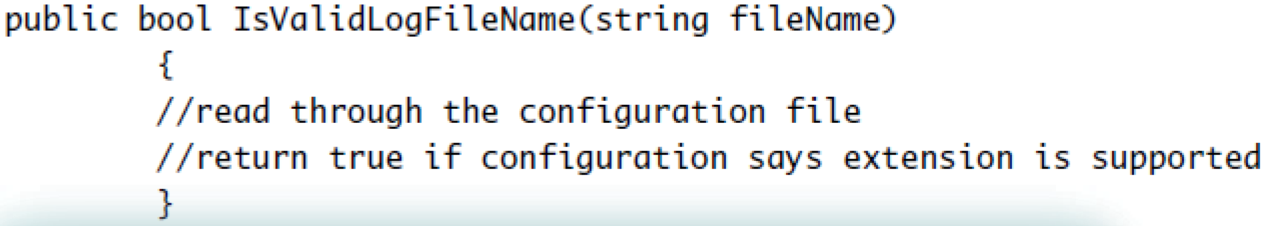
Verifikation nur am Mock

# *Testbares Design – Wie?*

- Indirektionen und Interfaces
- Injection / Factories / Dependency Injection
- Methoden extrahieren

# *Direkte Abhängigkeiten*

- Ein Beispiel - Loganalyser



```
public bool IsValidLogFileName(string fileName)
        {
        //read through the configuration file
        //return true if configuration says extension is supported
        }
```

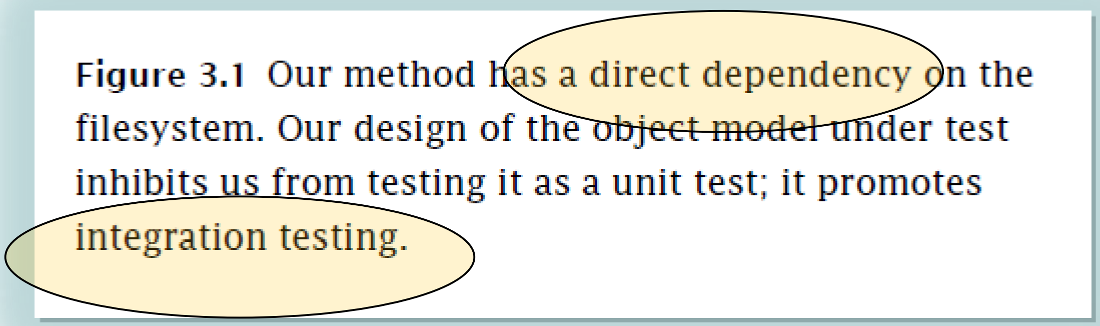
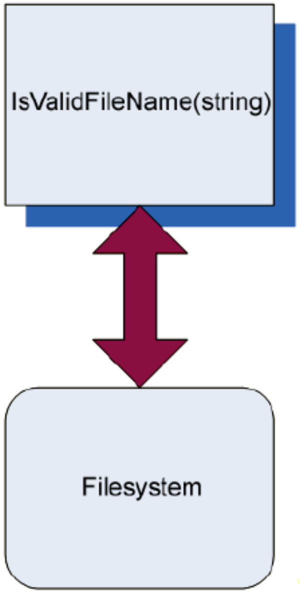


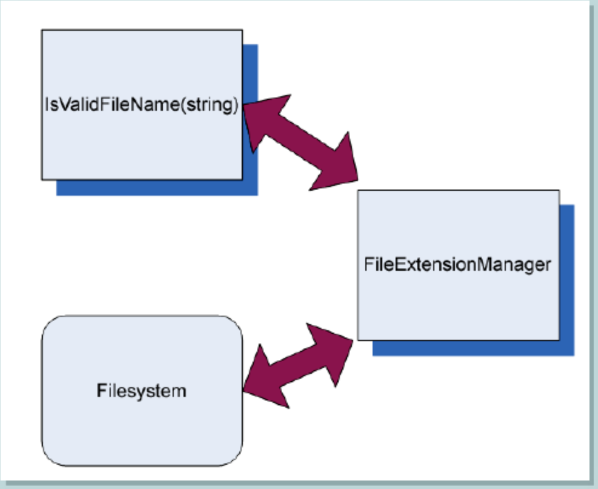
**Figure 3.1** Our method has a direct dependency on the filesystem. Our design of the object model under test inhibits us from testing it as a unit test; it promotes integration testing.

