On a blustery autumn evening five old friends met in the backroom of one of the city's oldest and most private clubs. Each had traveled a long distance — from all corners of the world — to meet on this very specific day… October 2, 1900 — 28 years to the day that the London eccentric, Phileas Fogg accepted and then won a £20,000 bet that he could travel Around the World in 80 Days.

When the story of Fogg’s triumphant journey filled all the newspapers of the day, the five attended University together. Inspired by his impetuous gamble, and a few pints from the local pub, the group commemorated his circumnavigation with a more modest excursion and wager – a bottle of good claret to the first to make it to Le Procope in Paris.

Each succeeding year, they met to celebrate the anniversary and pay tribute to Fogg. And each year a new expedition (always more difficult) with a new wager (always more expensive) was proposed. Now at the dawn of the century it was time for a new impossible journey. The stakes: $1 Million in a winner-takes-all competition. The objective: to see which of them could travel by rail to the most cities in North America — in just 7 days. The journey would begin immediately…

Ticket to Ride is a cross-country train adventure. Players compete to connect different cities by laying claim to railway routes on a map of North America.

For 2 - 5 players ages 8 and above
30 - 60 minutes

Patrick Schäfer
patrick.schaefer@hu-berlin.de

Semesterprojekt: Implementierung eines Brettspiels, WS 18/19
Today

• New User Story:
  – Observer-Mode
  – Backlog-Grooming

• Technical Talk: Unit tests & continuous integration

• This week:
  – technical refinement for the new user stories
  – finalize tasks in your sprint backlog (incl. tests, code review)

• Next Monday, 13 c.t.
  – Sprint #1 Review Meeting; bring a laptop & presentable prototype
  – Sprint #2 Sprint planning; kickoff; present your Sprint Backlog
How Software Used to Be Developed

• software developed in teams
• software is divided in modules
• modules are assigned to teams
• modules have to be integrated at some point
• integration is done manually at release time
What Are the Issues?

• effects of changes (bugfixes, new features) are hard to predict
  → Unit Tests

• no feedback (by the product owner) possible before release
  → Continuous Integration

• unnecessarily complicated integration process
• bad software quality
• integration hell: stress and frustration towards the end of a project
Unit Tests

unit test: systematic, automated test of a software component

• unit: smallest testable part of an application
  – in object-oriented programming, this is usually a method (or a class)
• the unit is tested in isolation (of other units)

• usually written in the same language as the tested software
• tests should be written by the developer (of the unit)
• can be written prior or concurrent to unit development
• tests can succeed or fail
Advantages of Unit Testing

unit tests ... 

• ... facilitate the design of robust code 
  (bugs make it through only if the unit and its test are poorly designed) 
• ... provide immediate feedback on the effect of changes in the code base 
• ... serve as a to-do list subsequent to changes in the code base 
• ... help define what a piece of code is (and isn’t) supposed to do
Best Practices

1. test the complete intended behavior of the unit, including
   a) expected cases (e.g., sort an unsorted array)
   b) special cases (e.g., sort already sorted array)
   c) boundary conditions (e.g., sort empty array)

2. test every behavior only once (no redundant test)

3. test only one unit at a time

4. design tests independent of the application’s state

5. design tests independent of external resources

6. name unit tests clearly and consistently

7. whoever breaks a working unit is responsible for fixing it
A concrete Example

- We want to add the functionality to claim routes
- From the rule book:

  To claim a route, a player must play a set of cards equal to the number of spaces in the route. A set of cards must be of the same type. Most routes require a specific type of set. For example a Blue route must be claimed using blue-colored Passenger Car cards. Some routes – those that are Gray colored – can be claimed using a set of cards of any one color.

  [...] Locomotives are Multi-colored and act as a wild card that can be part of any set of cards when claiming a route. [...]
A naïve Example

```java
class de.huberlin.wbi {

class Player {
    static enum PassengerColor {
        Blue, Black, Red, Rainbow
    }
    int[] playerCards = new int[4];

    // Claim a route between two adjacent cities using
    // the payByColor-passenger-cards in our hand
    public boolean claimRoute(PassengerColor payByColor,
                               int routeCost,
                               PassengerColor routeColor) {
        int currentCards = playerCards[payByColor.ordinal()];
        // Pay for a route between two adjacent cities on the map
        if (payByColor == routeColor
            || payByColor == PassengerColor.Rainbow
            || routeColor == PassengerColor.Rainbow) {
            if (currentCards >= routeCost) {
                playerCards[payByColor.ordinal()] -= routeCost;
                return true;
            }
        }
        // we cannot buy the route
        return false;
    }
}

class PlayerTest {
    @Test
    public void testContains() {
        Player p = new Player();
        Arrays.fill(p.playerCards, 3);

        assertTrue(p.claimRoute(Player.PassengerColor.Black, 1, Player.PassengerColor.Black));
        assertTrue(p.claimRoute(Player.PassengerColor.Red, 1, Player.PassengerColor.Rainbow));
        assertTrue(p.claimRoute(Player.PassengerColor.Rainbow, 1, Player.PassengerColor.Blue));
        assertFalse(p.claimRoute(Player.PassengerColor.Black, 2, Player.PassengerColor.Blue));
        assertFalse(p.claimRoute(Player.PassengerColor.Red, 10, Player.PassengerColor.Red));

        // Blue, Black, Red, Rainbow
        System.out.println(Arrays.toString(p.playerCards));
        for (int c : p.playerCards) {
            assertTrue(c >= 0);
        }
    }
}
```

