02291: System Integration

Week 3

Hubert Baumeister
huba@dtu.dk

DTU Compute
Technical University of Denmark

Spring 2018

Contents

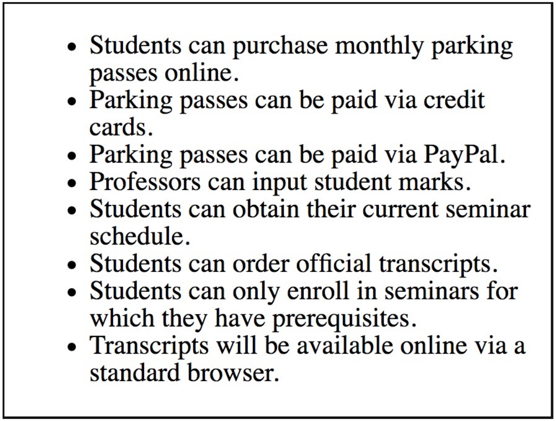
User stories

- Basic requirements documentation for agile processes
- Extreme Programming: Simplifies use cases
- ”story” the user tells about the the system
- Focus on features
  - ”As a customer, I want to book and plan a single flight from Copenhagen to Paris”.
- functional + non-functional requirement

  e.g. ”The search for a flight from Copenhagen to Paris shall take less than 5 seconds”
- user story cards: index cards

Example of user stories

Each line is one user story:



- Students can purchase monthly parking passes online.
- Parking passes can be paid via credit cards.
- Parking passes can be paid via PayPal.
- Professors can input student marks.
- Students can obtain their current seminar schedule.
- Students can order official transcripts.
- Students can only enroll in seminars for which they have prerequisites.
- Transcripts will be available online via a standard browser.

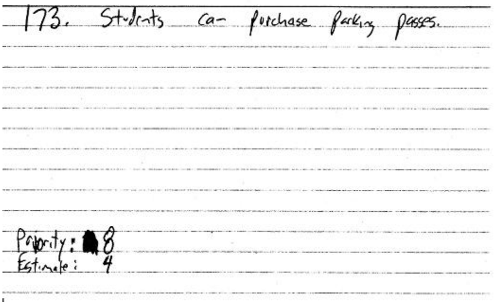
Scott Ambler 2003–2014 `http://www.agilemodeling.com/artifacts/userStory.htm`

Example of user story cards

"Use the simplest tool possible"

$\rightarrow$ index cards, post-its, ...

- electronically: e.g. Trello (`trello.com`)

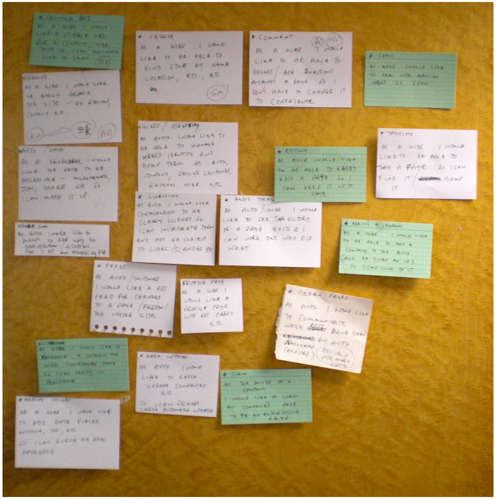


173. Students can purchase parking passes.

Priority: 8
Estimate: 4

Scott Ambler 2003–2014 `http://www.agilemodeling.com/artifacts/userStory.htm`

Use the simplest tool possible



* [illegible] URI
AS A USER I WOULD LIKE A STABLE URI FOR A COMPANY, USER, [illegible] SO I CAN BOOKMARK LINK TO STAFF

* SEARCH
AS A USER I WOULD LIKE TO BE ABLE TO FIND STAFF BY NAME LOCATION, ETC, ETC

* COMMENT
AS A USER I WOULD LIKE TO BE ABLE TO DISCUSS / ASK QUESTIONS AGAINST A PAGE SO I DON'T HAVE TO CHANGE IT TO CONTRIBUTE

* SPAM
AS AVOX I WOULD LIKE TO DEAL WITH MALICIOUS USERS EG SPAM

BROWSING
AS A USER I WOULD LIKE TO EASILY BROWSE THE SITE — BY REGION, COMPANY ETC.