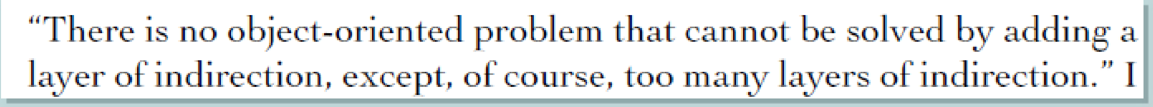


- Abhilfen: Indirektionen und Interfaces

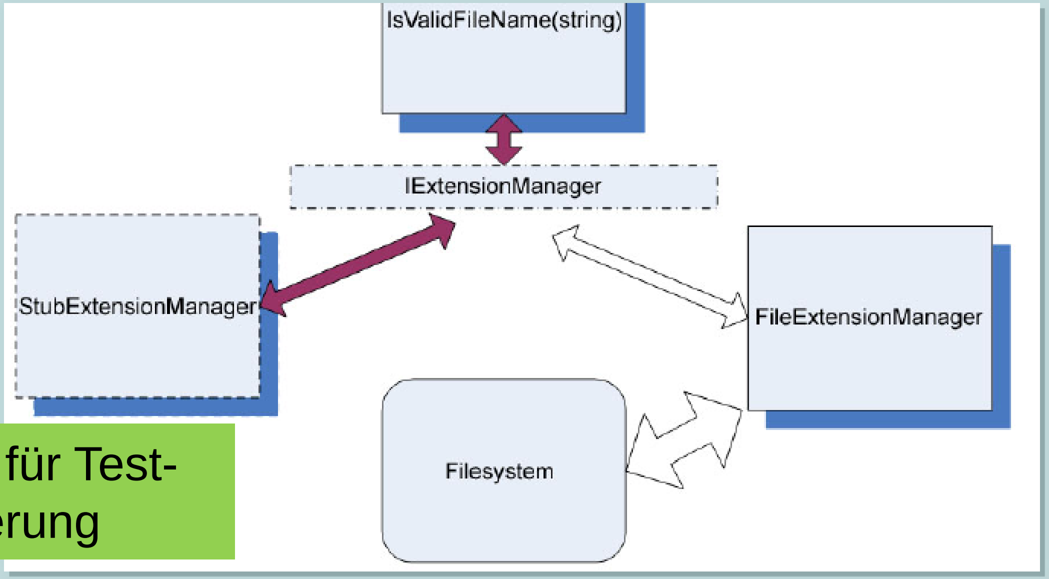
# *Direkte Abhängigkeiten - Abhilfen*

a) Indirektion (Layer)




“There is no object-oriented problem that cannot be solved by adding a layer of indirection, except, of course, too many layers of indirection.” I

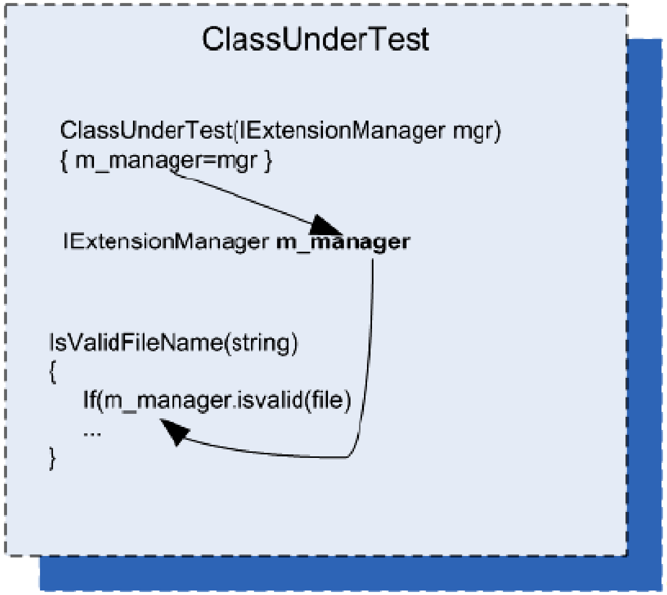
b) Interfaces separieren



Stub für Test-Isolierung

# *Injection Methodiken (1)*

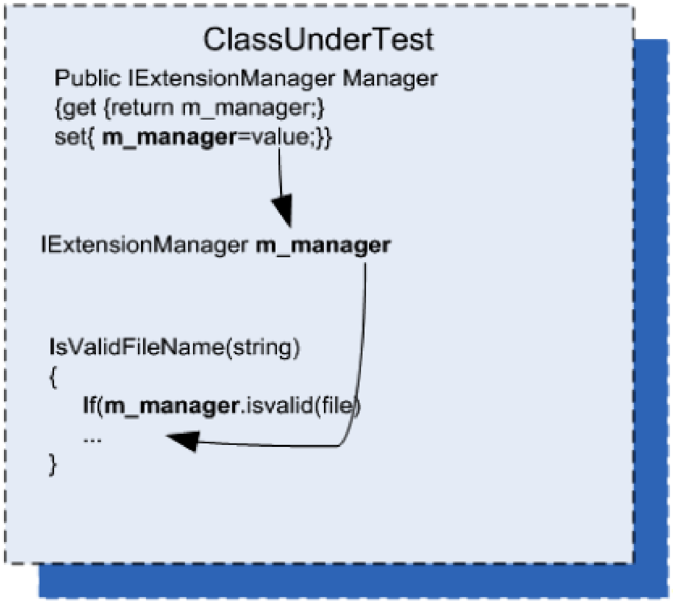
- Constructor Injection



```
ClassUnderTest

ClassUnderTest(IExtensionManager mgr)
{ m_manager=mgr }

IExtensionManager m_manager

IsValidFileName(string)
{
    If(m_manager.isvalid(file)
    ...
}
```

- Logischer im Sinne von OO (Kapselung)
- Problematisch bei vielen Depencies
- Sinnvoll nur mit DI Containern

