Lecture 1. Course structure

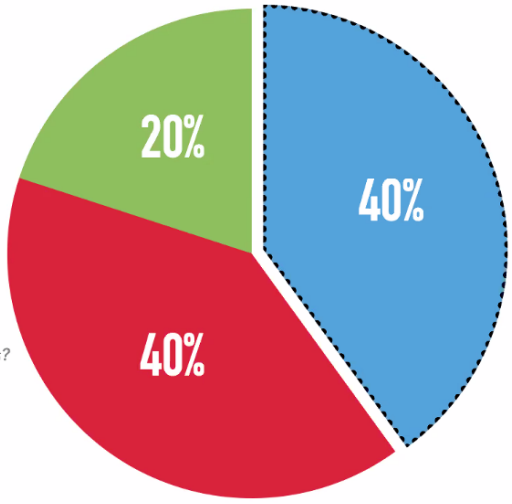
COURSE SCHEDULE — DETAILED VERSION ON BRIGHTSPACE

	3.1 Feb 8th	3.2 Feb 15th	3.3 Feb 22th	3.4 Mar 1st	3.5 Mar 8th	3.6 Mar 15th	3.7 Mar 22th	3.8 Mar 29th	3.9 Apr 5th	3.10 Apr 5th
Mo	**Lecture** Intro + Git	**Lecture** Requirem. Eng	**Online Module** Inf. Literacy							
			Code of C. + **Draft** Backlog	**Deadline** Project Plan		**Draft** Design Doc.				
Tu	**Lab**	**Lab** CTA	**Lab** CTA	**Lab**	**Lab**	**Lab** CTA	**Lab**	**Lab**	**Lab**	
We	**Lecture** Teamwork	**Lecture** Responsible CS	**Lecture** Spring + Testing							**Exam week** Presentation + Q&A
Th	**Lab**	**Lab** CTA	**Lab** CTA	**Lab**	**Lab**	**Lab** CTA	**Lab**	**Lab**	**Lab**	CTA
Fr	**Lecture** H-C Interaction	**Lecture** GitLab		**Buddycheck** #1		**Buddycheck** #2		**Buddycheck** #3		
	Deadline Git assignment		**Deadline** Backlog + InfLit	**Deadline** Everyone 1 MR					**Final Deadline**	
	THEORY / RESEARCH / PROTOTYPING			DEVELOPMENT						

GRADING

PROCESS (40%)

- **Code of Conduct**
  Agreements within your team on how you'll work together
- **Backlog**
  Your "todo-list", more in lecture on Requirements Engineering
- **Planning and task distribution**
  Initial planning, dealing with changes, and evenly distributing the work
- **Code contribution and code reviews**
  Git usage and the review process of the work of your fellow team members
- **Individual evaluation in final peer reviews**
  Did you take feedback into account, and did you provide valuable feedback to others?



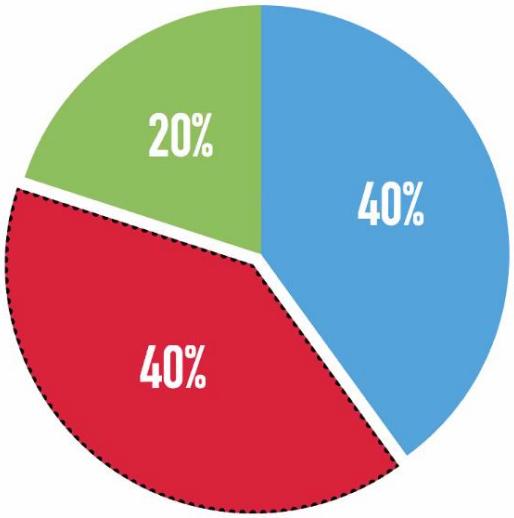
All items have equal weights.
All items require a minimum grade of 5.0

GRADING

PRODUCT (40%)

Quality of the product itself

- Implemented features
  Does the final product fulfil the requirements of the client?
- Testing
  Is the product well tested?
- Quality (code style, documentation, etc.)
  Consistency of the code style, sufficient documentation in- and outside the code.

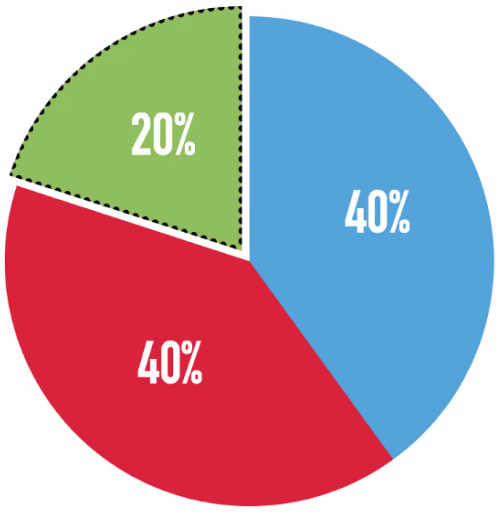


All items have equal weights.
All items require a minimum grade of 5.0

GRADING

DESIGN DOCUMENT (20%)

- 8% Value-sensitive Design (Responsible CS)
  Reflect on values and stakeholders of the system.
- 8% Human-Computer Interaction (HCI)
  Design your interface following basic principles of HCI.
- 4% Information Literacy
  Searching for and using/citing information the right way.

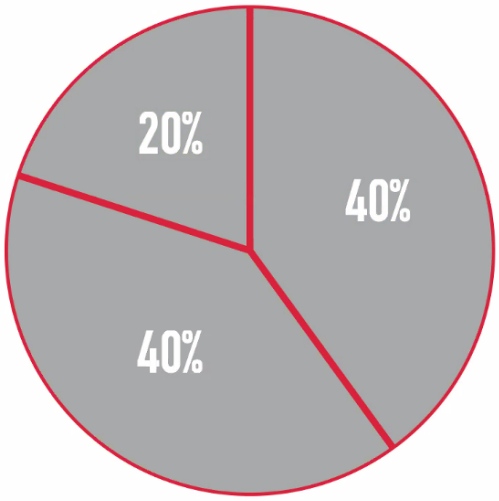


All items require a minimum grade of 5.0

GRADING

MINIMUM PASSING REQS

- **W1:** Git assignment
- **W3:** Information Literacy module
- **W4:** Have a Merge Request (MR) merged
- **W4+W6+W8:** BuddyCheck (peer reviews)
  - 3x, mark the dates in your calendar!
  - You may only miss one
- Participate actively in process *and* product
  - Attend meetings and be involved in discussions
  - Submitting your own work (a.o. code) every week
  - You cannot be responsible solely for docs/design/etc.
    - Everyone should be involved in all aspects. (code, docs, design, ...)
  - **TAs will monitor everyones activity**
- *More details on Brightspace / StudyGuide.*



THIS YEAR...

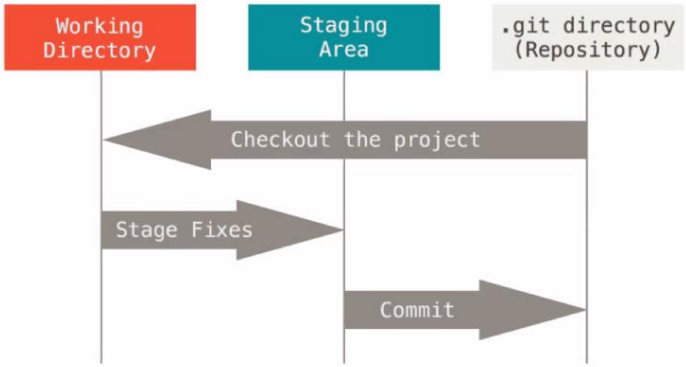
SOFTWARE TO SUPPORT LECTURERS

- 🎓 A platform to be used during lectures, using which students can ask questions and can give live feedback to the lecturers.
- 🛠️ Build
  - A **backend** that exposes an JSON API *(note: do **not** use WebSockets for this!)*
    Framework: Spring MVC
  - A **client application** that consumes this API
    Framework: OpenFX
- 💾 Information must be stored in a database
  (can be hosted locally, or host it yourself (you can opt to use: projects.ewi.tudelft.nl))
- 🏆 Certificates for best solution(s)
  🖼️ We aim to use the best project in actual lectures

Lecture 2. Git Version control

Git Terminology

- Entities
  - Working Directory
  - Staging Area / Index
  - Repository
- Activities
  - (Un-)track a File
  - Add changes
  - Commit