USERS / IDENTITY
AS AVOX I WOULD LIKE TO BE ABLE TO MANAGE USERS IDENTITY AND GROUP THEM AS AVOX, COMPANY, PAYING CUSTOMER, RANDOM USER ETC

* EDITING
AS AVOX I WOULD LIKE TO BE ABLE TO EASILY EDIT A PAGE SO I CAN KEEP IT UP TO DATE

* TAGGING
AS A USER I WOULD LIKE TO BE ABLE TO TAG A PAGE SO I CAN FIND IT / ~~[illegible]~~ GROUP IT

APIS / DATA
AS A DEVELOPER I WOULD LIKE THE DATA TO BE ACCESSIBLE — MICROFORMATS, JSON, GEORSS ETC SO I CAN MASH IT UP

* LICENSING
AS AVOX I WOULD LIKE CONTRIBUTIONS TO BE CLEARLY LICENSED SO I CAN INCORPORATE THEM AND NOT BE SUBJECT TO LIBEL / SLANDER ETC

* AUDIT TRAIL
AS AVOX / USER I WOULD LIKE TO SEE THE HISTORY OF A PAGE EDITS SO I CAN WORK OUT WHO DID WHAT

* ADDING A COMPANY
AS A USER I WOULD LIKE TO BE ABLE TO ADD A COMPANY TO THE AVOX (AND BE GIVEN AN ID) SO I CAN LINK TO IT

[illegible]
As AVOX I would like to publish as RDF using the [illegible] / [illegible] for SAP and others eg EU

* FEEDS
AS AVOX / CUSTOMER I WOULD LIKE A RSS FEED FOR CHANGES TO A PAGE / PERSON / THE WHOLE SITE.

PROFILE PAGE
AS A USER I WOULD LIKE A PROFILE PAGE WITH RSS FEEDS ETC

* [illegible] PAGES
AS AVOX I WOULD LIKE TO COMMUNICATE WHICH ~~[illegible]~~ EDIT HAVE BEEN ~~[illegible]~~ BY AVOX ~~[illegible]~~ VALIDATED (AUDITED) POSSIBLY WITH [illegible] DATA

* CITATIONS
AS AVOX I WOULD LIKE TO ENCOURAGE A CITATION FOR USER CONTRIBUTED DATA SO I CAN VERIFY ITS PROVENANCE

* BATCH UPDATES
AS AVOX I WOULD LIKE TO BATCH UPDATE COMPANIES ETC SO I CAN PROCESS LARGE AUTOMATED UPDATES

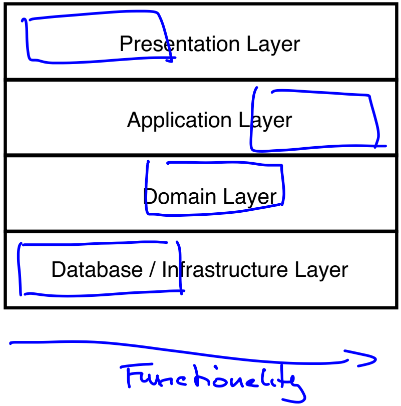
* CLAIM
AS THE OWNER OF A COMPANY I WOULD LIKE TO CLAIM MY COMPANIES PAGE TO BE AN AUTHORITATIVE EDITOR

* ADDING VALUES
AS A USER I WOULD LIKE TO ADD DATA FIELDS [illegible], [illegible], ETC SO I CAN ENRICH THE AVOX DATABASE

Paul Downey 2009 `https://www.flickr.com/photos/psd/3731275681/in/photostream/`

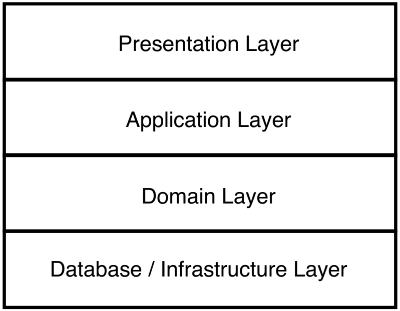
Two different ways of building the system

Traditional: Build the system by layer/framework

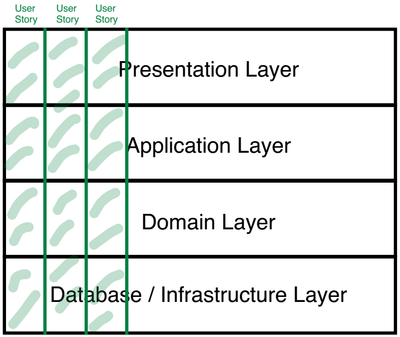


Two different ways of building the system

Traditional: Build the system by layer/framework



Agile: Build the system by user story



Comparision: User Stories / Use Cases

Use Case

- Advantage: Overview over functionality
- Disadvantage: Use case driven development