- Property Injection

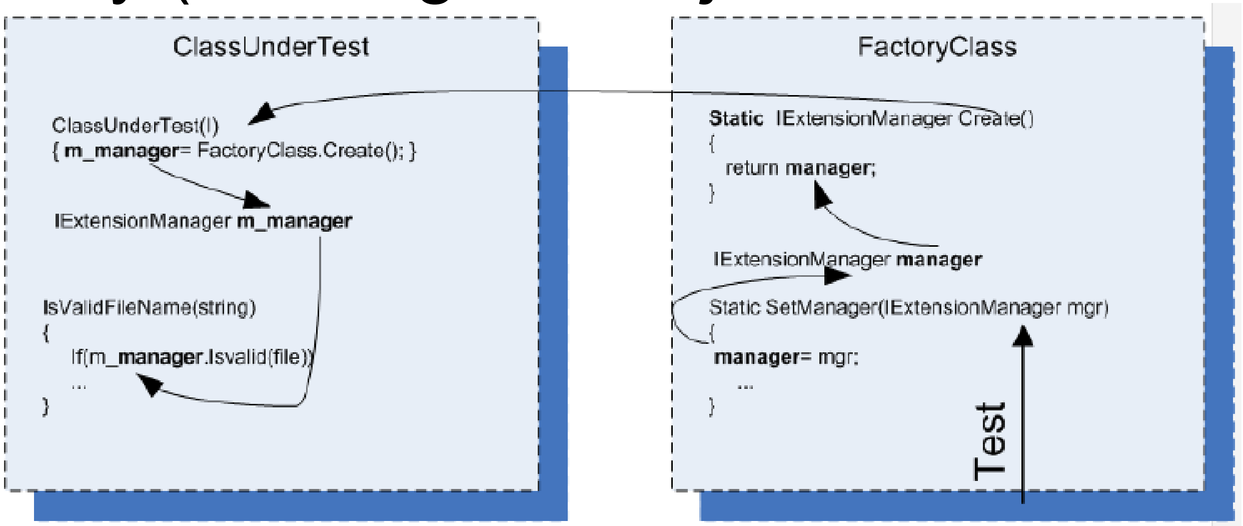


```
ClassUnderTest
Public IExtensionManager Manager
{get {return m_manager;}
set{ m_manager=value;}}

IExtensionManager m_manager

IsValidFileName(string)
{
    If(m_manager.isvalid(file)
    ...
}
```

- Unlogischer im Sinne von OO
- Suggeriert “optionale Objekte”
- Einfacher vom Design (do it!)

# *Injection Methodiken (2)*

- Factory (“Getting a stub just before a call”)



ClassUnderTest

```
ClassUnderTest(I)
{ m_manager= FactoryClass.Create(); }

IExtensionManager m_manager

IsValidFileName(string)
{
    If(m_manager.Isvalid(file))
    ...
}
```

FactoryClass

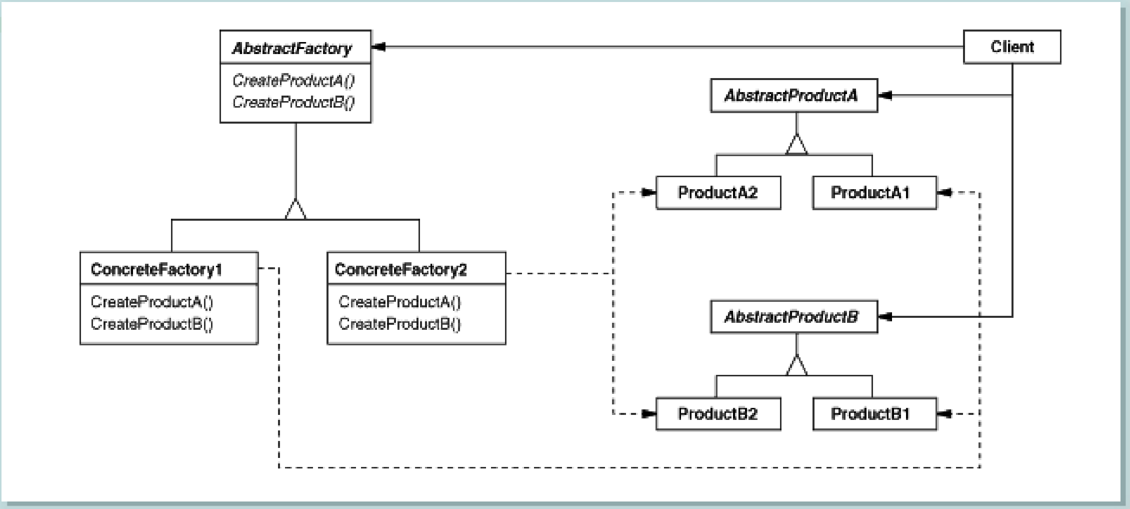
```
Static  IExtensionManager Create()
{
  return manager;
}

IExtensionManager manager

Static SetManager(IExtensionManager mgr)
{
manager= mgr;
  ...
}
```

- Klare Trennung (Separation of Concerns)
- Problem: Nachdenken darüber wie Factories Stubs/ Mocks liefern können
- Viele Varianten: Beispiele: Factory Method / Abstract Factory aus GOF Buch