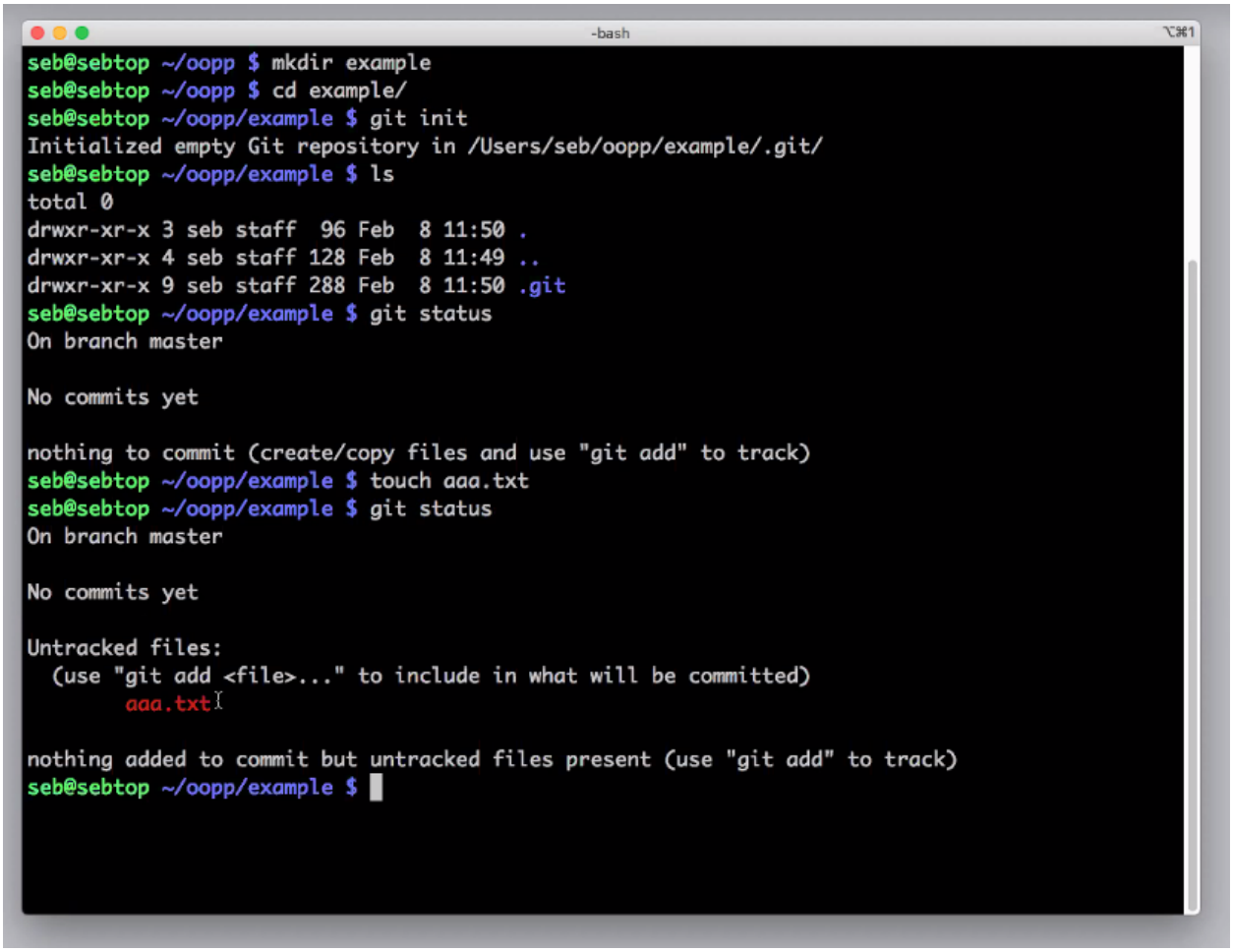


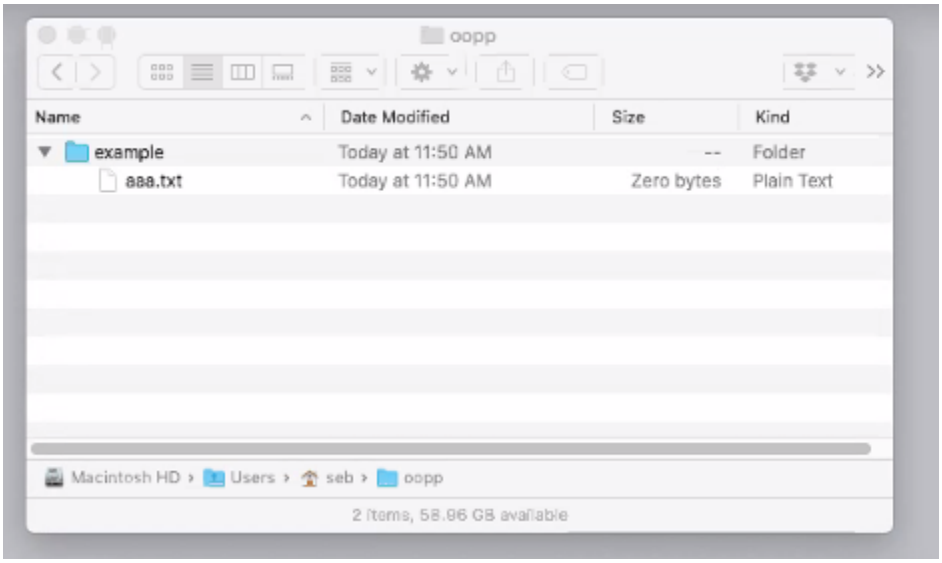
- A “Clean” working directory: no open changes in tracked files

git config – How you appear in commits

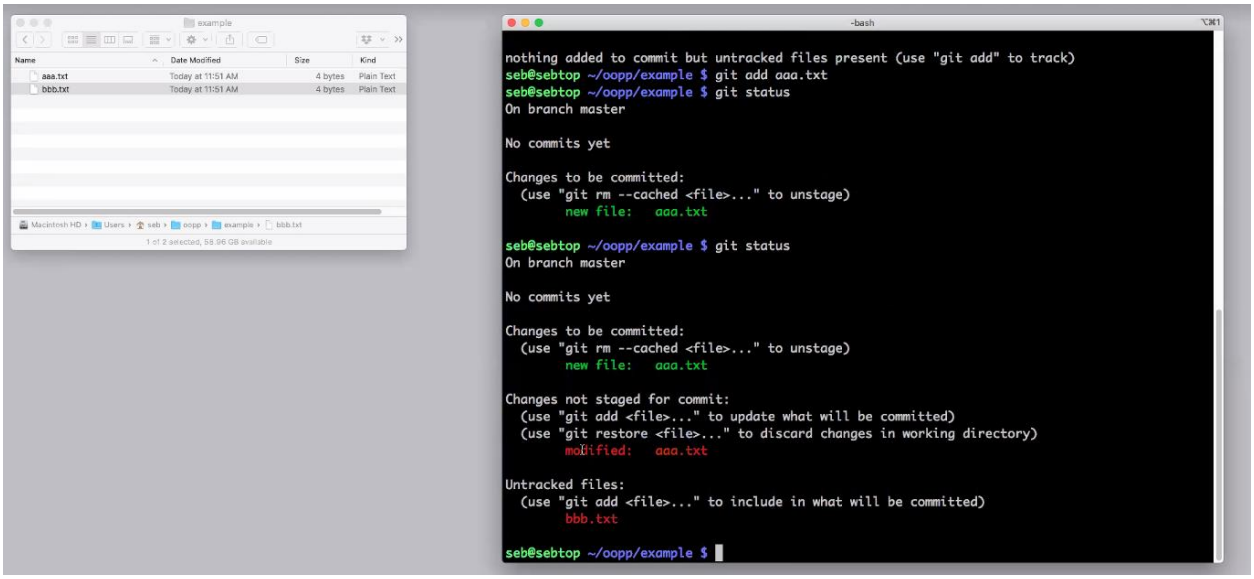
- git config --global user.name “Anna Beta”
- git config --global user.email “email@your.domain”

Please note: Powerpoint likes to replace hyphens. If you encounter a long hyphen like “–” in the slides, Powerpoint decided to take action. Unfortunately, it does this for both single and double hyphen. If in doubt, just try both.


-bash
seb@sebtop ~/oopp $ mkdir example
seb@sebtop ~/oopp $ cd example/
seb@sebtop ~/oopp/example $ git init
Initialized empty Git repository in /Users/seb/oopp/example/.git/
seb@sebtop ~/oopp/example $ ls
total 0
drwxr-xr-x 3 seb staff 96 Feb 8 11:50 .
drwxr-xr-x 4 seb staff 128 Feb 8 11:49 ..
drwxr-xr-x 9 seb staff 288 Feb 8 11:50 .git
seb@sebtop ~/oopp/example $ git status
On branch master
No commits yet
nothing to commit (create/copy files and use "git add" to track)
seb@sebtop ~/oopp/example $ touch aaa.txt
seb@sebtop ~/oopp/example $ git status
On branch master
No commits yet
Untracked files:
(use "git add <file>..." to include in what will be committed)
aaa.txt
nothing added to commit but untracked files present (use "git add" to track)
seb@sebtop ~/oopp/example $


oopp
Name
Date Modified
Size
Kind
example
Today at 11:50 AM
--
Folder
aaa.txt
Today at 11:50 AM
Zero bytes
Plain Text
Macintosh HD › Users › seb › oopp
2 items, 58.96 GB available

```
seb@sebtop ~/oopp/example $ git add aaa.txt
seb@sebtop ~/oopp/example $ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   aaa.txt

seb@sebtop ~/oopp/example $
```



You don’t add files, you add changes to a file.

```
seb@sebtop ~/oopp/example $ git add -A
seb@sebtop ~/oopp/example $ git staus
git: 'staus' is not a git command. See 'git --help'.

The most similar command is
        status
seb@sebtop ~/oopp/example $ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   aaa.txt
        new file:   bbb.txt

seb@sebtop ~/oopp/example $
```

Commits to a set of changes (version):

```
seb@sebtop ~/oopp/example $ git commit -m "first files"
[master (root-commit) 211144f] first files
 2 files changed, 4 insertions(+)
 create mode 100644 aaa.txt
 create mode 100644 bbb.txt
```