Use Story

- Advantage: user story driven
- Disadvantage: Overview over the functionality is lost

Example: Login

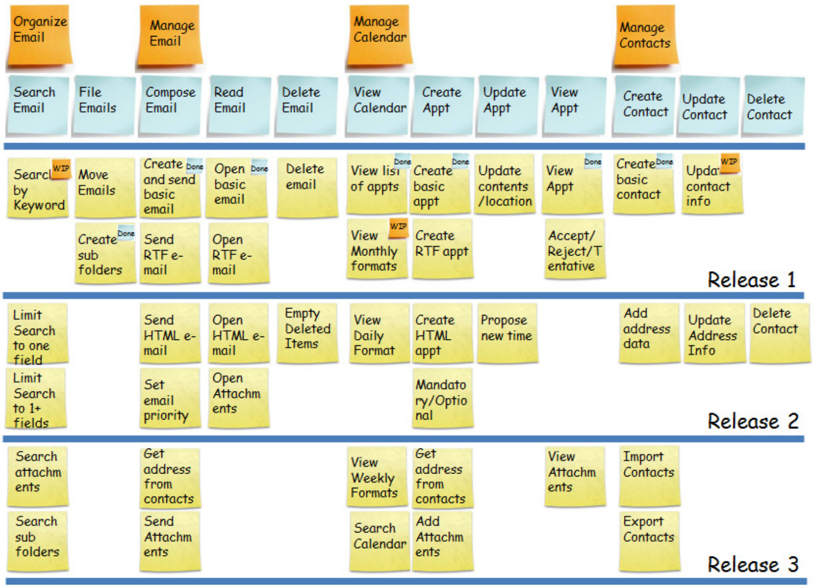
Use case

- name: Login
- actor: User
- main scenario
  1. User logs in with username and password
- alternative scenario
  - 1’ User logs in with NEMID

User stories

1. User logs in with username and password
2. User logs in with NEMID

User Story Maps



Shrikant Vashishtha `http://www.agilebuddha.com/wp-content/uploads/2013/02/IMAG0144.png`

Combining Use Cases and User Stories

1. Use case diagram: Overview
2. Use case scenarios give user stories
3. User story driven implementation by priority

Problem: Changing Requirements

Requirements can change

- Feedback: design, implementing, using

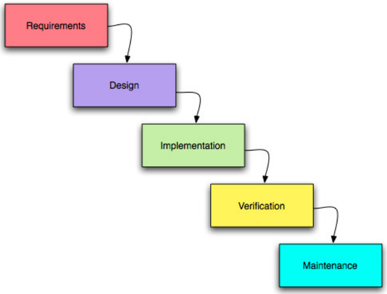
$\rightarrow$ clarification, changing, and new requirements

- The business case changes

Different type of software

- s-type: mathematical function, sorting: complete specfication
- p-type: real world problems, e.g., chess: modelling the real world
- e-type: embeded into socia-technical systems. Requirements change as the environment changes. System changes the environment: e.g., operating system

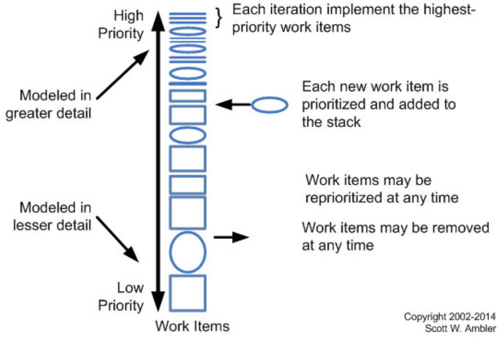
Requirements management: Waterfall



- Defined requirement management process
  - E.g. Agreement of all stakeholders
- Changed / new requirements
  - Cost of change not predictable

  $\rightarrow$ Avoid changing/new requirements if possible

Requirements management: Agile Methods



High Priority
Each iteration implement the highest-priority work items
Modeled in greater detail
Each new work item is prioritized and added to the stack
Work items may be reprioritized at any time
Modeled in lesser detail
Work items may be removed at any time
Low Priority
Work Items
Copyright 2002-2014 Scott W. Ambler

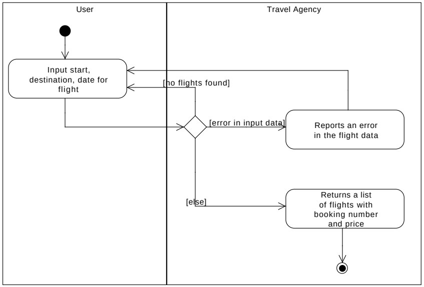
Scott Ambler 2003–2014 `http://www.agilemodeling.com/artifacts/userStory.htm`