How to test this:
- Special-Cases?
- Expected Cases?
- Boundary Cases?
Rainbow Cards

• From the rule book:
  To claim a route, a player must play a set of cards equal to the number of spaces in the route. A set of cards must be of the same type. Most routes require a specific type of set. For example a Blue route must be claimed using blue-colored Passenger Car cards. Some routes – those that are Gray colored – can be claimed using a set of cards of any one color.

  [...] **Locomotives are Multi-colored and act as a wild card that can be part of any set of cards when claiming a route.** [...]

• So lets add this functionality
public boolean claimRoute(
    PassengerColor payByColor,
    int routeCost,
    PassengerColor routeColor
) {
    int currentCards = playerCards[payByColor.ordinal()];
    int rainBowCards = playerCards[PassengerColor.Rainbow.ordinal()];

    // Pay for a route between two adjacent cities on the map
    if (payByColor == routeColor
        || payByColor == PassengerColor.Rainbow
        || routeColor == PassengerColor.Rainbow}) {
        // no rainbow cards needed
        if (currentCards >= routeCost) {
            playerCards[payByColor.ordinal()] -= routeCost;
            return true;
        }
        // rainbow cards are needed
        else if (currentCards + rainBowCards >= routeCost){
            playerCards[payByColor.ordinal()]= 0;
            playerCards[PassengerColor.Rainbow.ordinal()] -= (routeCost - currentCards);
            return true;
        }
    }
    // we cannot buy the route
    return false;
}
Demo: Unit Tests

Repository:
https://github.com/hu-berlin-semesterprojekte/cidemo
Continuous Integration (CI)

- automatically test and merge all units into an integrated software (multiple times a day)

- **every change** (e.g., git push) in the software triggers a **new build**
- **unit tests** are executed to determine the success of a build
- gives feedback in form of **reports**

- requires version control and **build automation** for downloading dependencies, compiling code, and running tests
  - build automation tools for Java: **Maven**, Ant, Gradle

- builds can succeed or fail
CI in Practice

Version Control System

Commit

Developer

Report

CI Server

Fetch HEAD

Create build

Create build
Advantages of Continuous Integration

with continuous integration, we ...

• prevent “integration hell” early

• always know the latest stable version of our software

• instant feedback if a developer’s work in progress breaks the stable version

• can automatically test different setups
  – different databases
  – multiple versions of 3rd party libraries
  – different configurations
Best Practices

1. design meaningful unit tests for your software modules
2. commit frequently; keep iterations small
3. keep your tests fast; keep the build fast
4. don’t (ever) commit into a stable branch when the build is broken
Travis CI

- open-source continuous integration service / server
- website: https://travis-ci.org
- coupled with GitHub

- easy to set up:
  1. sign in using your GitHub account
  2. select repositories that Travis should build
- build is configurable via .travis.yml file
  - YAML is a popular data serialization file format, similar to XML or JSON
Travis CI
Travis CI: Build History
Using Travis YML

1. Write Unit Tests
2. Configure build automation
   - **pom.xml** for Maven (Java)
   - **build.gradle** for Gradle (Java)
3. Configure travis to use build automation
   - **.travis.yml**:
     - language: java
     - jdk: - oraclejdk8
     - sudo: false
     - script: mvn clean verify
Demo: Continuous Integration

Repository:
https://github.com/hu-berlin-semesterprojekte/cidemo
Further Reading

• unit tests in Java using JUnit:
  http://www.frankwestphal.de/UnitTestingmitJUnit.html

• build automation in Java using Maven:
  https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

• unit tests in Unity using Unity Test Tools:
  https://unity3d.com/learn/tutorials/topics/production/unity-test-tools

• continuous integration in Unity using GitHub and Travis CI:
  https://stablekernel.com/continuous-integration-for-unity-5-using-travisci/
Next steps

• familiarize yourself with unit tests & continuous integration
  – further reading (→ last slide)
  – start testing and integrating (→ user story “Continuous Integration”)

• this week (w/o POs)
  – finalize tasks in your sprint backlog (incl. tests, code review)
  – mid-week: technical refinement for the new user stories

• Next Monday, 13:30
  – Sprint #1 Review Meeting; bring a laptop & presentable prototype
  – Sprint #2 Sprint planning; kickoff; present your Sprint Backlog

• Further technical talks...

• Questions?