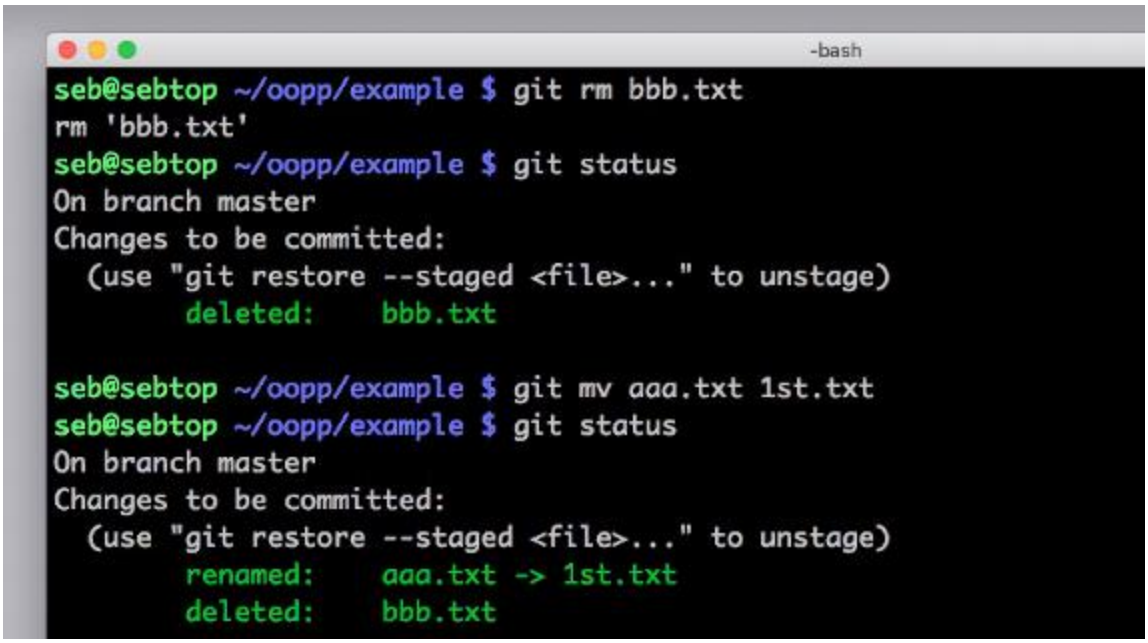
```
seb@sebtop ~/oopp/example $ git log
commit 211144faf1b794c3fbfe1994d699a523415892bd (HEAD -> master)
Author: Sebastian Proksch <development@mail.proks.ch>
Date:   Mon Feb 8 11:55:41 2021 +0100

    first files
```

Log -n where n is the nth last change

```
seb@sebtop ~/oopp/example $ git log
commit 4c4dfaea7ac24415871bf4e0107ac037cc281dce (HEAD -> master)
Author: Sebastian Proksch <development@mail.proks.ch>
Date:   Mon Feb 8 11:57:53 2021 +0100

    5

commit 4da95282bd20694988dfb17d6b2a124a022a8285
Author: Sebastian Proksch <development@mail.proks.ch>
Date:   Mon Feb 8 11:57:43 2021 +0100

    4

commit 17fdad10a4a8986ee6e2f5eaf637eb2fba6bbab5
Author: Sebastian Proksch <development@mail.proks.ch>
Date:   Mon Feb 8 11:57:31 2021 +0100

    3rd

commit 211144faf1b794c3fbfe1994d699a523415892bd
Author: Sebastian Proksch <development@mail.proks.ch>
Date:   Mon Feb 8 11:55:41 2021 +0100

    first files
```

```
seb@sebtop ~/oopp/example $ git log --pretty=oneline
4c4dfaea7ac24415871bf4e0107ac037cc281dce (HEAD -> master) 5
4da95282bd20694988dfb17d6b2a124a022a8285 4
17fdad10a4a8986ee6e2f5eaf637eb2fba6bbab5 3rd
211144faf1b794c3fbfe1994d699a523415892bd first files
```

--diff shows changes



Git recognizes that is a rename of a file and not a new file.

Commit Ids (SHA)

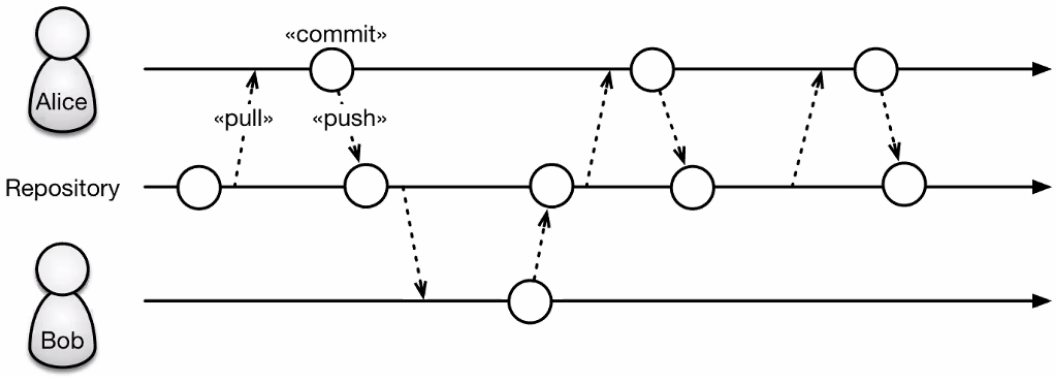
- Committer
- Author
- Dates (Author+Commit)
- Commit Message
- Patch (Content)
- Parents Commit Ids

Main Branch

- You are always working on a branch
- This branch is typically called “master” or “main”

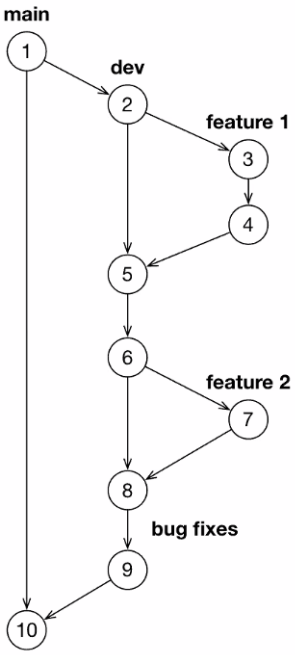
Trunk-based development

- Everybody works on the main branch
- This is how history works in Dropbox, Google docs, ... !



Feature Branches

- Developers develop in isolation and commit to branches
- Avoids that partial features break the system
- Merge can trigger event, e.g., a code review
- Core idea for Pull Requests (will be discussed later)
- A branch only points to a commit
- New commits on a branch move the pointer, it always points to the most recent commit




```
-bash
seb@sebtop ~/oopp/example $ git branch
* master
seb@sebtop ~/oopp/example $ git branch -c fb1
```

```
seb@sebtop ~/oopp/example $ git branch
  fb1
* master
```

Rename with:

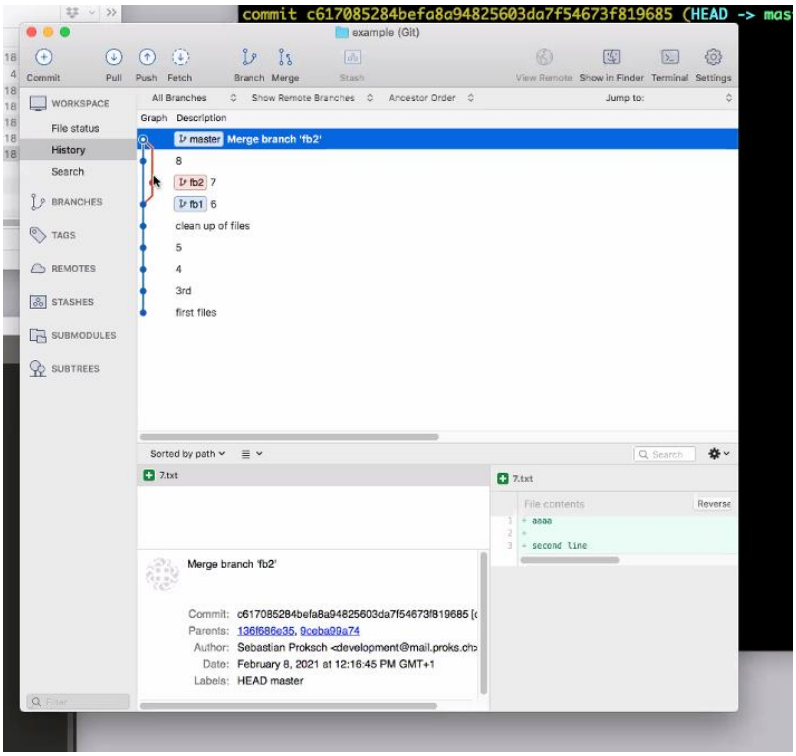
```
seb@sebtop ~/oopp/example $ git branch -m fb1 fb_1
```

```
seb@sebtop ~/oopp/example $ git branch -d fb_1
Deleted branch fb_1 (was 9d49f39).
```

```
seb@sebtop ~/oopp/example $ git checkout fb1
Switched to branch 'fb1'
```

```
seb@sebtop ~/oopp/example $ git merge fb1
Updating 9d49f39..5548ef6
Fast-forward
 6th.txt | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 6th.txt
```

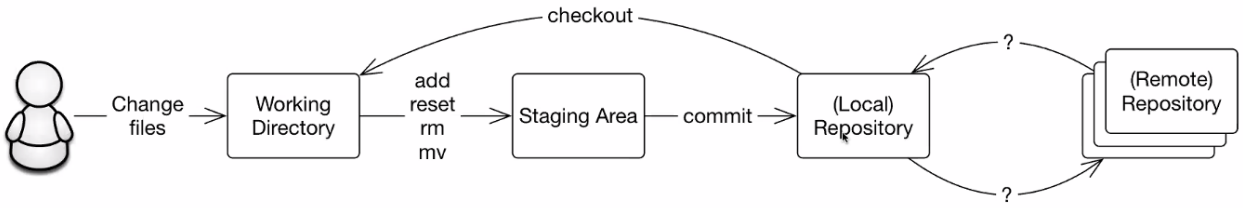
(fast forward merge)