- Cost of change
  - New / changed requirements not done yet: zero costs
  - Changed requirements already done: the cost of a requiment that can not be implemented

Contents

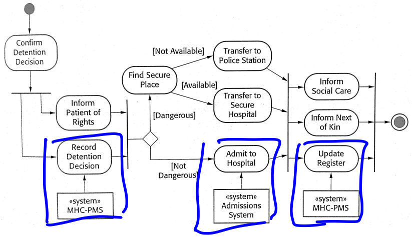
Examples of the use of Activity Diagrams

Shows main- and alternative scenarios of use cases



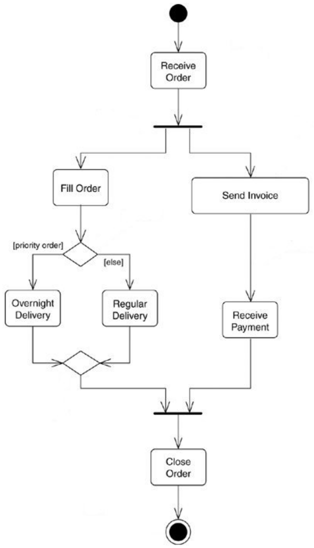
User
Travel Agency
Input start, destination, date for flight
[no flights found]
[error in input data]
Reports an error in the flight data
[else]
Returns a list of flights with booking number and price