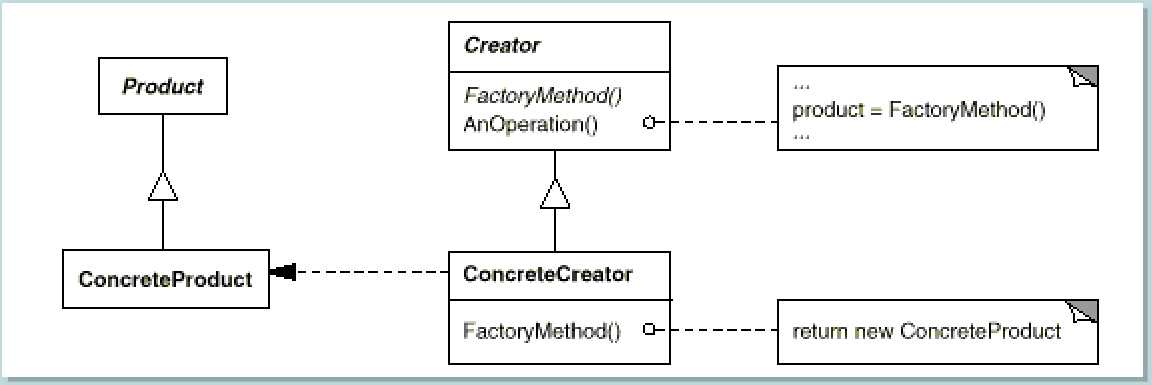
**Tipp für die Praxis: DI Framework nutzen! -> state of the art**
**Die meisten DI Frameworks können parametriseriert injecten**

# Zum Nachlesen - GOF Factories

- Abstract Factory



- Factory Method

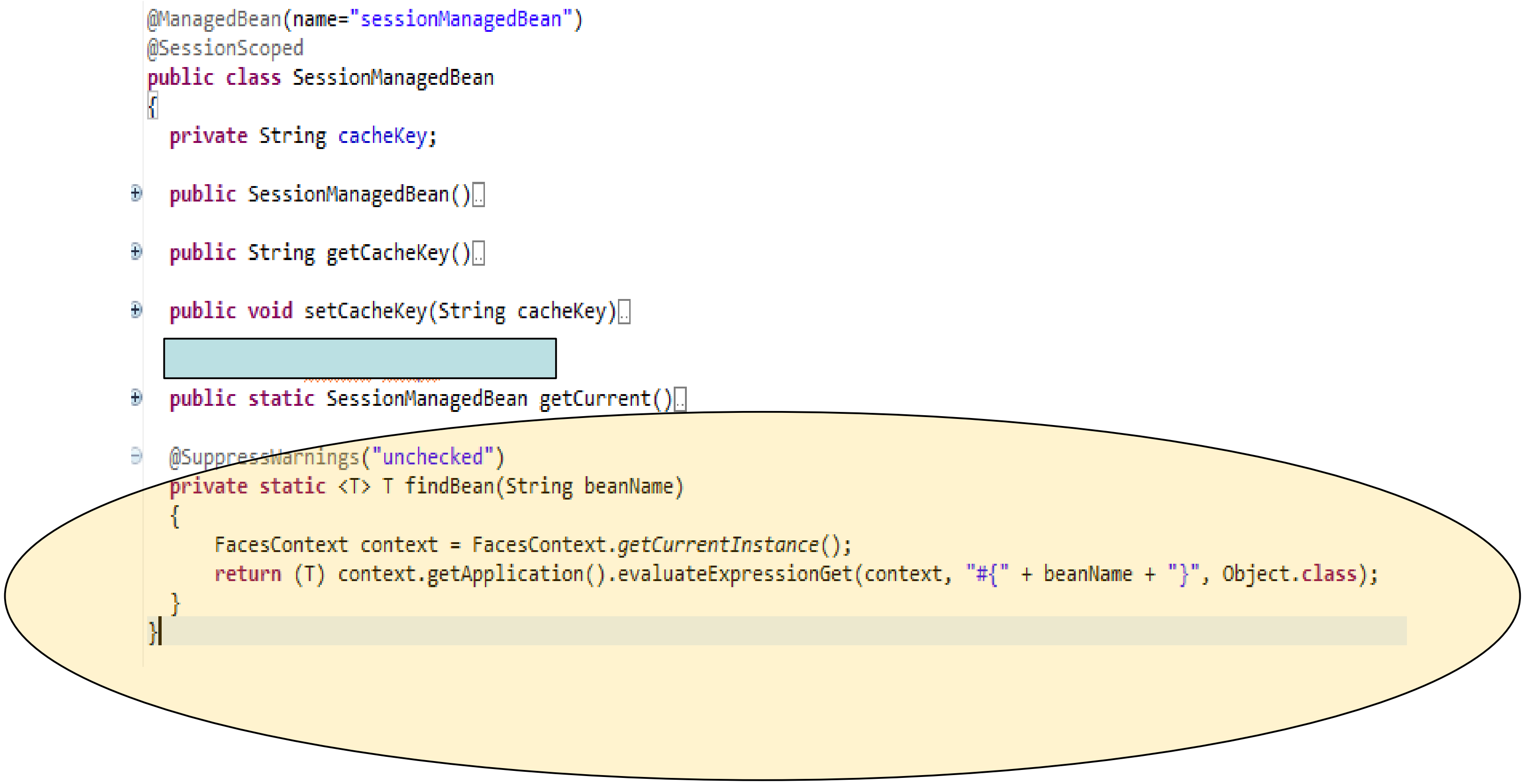


http://www.csie.ntu.edu.tw/~b92103/GoF/hires/contfso.htm

# *Protected /Private Methoden (1)*

- Testen von Private und Protected Methoden
  - Müssen per se nicht getestet werden (Rule of Thumb)
  - Warum: Häufige Änderungen brechen oft Tests
- Hintergrund
  - Sie sind meist bewusst private/protected by Design
  - Implementierung kann sich ändern
- Aber
  - Beinhalten aber oft viel Logik
  - Wie testbar machen?