Concurrent changes merge

Git is a **Distributed** Version Control System

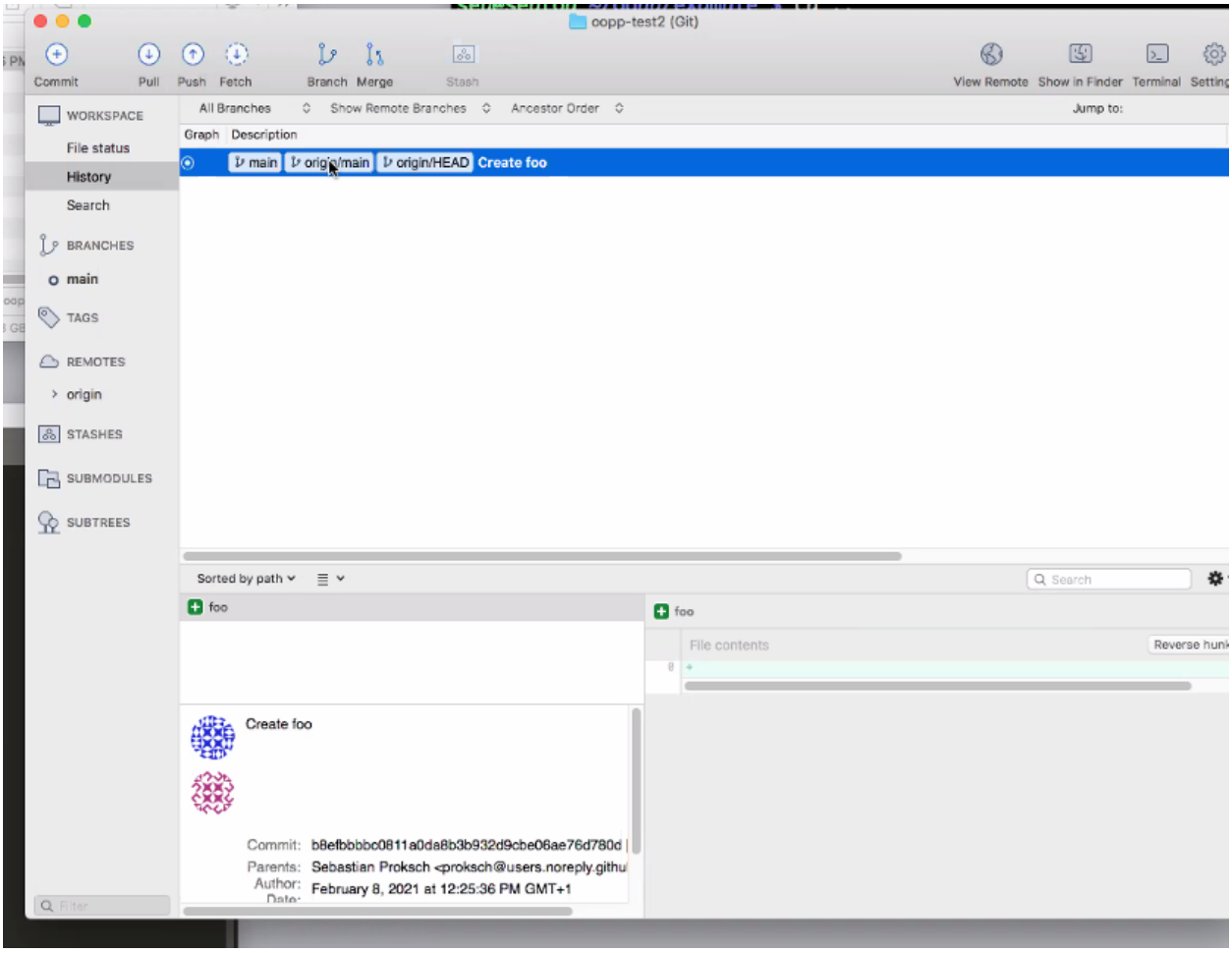
- Git is distributed, you can add other remote repositories
- Number of remotes is unlimited
- Remotes have a name, default is “origin”