Business Processes

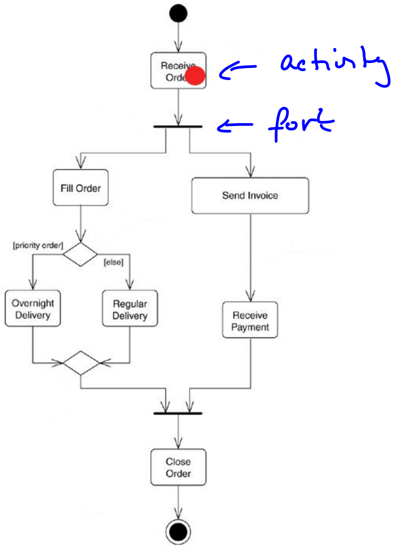


Ian Sommerville, Software Engineering – 9, 2010

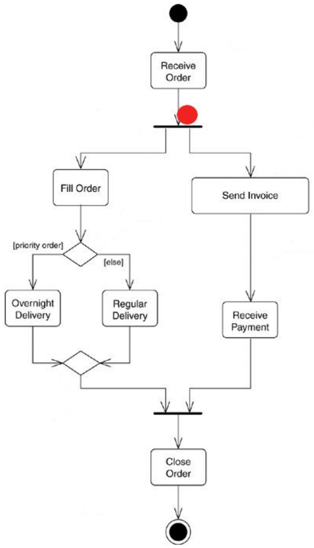
Activity Diagram Concepts



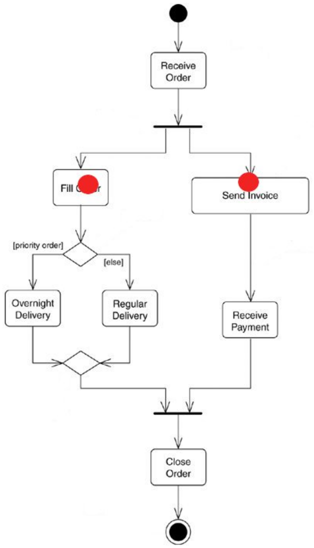
Activity Diagram Execution



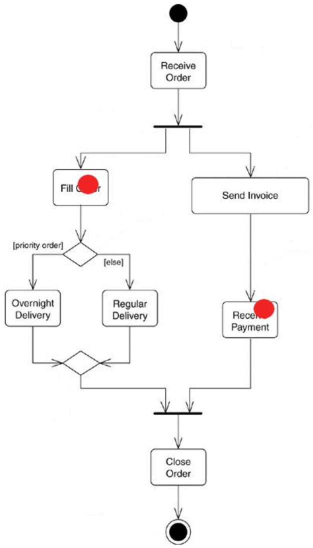
Activity Diagram Execution



Activity Diagram Execution

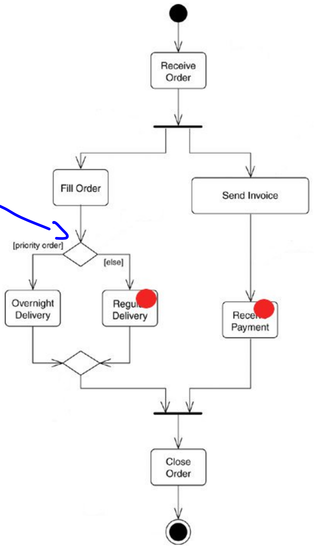


Activity Diagram Execution

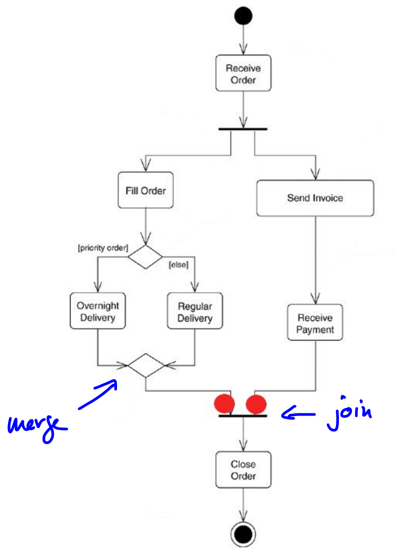


Activity Diagram Execution

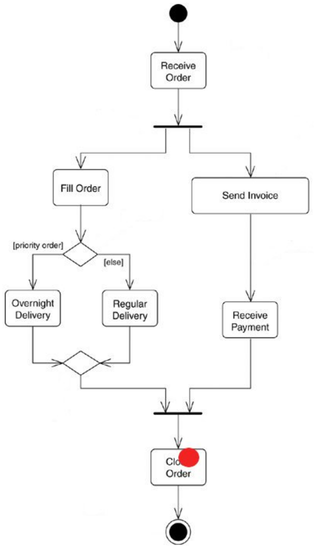
decision



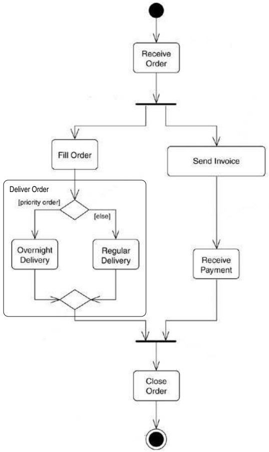
Activity Diagram Execution



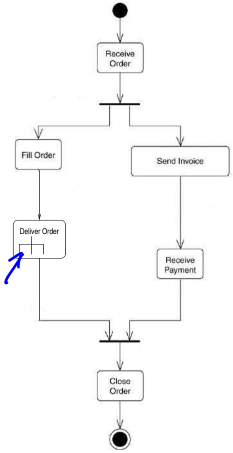
Activity Diagram Execution



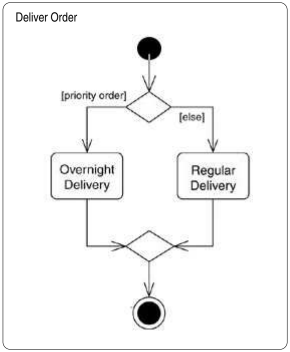
Subactivities



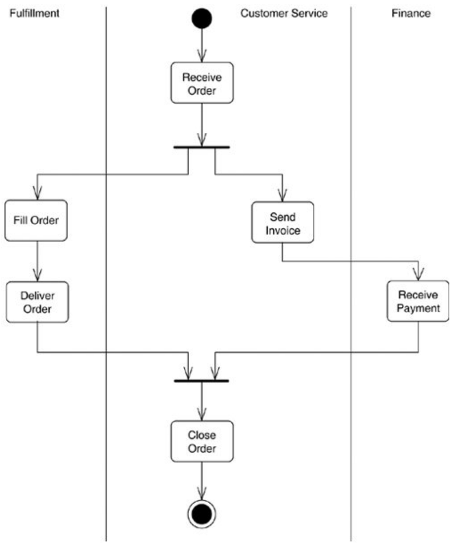
Subactivities



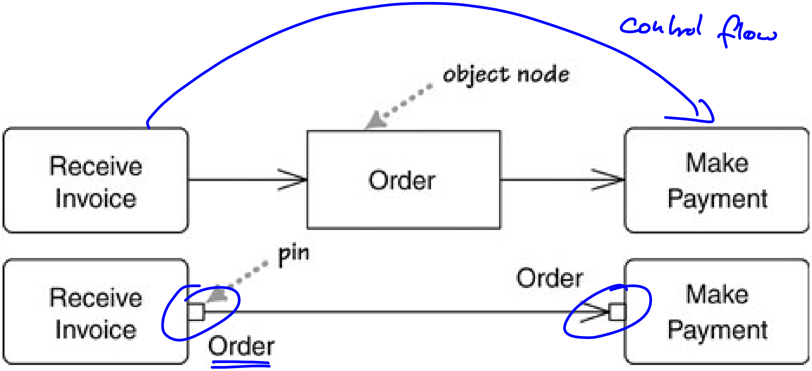
Subactivity Deliver Order



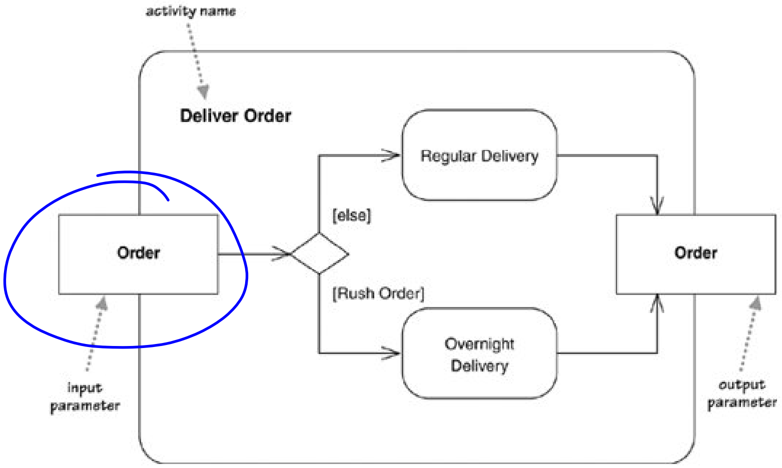
Swimlanes / Partitions



Objectflows / Dataflows



Pins



activity name
Deliver Order
Regular Delivery
[else]
Order
[Rush Order]
Overnight Delivery
Order
input parameter
output parameter

Contents

Why testing?