# Bsp aus Übung aus 201x



```
@ManagedBean(name="sessionManagedBean")
@SessionScoped
public class SessionManagedBean
{
  private String cacheKey;

  public SessionManagedBean()..

  public String getCacheKey()..

  public void setCacheKey(String cacheKey)..

  [redacted]

  public static SessionManagedBean getCurrent()..

  @SuppressWarnings("unchecked")
  private static <T> T findBean(String beanName)
  {
      FacesContext context = FacesContext.getCurrentInstance();
      return (T) context.getApplication().evaluateExpressionGet(context, "#{" + beanName + "}", Object.class);
  }
}
```

# *Protected /Private Methoden (2)*

- Refactor Option: Methode Public machen
  - Ok für Testbarkeit
  - Aber: Ist das immer gewollt? Geht das immer?
  - Public Hintergrund: Jeder darf Methode benutzen
- Refactor Option: Methode in Neue Klassen extrahieren
  - Wenn viel Logik in Methode
  - Wenn Zustände der Klasse genutzt werden, die nur für Methode relevant
  - Methode wird so testbar
- Refactor Option: Public Static Methoden
  - Wenn keine Zustände
  - Methode wird testbar
- Refactor Option: Package Methode
  - Wenn Methode private aber Tests notwendig; Notnagel!

# *Gute Tests sind …*

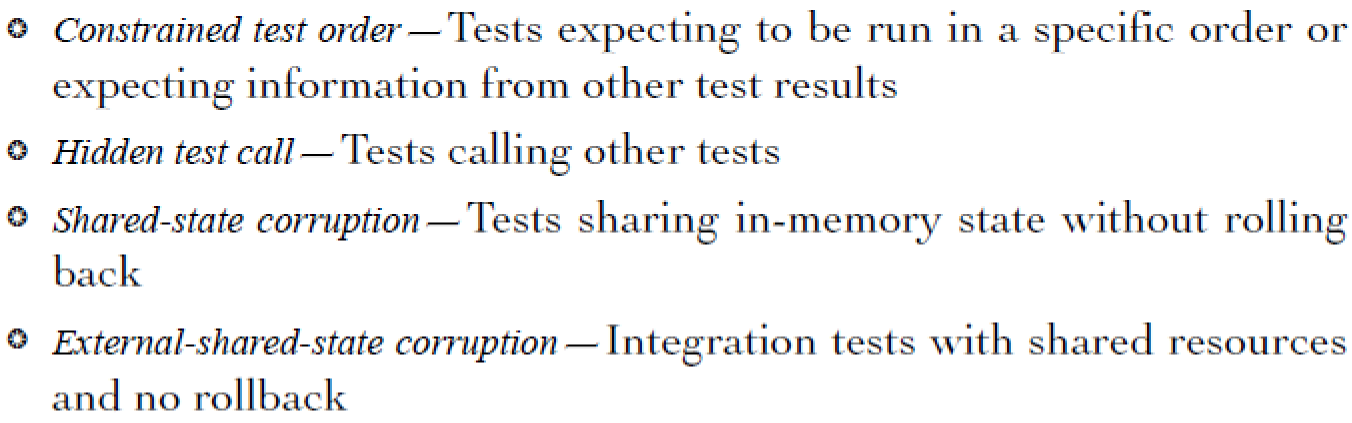
- Vertrauenswürdig
  - Tests sollten vertrauenswürdig sein (sonst Leichen)
  - Keine Bugs haben
  - Das Richtige Testen
- Wartbar
  - Nicht wartbare Tests werden Leichen
- Lesbar
  - Nicht lesbare Tests brechen die beiden oberen Forderungen
  - Dokumentation

# *Wartbare Tests (1)*

- Duplikate entfernen
  - DRY Prinzip (don’t repeat your self)
  - Auslagern in Helper / Setup
- Setup Methoden richtig verwenden
  - Nur für allgemeine Dinge (nicht für einzelne Tests)
- Keine mehrfache Asserts eines Objekts
  - Parametrisieren
  - Auslagern in einzelne Tests
- Mehrere Aspekte eines Objekts
  - Problem: Nicht klar, was schiefgeht