Remotes

- git remote add <name> <url> # create new remote, url can be a dir!
- git remove rename <old> <new> # rename a remote
- git remote remove <name> # delete a remote

```
seb@sebtop ~/oopp $ git clone git@github.com:proksch/oopp-test2.git
Cloning into 'oopp-test2'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```



Origin/main is the branch name of remote

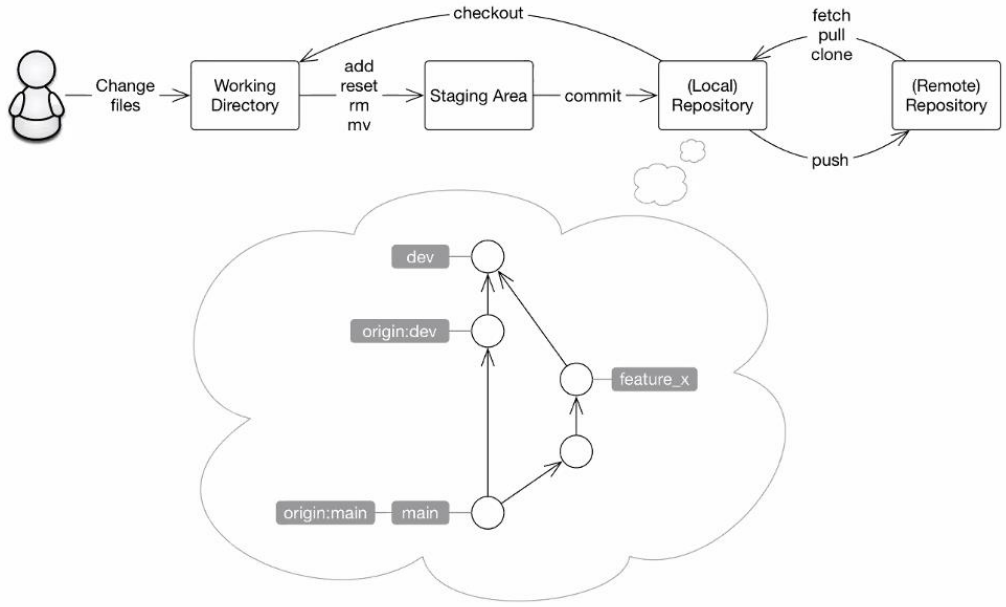
```
seb@sebtop ~/oopp/oopp-test2 $ git branch
* main
seb@sebtop ~/oopp/oopp-test2 $ touch bar
seb@sebtop ~/oopp/oopp-test2 $ git add bar
seb@sebtop ~/oopp/oopp-test2 $ git commit -m "added bar"
[main 096949d] added bar
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 bar
seb@sebtop ~/oopp/oopp-test2 $ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 272 bytes | 272.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:proksch/oopp-test2.git
```

Git push synchronizes local changes to the remote tree.

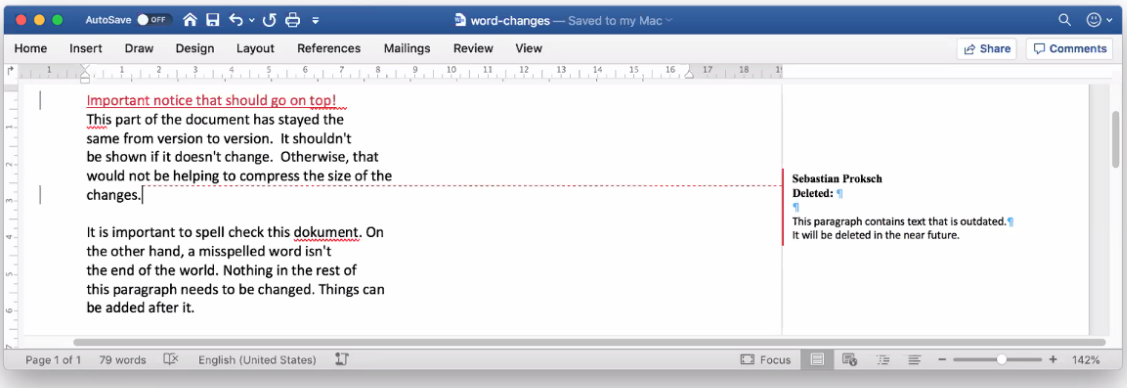
Git pull is fetching + merging

Git fetch just synchronizes the changes and puts them into the local tree.

Git, the Full Picture



Someone edited your .docx... what changed?



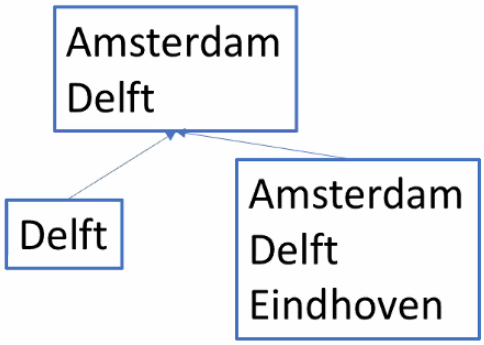
What happens if your forgot to enable review mode?

Change tracking of in *binary* files needs to be supported by specific applications. Use plain text files wherever you can!

40

Git to the rescue: Three-way Merge

- It becomes easier if you know the base version!



git diff – show changes

- git diff # all changes in current WD
- git diff <commit1> <commit2> # between two commits
- git diff <commit> <path> # between commit and file
- ...

Conflicts

- Example 1: Edit a file in a branch and delete it on master
- Example 2: Edit a line in a branch and delete this line on master
- In both cases, try to merge the branch

Outcome:

- Cannot be decided by Git, the developer needs to fix this
- Afterwards, add the file(s) again and commit

If there is a conflict the merge stops and git prompts you to review it manually.

Recommended Reading

- Read Git Man-Pages
- Git Book
  https://git-scm.com/book/en/v2
- GitHub Git Cheat Sheet
  https://education.github.com/git-cheat-sheet-education.pdf

Lecture 3. Human Computer Interaction

History

- Usability problems first researched in a Danish computer magazine.
- Apple shared a Human Interface Guidelines around the same time.
- 30 usability problems identified as a result.
- 37% of those problems are identified by the average participant.
- The best participant could only spot 60% of usability problems.
- **Conclusion: people are bad at spotting (all) usability problems.**

Evaluators

- Nielsen observed that "bad/amateur participants" are still able to spot rare usability problems.
    - Therefore decides to measure aggregated usability problems found by group size.

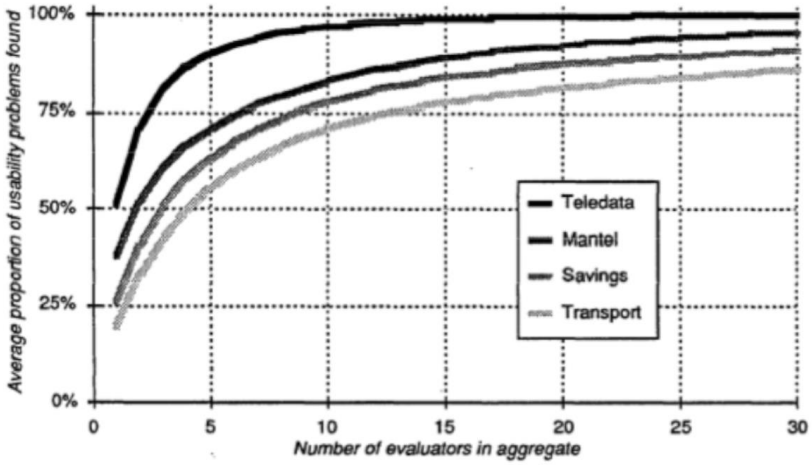


Figure 4. *Proportion of usability problems found by aggregates of size 1 to 30.*

$$f(i) = (1 - (1 - \lambda)^i)$$

$f(i)$ % problems found

λ % found by single evaluator

i Number of evaluators

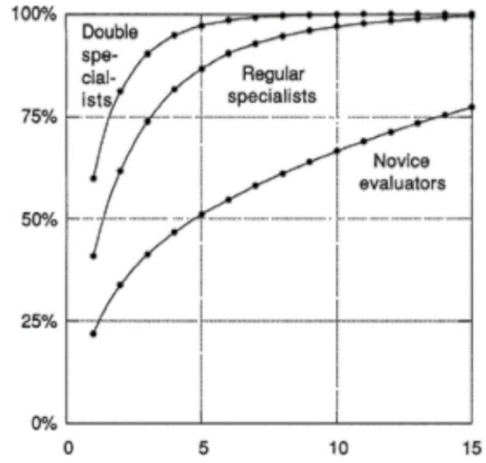


Figure 2 *Average proportion of usability problems found as a function of number of evaluators in a group performing the heuristic evaluation.*

- The graph shows diminishing returns. Given by the formula:
- **We can now calculate the exact number of evaluators needed to identify the desired percentage of problems.**
- It also shows how many of them **depending their expertise**: Novice, Expert and "Double" experts have different lambda values (22, 45, 60 respectively).

Reporting

- **Huge variance in agreeing of problem identification (5%-45%) until researches formalized the reporting process**:
    - **Problem description**: a brief description of the problem
    - **Likely/actual difficulties**: the anticipated difficulties that the user will encounter as a consequence of the problem
    - **Specific contexts**: the specific context in which the problem may occur
    - **Assumed causes**: description of the causes of the problem

Confusion matrix

		Actual problem: Yes	Actual problem: No
Reported as problem	Yes	True positive	False Positive
	No	False Negative	True Negative

-
  - Not clear whether it is specifically a ***usability*** problem

		User test problem: Yes	User test problem: No
Heuristic evaluation problem	Yes	True positive	False Positive
	No	False Negative	True Negative

-
  - Heuristic: “any approach to problem solving or self-discovery that employs a practical method that is not guaranteed to be optimal, perfect, or rational, but is nevertheless sufficient for reaching an immediate, short-term goal or approximation.”
  - “Heuristics can be mental shortcuts that ease the cognitive load of making a decision”
- The goal of the study is to minimize false positives and false negatives

$$Precision = \frac{TP}{TP + FN}$$

- Standardization of the study doubled the precision.
- **Take away: <u>Whatever we do to measure something, standardize it</u>!**