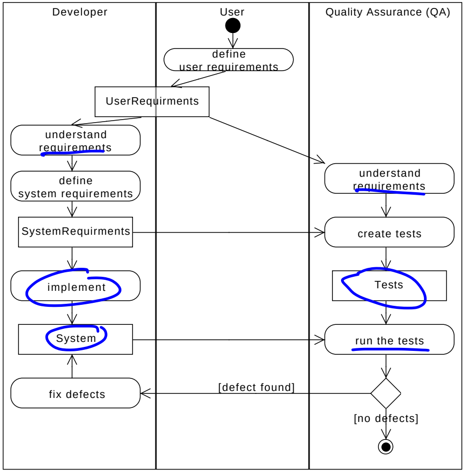
- Validation testing
  - Tests that the user requirements are satisfied
  - *Have we built the right system?*
- Defect testing
  - Tests that the system has no defects
  - *Have we built the system right?*
- Documentation
  1. System properties
  2. Surprising or non-intuitive behaviour of the system
  3. Bugs and bug fixes, also known as regression testing (Prevents from reintroducing the bug later)
- Experiment with the system

Types of tests

1. Developer tests (basically validation testing)
   a) Unit tests (single classes and methods)
   b) Component tests (single components = cooperating classes)
   c) System tests / Integration tests (cooperating components)
2. Release tests (validation and defect testing)
   a) Scenario based testing
   b) Performance testing
3. User tests
   a) Acceptance tests

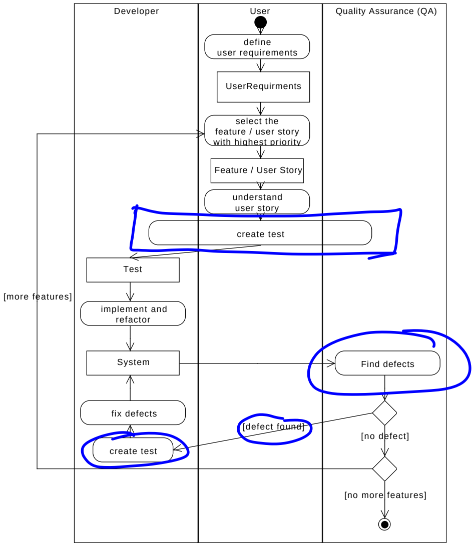
Acceptance Tests

Traditional testing



Acceptance Tests in Agile processes

Test-Driven Development



Example of acceptance tests

- Use case

  name: Login Admin
  actor: Admin
  precondition: Admin is not logged in
  main scenario