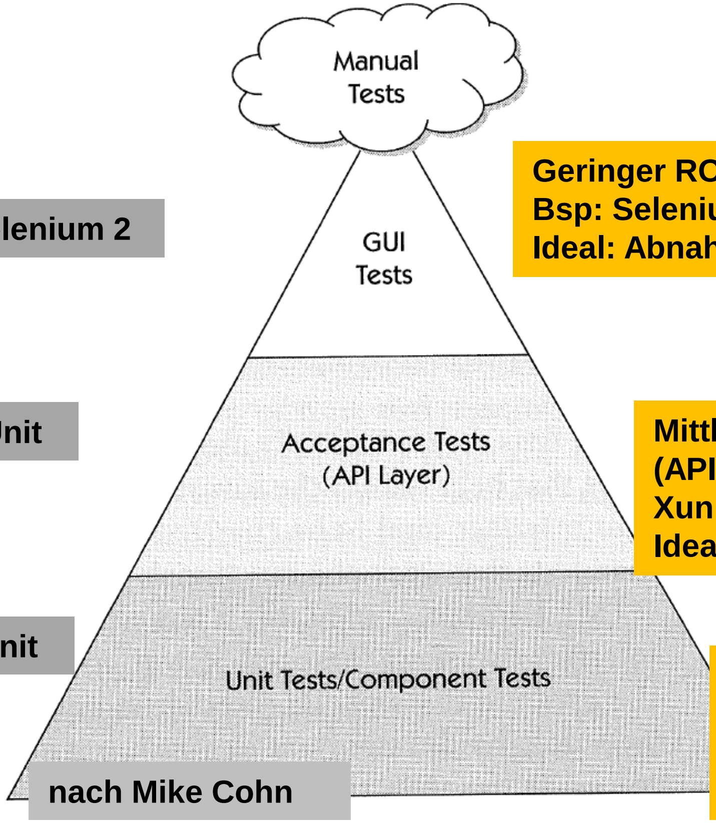
# *Wartbare Tests (2)*

- Testisolierung
  - Hintergrund: Wenn Tests zustandsbehaftet, ist das Finden der Ursache meist hoher Aufwand
  - Antipatterns



- *Constrained test order*—Tests expecting to be run in a specific order or expecting information from other test results
- *Hidden test call*—Tests calling other tests
- *Shared-state corruption*—Tests sharing in-memory state without rolling back
- *External-shared-state corruption*—Integration tests with shared resources and no rollback

# *Testautomatisierungs-Pyramide*



**Tool in der Übung: Selenium 2**

**Geringer ROI Tests inkl. GUI**
**Bsp: Selenium**
**Ideal: Abnahmerelevante Testdaten**

**Tool in der Übung: xUnit**

**Mittlerer ROI Tests hinter der GUI (API, Webservices)**
**Xunit, oft andere bsp. Fitnesse**
**Ideal: Abnahmerelevante Testdaten**

**Tool in der Übung: xUnit**

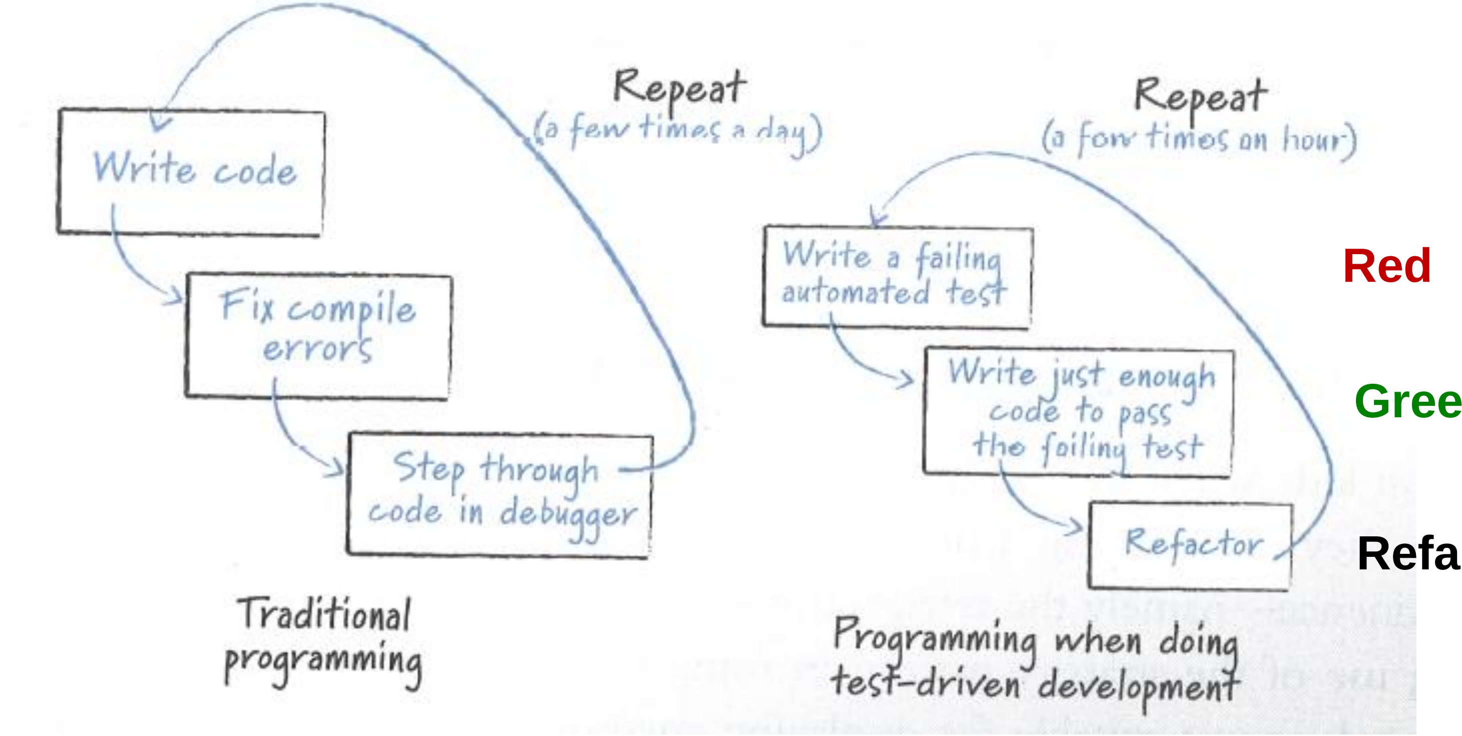
**Hoher ROI (isolierte Unit Tests)**
**Ideal: Alle Ebenen (auch GUI Klassen)**
**Bsp: DAO, Service, ManagedBeans**
**Typisch: synthetische Testdaten**