Nielsen Heuristics

Revised

1. Visibility of system status
2. Match between system and the real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, and recover from errors
10. Help and documentation

An internal survey with their old data addressed from 1-5 to what extent does the identified usability problem matches a list of old heuristic. After a factor analysis (correlations between ratings) reduced the problems to the first revised 7, and added 8, 9 because they overlap a lot of problem. Number 10 was arbitrarily added (not reported why).

Source guidelines

Feedback: keep user informed about what foes on
Provide status information
Feedback: show that input has been received
Features changes as user carries out task
Feedback provided for all actions
Feedback timely and accurate
Indicate progress in task performance
Direct manipulation: visible objects, visible results
Identity cues system response's vs user's goals
Show icons and other visual indicators
WYSIWYFG: do not hide features
What incorrect inferences are most likely

Visibility of system status

"The system should keep users informed about what is going on, with appropriate feedback in good time."

Source guidelines

Speak the user's language
Contains familiar terms and natural language
Speak the user's language
Metaphors from the real world
Familiar user's conceptual model
Use of user's background knowledge
Learnable through natural, conceptual model
Follow real-world conventions
Screen representation matches non-computer
Encourage user to import pre-existing tasks
Identity cues between actions and user's goals
Understand the user's language

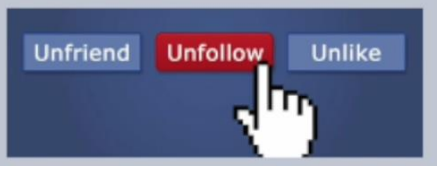
Match between the system and the real world

"The system should speak the users language, with words, phrases and concepts familiar at the user (not technical terms)."

Source guidelines
Undo and redo should be supported
Obvious way to undo actions
Forgiveness: make actions reversible
Ability to undo prior commands
Clearly mark exits
Ability to re-order or cancel tasks
Modeless interaction
User control: allow user to initiate/control action
Modelessness: allow user to do what they want

User control and freedom

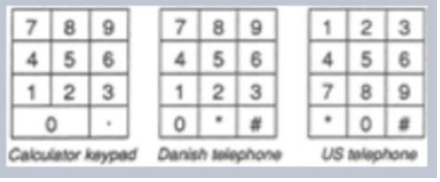
"System functions may be chosen by the user by mistake; they need a clearly marked ' emergency exit ' without having to go through an extended dialogue."



Source guidelines
Consistency: express same thing same way
Consistency
Consistency: same things look the same
Uniform command syntax
Conform to platform interface conventions
Consistent key definitions throughout
Universal commands: a few, generic commands
Show similar info at same place on each screen

Consistency and Standards

"Users should not have to wonder whether different words, situations or actions mean the same thing"



Source guidelines
Prevent errors from occurring in the first place
System designed to prevent errors
Understand the user's language
What planning mistakes are most likely?
What slips are most likely?
Identity cues between actions and user's goals

en, J. (1994, April). Usability inspection methods. In *Conference companion on Human rs in computing systems* (pp. 413-414).

Error prevention

"Even better than a good error message is a careful design which prevents a problem from occurring in the first place."

Door open, microwave off

Source guidelines
See-and-point instead of remember-and-type
Make the repertoire of available actions salient
Seeing and pointing: objects and actions visible
All user needs accessible through the GUI
What features often missed and at what cost?
Provide lists of choices and picking from lists
Minimise the users' memory load
Direct manipulation: visible objects, visible results
Easy or difficult to perform (execute) tasks?
Evoke goals in the user
Allow access to operations from other apps
Clearly marked exits
Show icons and other visual indicators
Integrate with the rest of the desktop

Recognition rather that than recall

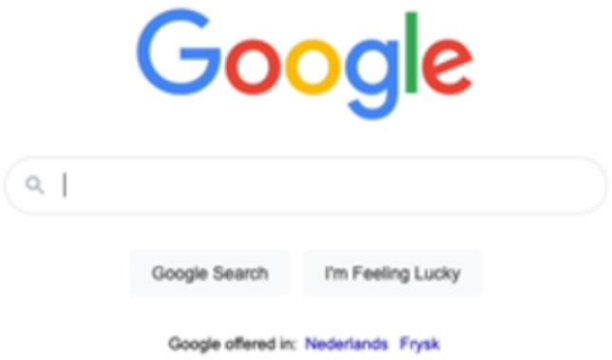
"Try to make objects, actions and options visible. The user should not have to mentally hold information and transfer it to another part of the interface. Instructions for using the system should be visible or easily retrievable whenever appropriate."

Source guidelines
Accelerators should be provided
Shortcuts: Accelerators to speed up dialogue
User tailorability to speed up frequent action
User control: allow user to initiate/control actions
System should be efficient to use
User interface should be customizable
Ability to re-order or cancel tasks
Keyboard core functions should be supported
Physical interaction with system feels natural

sen, J. (1994, April). Usability inspection methods. In *Conference companion on Human*

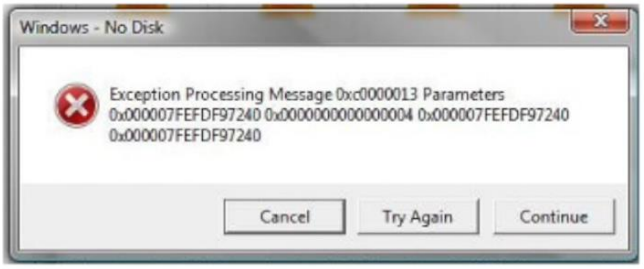
Flexibility and efficiency of use

"Accelerators (such as a function keys or macros) and senior buyer of the novice user may often speed up the interaction of for the expert user. Thus the system that can cater to both inexperienced and experienced users."