  1. Admin enters password
  2. System responds true

  alternative scenarios:

  1a. Admin enters wrong password
  1b. The system reports that the password is wrong and the use case starts from the beginning

  postcondition: Admin is logged in

Manual tests

Successful login

Prerequisit: the password for the administrator is “adminadmin”

Input	Step	Expected Output	Fail	OK
	Startup system	“0) Exit” “1) Login as administrator”		✓
“1”	Enter choice	“password”		✓
“adminadmin”	Enter string	“logged in”	✓	

Failed login

Prerequisit: the password for the administrator is “adminadmin”

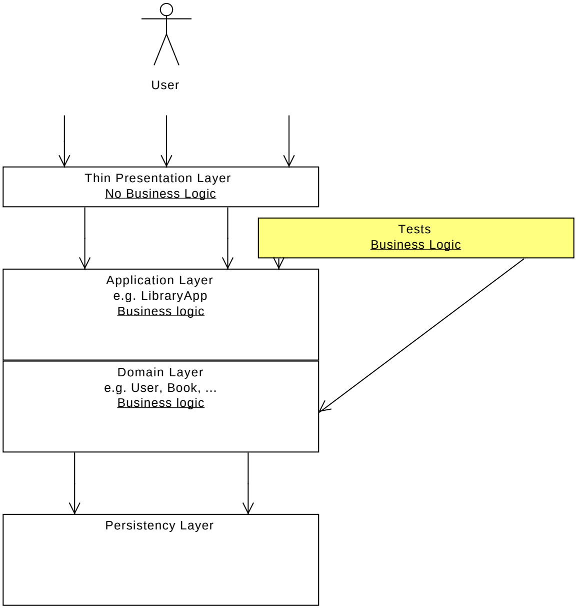
Input	Step	Expected Output	Fail	OK
	Startup system	“0) Exit” “1) Login as administrator”		✓
“1”	Enter choice	“password”		✓
“admin”	Enter string	“Password incorrect” “0) Exit” “1) Login as administrator”		✓

Manual vs. automated tests

- Manual tests should be avoided
  - Are expensive; can't be run often
- Automated tests
  - Are cheap; can be run often
- Robert Martin (Uncle Bob) in `http://www.youtube.com/watch?v=hG4LH6P8Syk`
  - manual tests are immoral from 36:35
  - how to test applications having a UI from 40:00
- How to do UI tests?

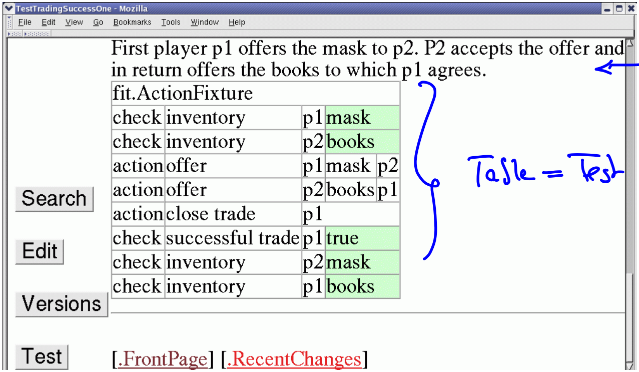
  $\rightarrow$ Solution: Test under the UI

Test under the UI



Language to express acceptance tests

Framework for integrated tests (Fit)



TestTradingSuccessOne - Mozilla
File Edit View Go Bookmarks Tools Window Help
Search
Edit
Versions
Test

First player p1 offers the mask to p2. P2 accepts the offer and in return offers the books to which p1 agrees. ←

fit.ActionFixture				
check	inventory	p1	mask	
check	inventory	p2	books	
action	offer	p1	mask	p2
action	offer	p2	books	p1
action	close trade	p1		
check	successful trade	p1	true	
check	inventory	p2	mask	
check	inventory	p1	books	

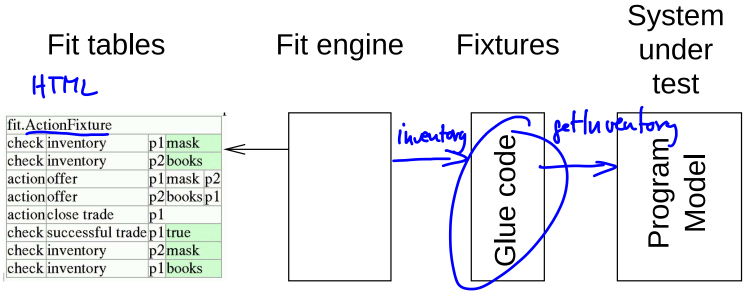
} Table = Test

[.FrontPage] [.RecentChanges]