**nach Mike Cohn**

# *Unit Test allgemein Wiederholung*

- Happy Path (Äquivalenzklassen beachten)
  - Pragmatismus nicht verlieren (ROI bzw. Aufwand)
  - (=Red/Green im TDD)
- Relevante Fehlerpfade (nicht jede Option)
  - Pragmatismus nicht verlieren (ROI bzw. Aufwand)
  - (=Refactoring im TDD)
- TDD anwenden
  - Es hilft!
- Auf testbares Design achten (Prinzipien einfach)

# *Wiederholung: Test Driven Development (TDD)*


Repeat
(a few times a day)
Write code
Fix compile errors
Step through code in debugger
Traditional programming
Repeat
(a few times an hour)
Write a failing automated test
Write just enough code to pass the failing test
Refactor
Programming when doing test-driven development

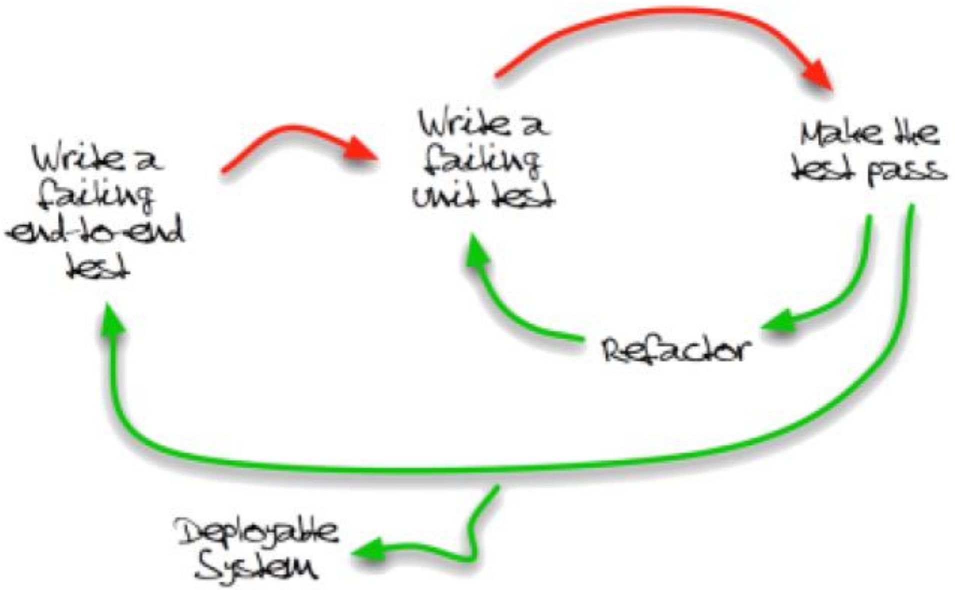
**Red**

**Green**

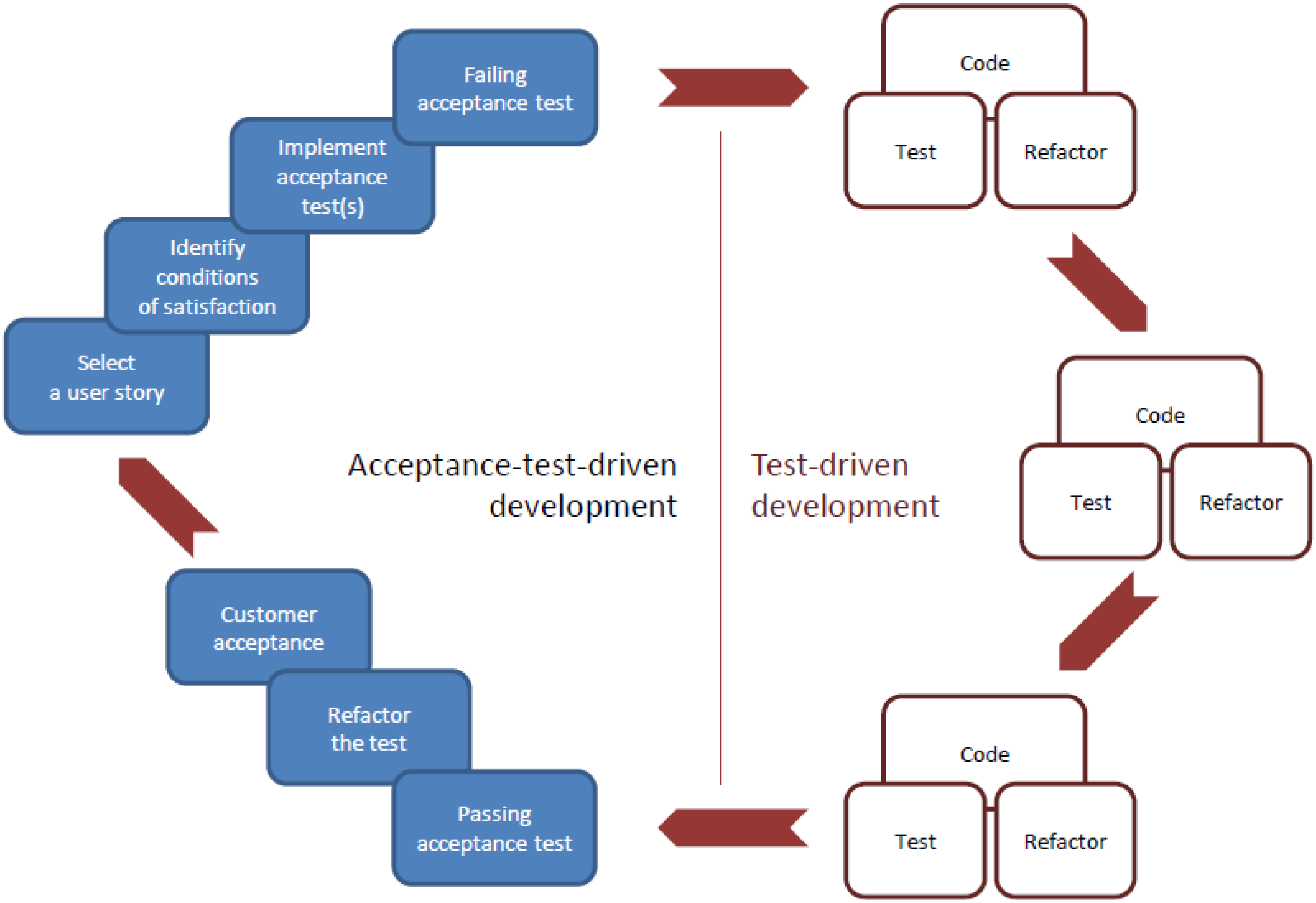
**Refactor**

# Exkurs: BDD (1)

- BDD (Behaviour Driven Development)
  - Hat Viele Synonyme: Acceptence-Test DD, Story Testing, Agile Acceptance Testing, Specification by Example…)
  - Behaviour = „Verhalten“ bzw. Akzeptanzebene ( bzw. Businessebene)
  - Erweitert TDD Mantra auf die Businessebene

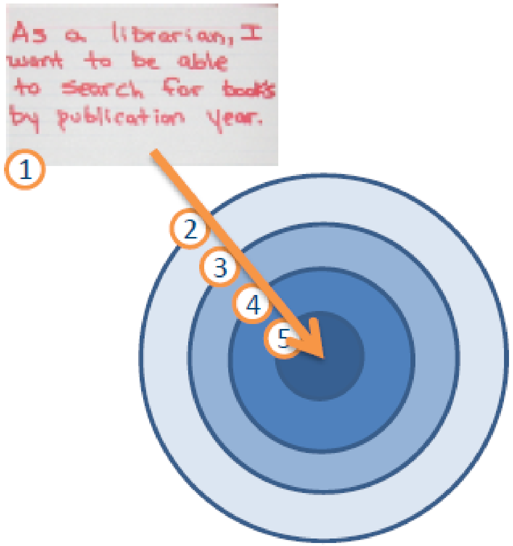


# Exkurs: BDD (2)


Failing acceptance test