Aesthetic and minimalist design

"Dialogues or other interface items should not contain information that is irrelevant or rarely needed. All information on the screen competes with relevant units of information, diminishing at their relative visibility."



Help users recognize, diagnose, and recover from errors

"Helping users to recognise, diagnose and recover from errors. Error messages should be clearly expressed in plain language, precisely indicate the problem, and constructively suggest a solution."

Help and documentation

"Although the system should be able to be used without documentation, it may be necessary to provide help of some form. This information should be easy to search, focused on it the user's ongoing task (context sensitive), list the steps to carried out, and be brief and to the point."

Heuristic Evaluation/procedure

'Having a small set of evaluators examine the interface and judge its compliance with recognized usability principles (the "heuristics")' (Nielsen, 1994, p. 155)

1. Identify participants
2. Explain them the heuristics
3. Ask them to identify problems related to each of the heuristics
4. Let participants fill the survey individually (to avoid group bias) (user Qualtrics/google form)
    a. Also track the self-reported expertise of the user

Procedure

hell no...

5–15 min

- **Evaluators do their inspection of interface alone (independent and unbiased results)**
- Found problems are written down
- Evaluation session last 1 to 2 hours, depending on the complexity of the system
- During the interaction evaluator goes through the interface several times and inspect each dialogue component with heuristic principles
- Evaluation of actual system or paper prototype is possible
- For complex system domain-specific systems, evaluator might need support from domain assistant.

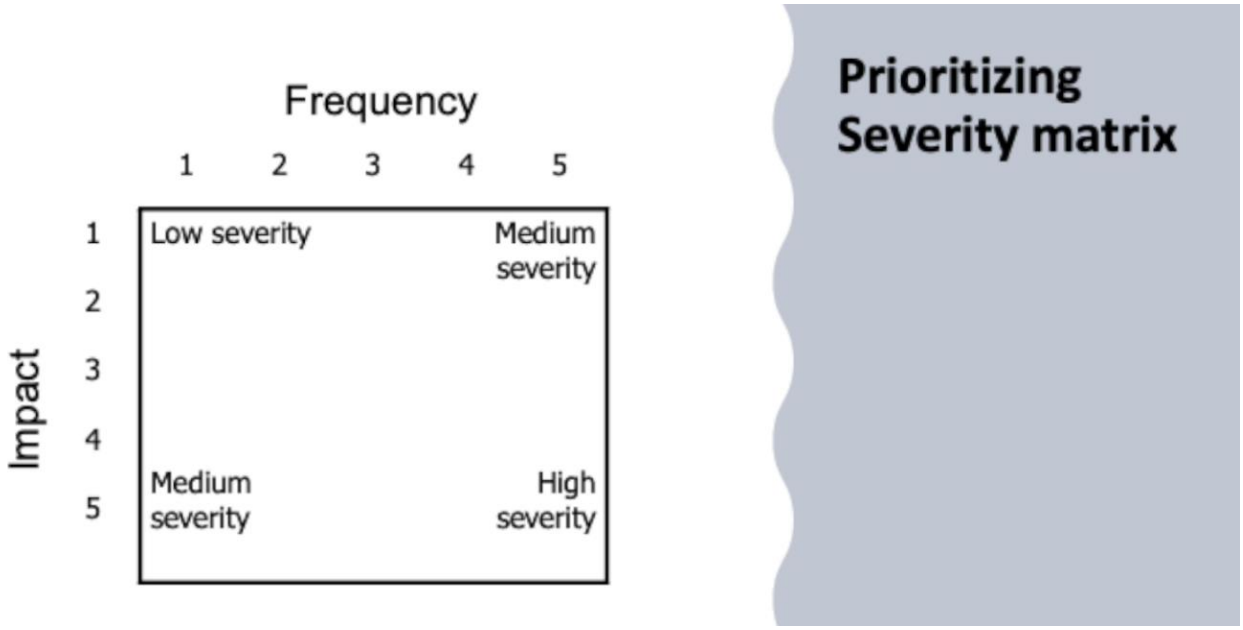
1. Nobody is willing to spend 1 hour of their time to do an experiment (and that time generally regards bigger projects i.e. Spotify, Facebook, etc) for a school project.
2. COVID! -> We will send an online a survey
3. (Optional) We can put 1-4 euro each and offer a bol.com/amazon discount voucher prize for the participants that identifies the most problem->heuristic relations. That way we can ensure enough participants and serious responses.
    a. To avoid people repeating the survey we will require them to add their email address if they want to participate in the price.
    b. As a sanity check also track the IP address to invalidate extra submissions.

Material

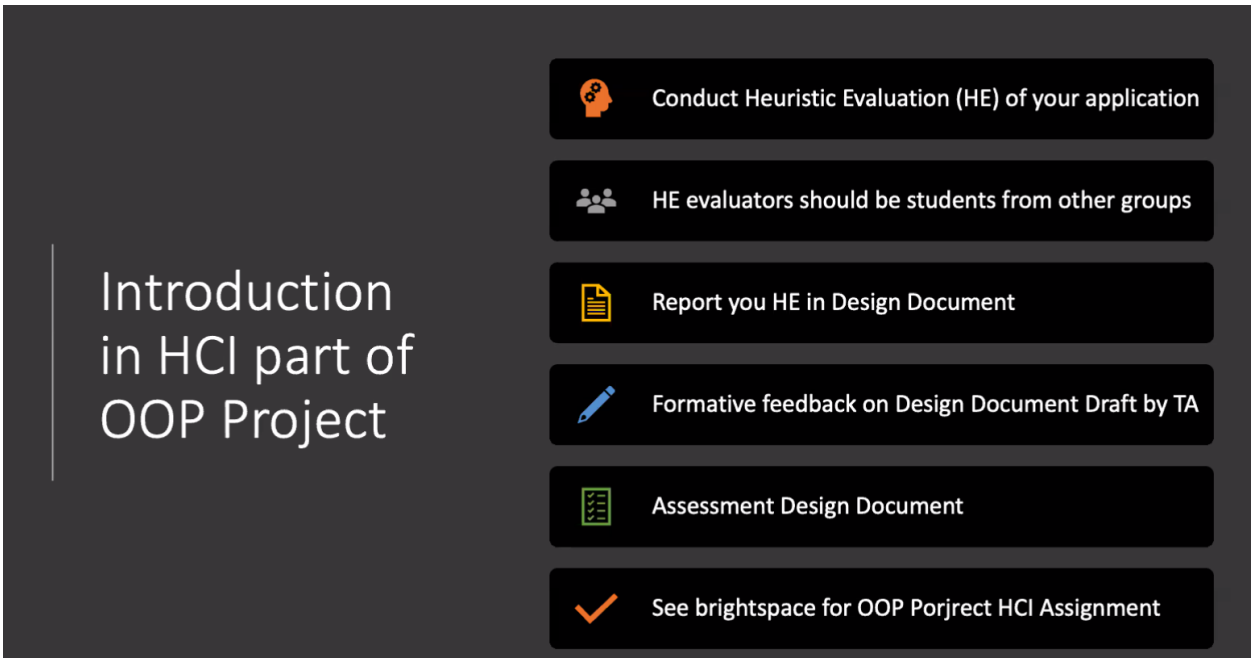
LIST OF HEURISTICS

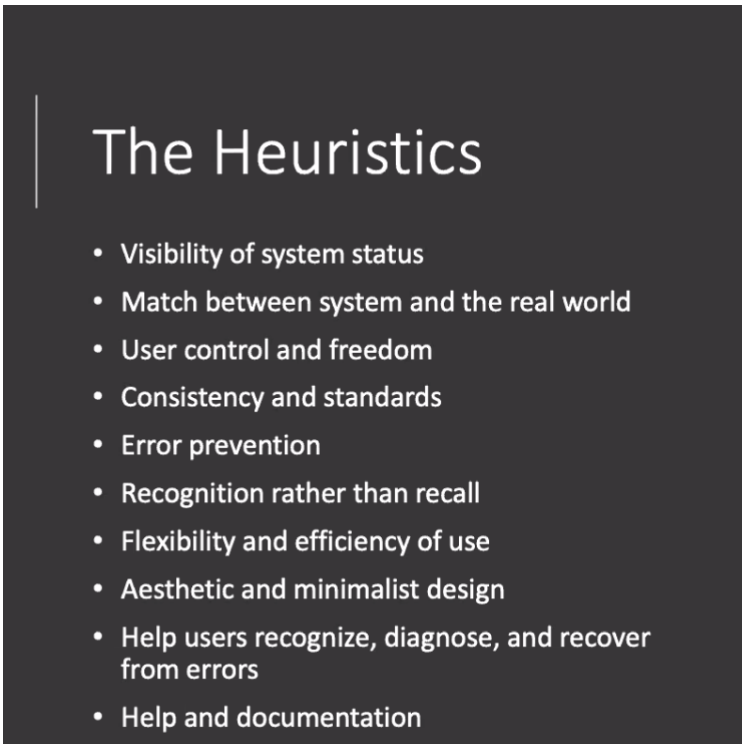
THE SYSTEM OR PROTOTYPE

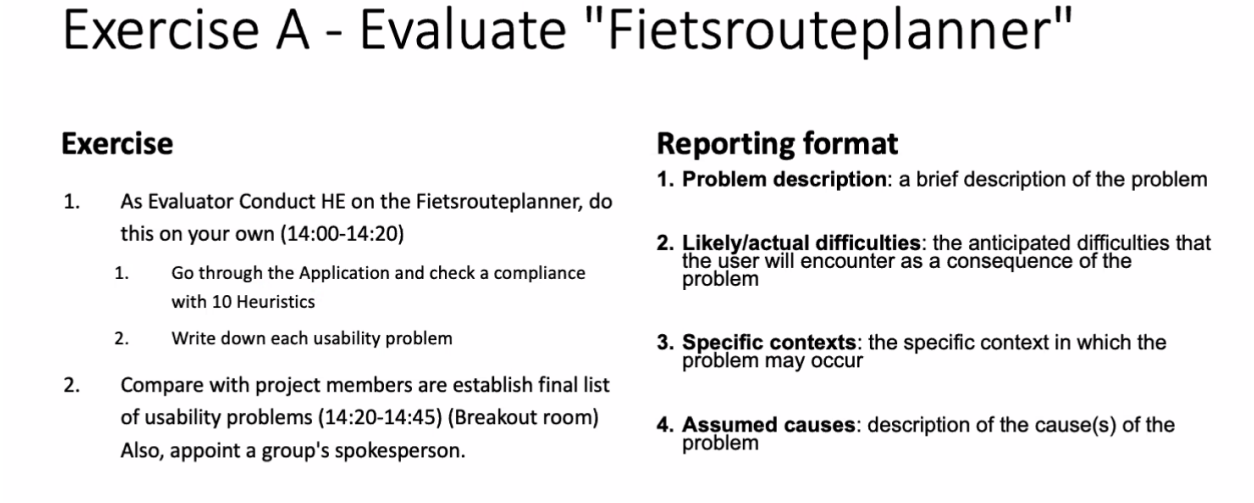
FORM TO REPORT PROBLEMS



HCI Live session





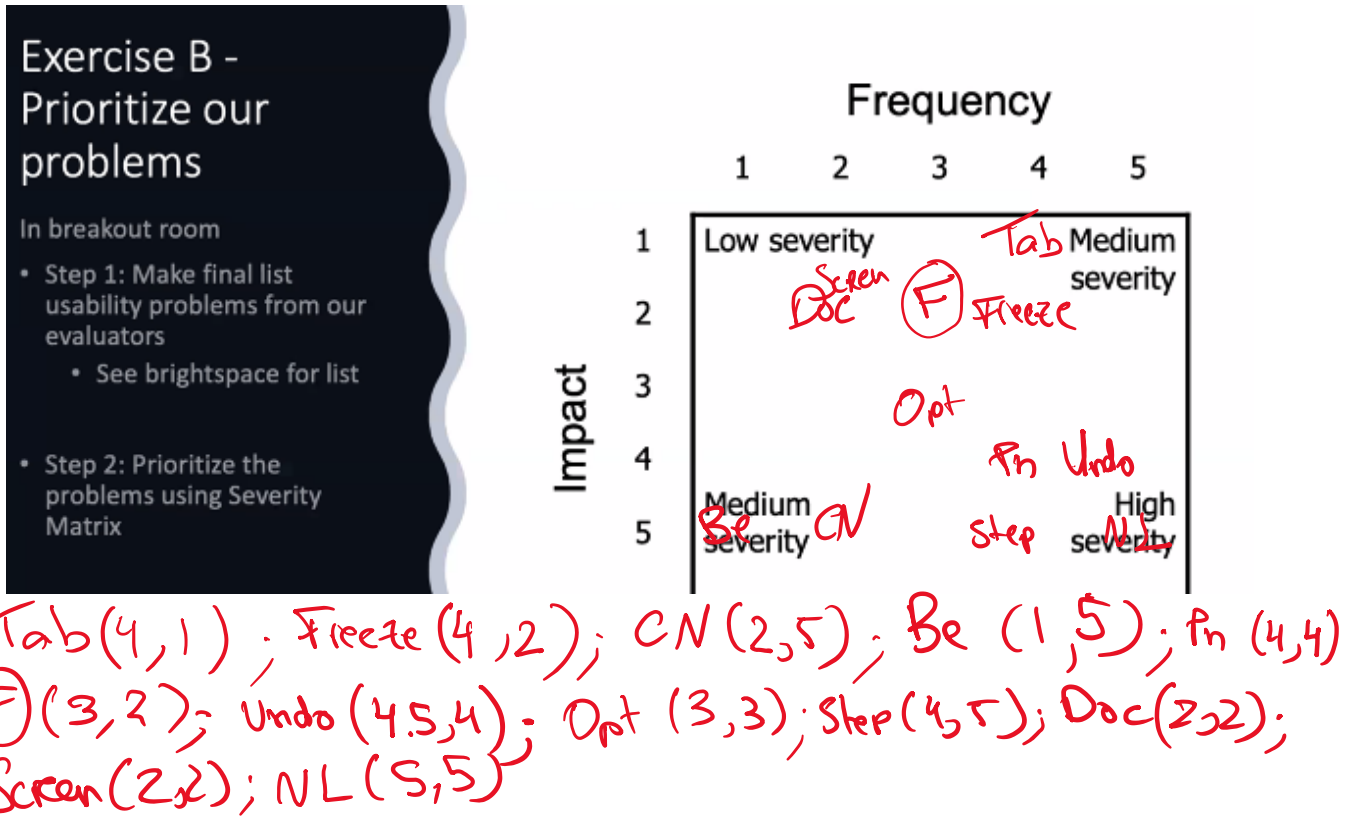


Problem:

1. Hard to edit a trip (same for adding in between locations)
2. User frustration from having to re-enter previous location(s)
3. Consistency
4. Not implemented

Utrecht automatic zoom

1. Start -> Typing: Start typing. Suggestions is out of the box (Aesthetics)
2. English: Dutch stuff written in Dutch.
3. Show Dutch discount ad for Utrecht
4. Postal code address doesn't work well.
5. Town (Plaats)
6. Loading screen unnecessary -> Aeshtetics
7. English problem: Turns are still in Dutch. -> Aestheics.
8. Pop up aesthetics.
9. Aeshtetics. Button. Good.
10. User control -> share button. Bad
11. User control. -> Input validation zoom map.
12. Help: check
13. Design/recall: (like google maps) check
14. User control -> can't customize the route.



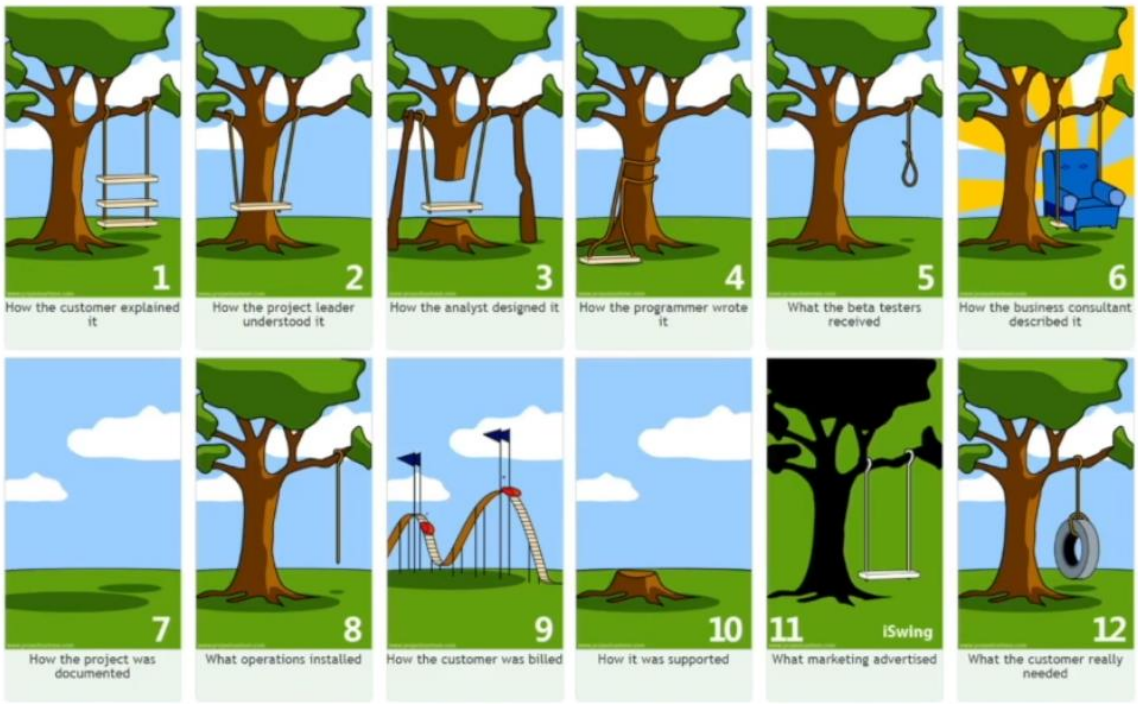
1. Suggestions" tab appears even when no suggestion is available. - #5 Error prevention
2. Interface freezes when "processing input" even if the user hasn't fully finished inputting. - #1 Visibility of system status
3. Novice users can't figure out what Combined networks, LF, gritting route means. - #2 Match between system and the real world
4. Path going through Belgium do not cover driving directions. - #2 Match between system and the real world
5. Slider bars do not move continuously when on touchscreen. - #1 Visibility of system status

6. The website only provides options to print or share the route. Offering less flexibility. - #7 Flexibility and efficiency of use
7. User can't change the route planner once results are displayed. No undo button! - #3 User control and freedom
8. User must navigate through all the options in order to compare routes. - #6 Recognition rather than recall
9. Users can't keep track of what step they are on. - #1 Visibility of system status
10. Weak structure of help and documentation, especially lack of "search" button. - #10 Help and Documentation
11. When entering addresses to plan a route the feedback is a bit too long, since it interferes with the design/interface in an annoying way. - #8 Aesthetic and minimalist design