Fit Framework

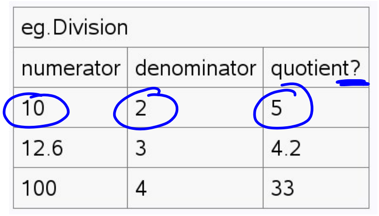
- Framework for integrated test (Fit)
  - Goal: Automated acceptance tests
  - Ward Cunningham (CRC cards, Wiki, patterns, XP)
  - Tests are HTML tables
  - → Customer formulates tests
  - `http://fit.c2.com`
- Fitnesse
  - Standalone Wiki with Fit integration
  - `http://www.fitnesse.org`
  - → use this to play around with Fit tests
  - Download `fitnesse-standalone.jar`, run
    `java -jar fitnesse-standalone.jar -p 8080`
    and go to `localhost:8080`
  - Set the class path with `!path ...`
  - Compile with
    `javac -cp fitnesse-standalone.jar:. ...`

Fit Framework III



fit.ActionFixture				
check	inventory	p1	mask	
check	inventory	p2	books	
action	offer	p1	mask	p2
action	offer	p2	books	p1
action	close trade	p1		
check	successful trade	p1	true	
check	inventory	p2	mask	
check	inventory	p1	books	

Column fixture



eg.Division		
numerator	denominator	quotient?
10	2	5
12.6	3	4.2
100	4	33

```
public class Division extends ColumnFixture {
   public double numerator;
   public double denominator;
   public double quotient() {
      Div sut = new Div();
      return sut.divide(numerator, denominator);
   }
}

public class Div {
  public double divide(doube numerator, double denominator) {
    return numerator / denominator;
   }
}
```

Row fixture

fitnesse.fixtures.PrimeNumberRowFixture
prime
3
2
5
7
11

```
public class PrimeNumberRowFixture extends RowFixture {
   public Object[] query() throws Exception {
      Primes sut = new Primes();
      PrimeData[] array = new PrimeData[5];
      int[] primes = sut.primes(5);
      for (int i = 0; i < 5; i++) {
         PrimeData pd = new PrimeData();
         pd.setPrime(primes[i]);
         array[i] = pd;
      }
      return array;
   }

   public Class getTargetClass() {
      return PrimeData.class;
   }
}
```

Action fixture

Action Fixture.		
start	fitnesse.fixtures.CountFixture	
check	counter	0
press	count	
check	counter	1
press	count	
check	counter	2
enter	counter	5
press	count	
check	counter	6

Glue

```
public class CountFixture extends Fixture {
   private Counter sut = new Counter();
   public void count() { sut.count(); }
   public int counter() { return sut.getCounter(); }
   public void counter(int c) { sut.setCounter(c); }
}
```

```
public class Counter {
   int counter = 0;
   public void count() { counter++;}
   public int getCounter() { return counter;}
   publc void setCounter(int c) { counter = c;}
}
```

Action Fixture: From use case to test

- Interactions
  - The user does something with the system
    - *press*: performing one action: press a button: e.g. press | count
    - *enter*: performing one action with a parameter: e.g. enter | name | Anne
  - The system changes because what the user did
    - *check*: e.g. check | counter equals | 3
- Preconditions / postconditions
  - *check*: e.g. check | user registered | true

Travel Agency: detailed use case *list available flights*

name: list available flights
description: the user checks for available flights
actor: user

main scenario:

1. The user provides information about the city to travel to and the arrival and departure dates
2. The system provides a list of available flights with prices and booking number

alternative scenario:

1a. The input data is not correct (see below)

  2. The sytem notifies the user of that fact and terminates and starts the use case from the beginning

2a. There are no flights matching the users data

  3. The use case starts from the beginning

note: The input data is correct, if the city exists (e.g. is correctly spelled), the arrival date and the departure date are both dates, the arrival date is before the departure date, arrival date is 2 days in the future, and the departure date is not more then one year in the future

- Acceptance Tests:

  `http://www2.compute.dtu.dk/courses/02291/examples/test/travel_agency_fit_tests.pdf`

Testing in the system integration course

- Learn how to write test
  - → Acceptance tests as tables
- Check that tests and scenarios describe the same interactions
- Explain the tables and their kind (column-, row-, or action fixtures)
- Just the tables: LaTeX, Word, …