Implement acceptance test(s)
Identify conditions of satisfaction
Select a user story
Customer acceptance
Refactor the test
Passing acceptance test
Acceptance-test-driven development
Test-driven development
Code
Test
Refactor
Code
Test
Refactor
Code
Test
Refactor

# Exkurs BDD (3) „Outside in Development“

- Incrementally work on
  - User story
  - Acceptance Criteria
  - Units


As a librarian, I want to be able to search for books by publication year.
1
2
3
4
5

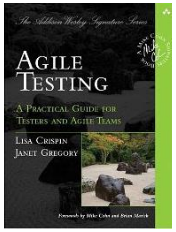
What development should focus on

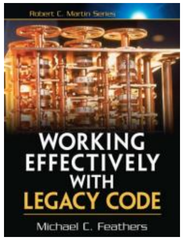
- Deliver business value
- Infrastructure only when needed

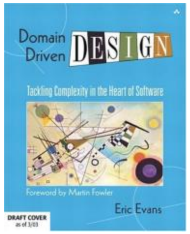
**Anwenden auch bei „klassischem“ TDD: hilft Fokus bewahren!**

# Referenzen

- Agile Testing
  - – Allgemeine Einführung
- The Art of Unit Testing
  - – .Net basiert aber pägnant und gut
  - – Beispiele aus dem Buch
- Hilfreich


The Addison Wesley Signature Series
Agile Testing
A Practical Guide for Testers and Agile Teams
Lisa Crispin
Janet Gregory




Domain Driven DESIGN
Tackling Complexity in the Heart of Software
Foreword by Martin Fowler
Eric Evans
DRAFT COVER