12. When the language setting in English (or any other languages), but the directions are still in Dutch. - #4 Consistency and standards

Lecture 4. Requirements Engineering

SOFTWARE PROCESSES AND REQUIREMENTS ENGINEERING

- Requirements Engineering Process
  - 🔎 Elicitation
  - ✏️ Specification
  - ✅ Validation
  - 🤔 Change
- Software Development Process
  - Waterfall
  - Scrum



- It is vital to list, specify, **validate** and share the requirements in a clear and structured way
    - Even when client are not able to explain what they want

REQUIREMENTS

DESCRIPTIONS OF THE SERVICE(S) THAT A SYSTEM SHOULD PROVIDE AND CONSTRAINTS ON ITS OPERATION

Ian Sommerville in "Software Engineering" (10th edition)

- What a system should do (goal)
- What can it do (features)
- Under which constraints

User vs system requirements

User requirements
Statements in a natural language (+ diagrams)

System requirements
More detailed descriptions of the software system's functions, services and operational constraints.

- User requirements must be jargon free and easily understood by the client.
    - A user requirement can be branched into multiple system requirements
- System requirements are much more specific and regard the technical implementation

Example:

User requirement

- The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements

- On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
- The system shall generate the report for printing after 17:30 on the last working day of the month.
- If drugs are available in different dose units (e.g. 10mg, 20mg, etc.) separate reports shall be created for each dose unit.

Functional vs non-functional requirements

- **Functional requirements**: what services the system should provide.
  - Given an input: what should system (not) do?

- **Non-functional requirements**: constraints on the services offered by the system.
  - For example: timing constraints (performance), process constraints, etc.
  - But also, usability constraints (think of HCI)

Uptime, number of requests per minute, are other examples.

- Examples of **functional requirements**
  - The user shall be able to search for appointments in its calendar.
  - Every day the system shall generate a list containing the patients that are expected for an appointment that day.
  - Each student shall be uniquely identified by their seven-digit number.

"The system shall provide appropriate viewers for the user to read documents in the document store"

What does "appropriate" mean?

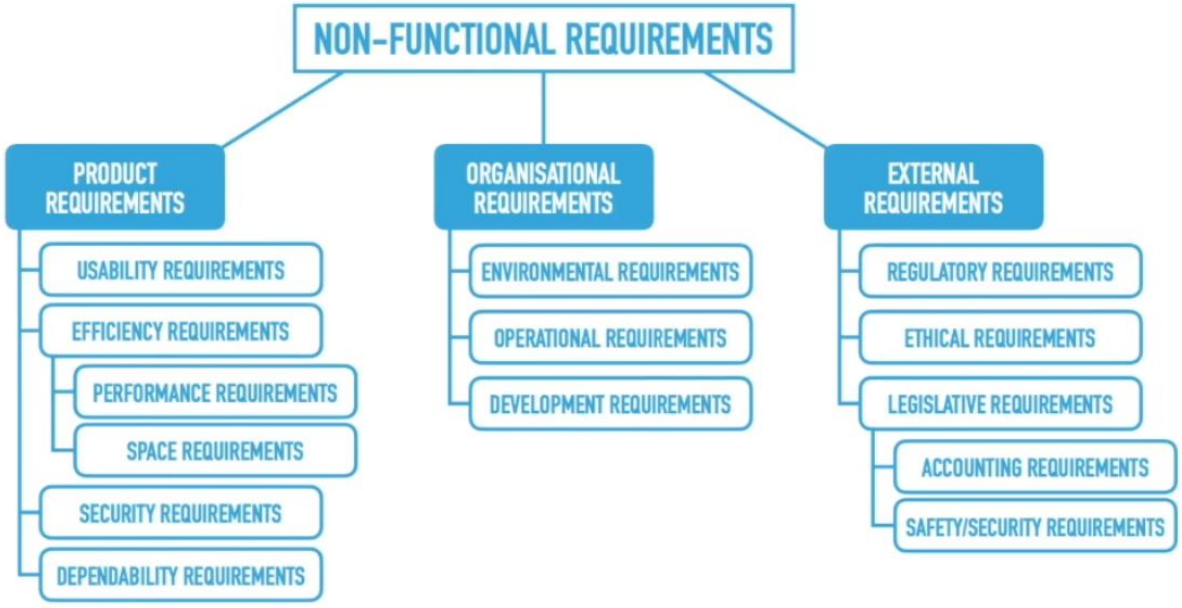
Requirements should be **complete** and **specific**.

- Complete: all services required by the user are specified
- Consistent: requirements do not contain contradictory definitions

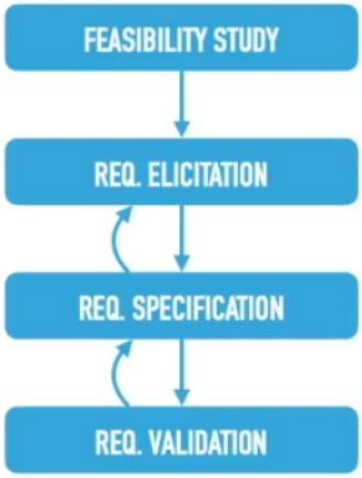
For larger, complex systems it's practically impossible to achieve both.

NON-FUNCTIONAL REQUIREMENTS

- Three main categories
  - **Product**
    System must behave in a particular way (e.g. execution speed, reliability, ...)
  - **Organisational**
    Consequence of policies and procedures (e.g. process standards used, ...)
  - **External**
    Arise from factors external to the system and its development process (e.g. interoperability requirements, legal requirements, ...)



Listing requirements

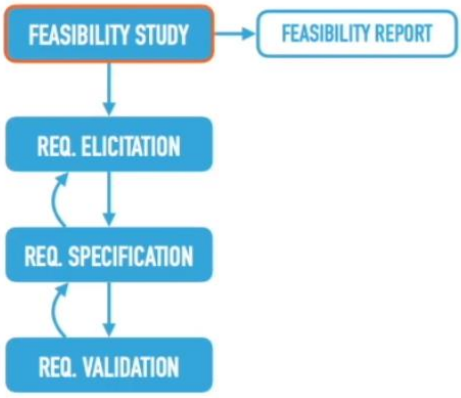


1. Feasibility :Assess whether the client needs are realistic, solvable and/or within budget.
2. Elicitation: Gather all requirements to get initial understanding of the problem and the "problem domain".
3. Specification: Document the problem illustrated in step 2 in a way that a) the client can verify that you understood the problem and b) developers know what to build precisely.
4. Validate: Agree with the client that the documented requirements compromise the entirety of the project enforceable deliverables.

In practice these steps are not necessarily a waterfall process.

FEASIBILITY STUDY

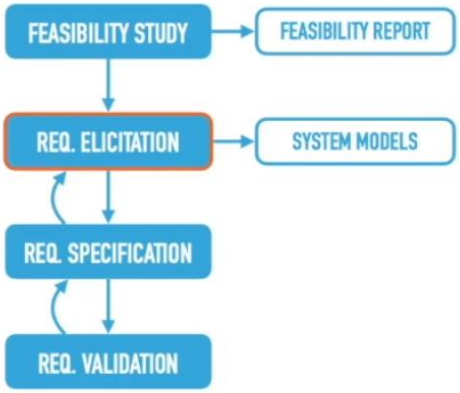
- Optional, before detailed analysis starts
- Short term, relatively cheap, studies
- Assess whether there's
  - a need and/or market for the software
  - technically and/or financially realistic to develop
    - *experiment/make prototypes*
- Requirements Engineering Process in practice;
  - No sequential process, but iterative
  - Activities are commonly interleaved



This report is not part of the CSE110 assignments.

REQUIREMENTS ELICITATION AND ANALYSIS

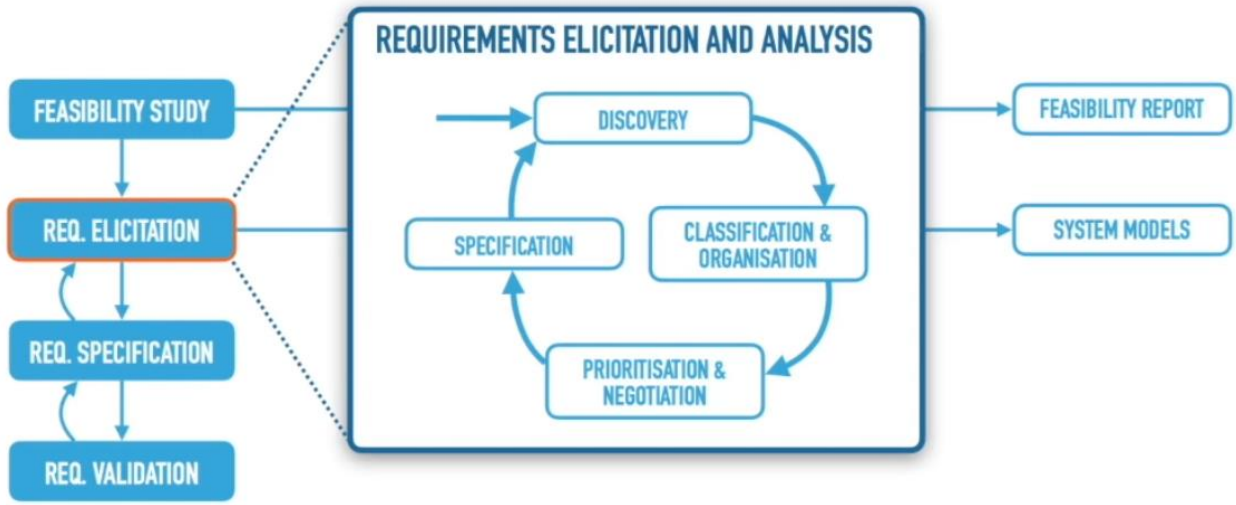
- Deriving **system requirements** through
  - Observing existing systems
  - Discussions with potential users
  - Task analysis
  - ...
- May involve the development of
  - System models
  - Prototypes



Prototypes may be wireframes and sketches (like for the web-socket app for *CSE1500*.

- Meet with client / end-users to find out;
  - Application domain
  - Services that need to be provided
  - Performance requirements
  - Hardware/software contraints (e.g. Windows,...)
  - ...
- Involves interaction with all system stakeholders: end users, business managers, etc.
- Interview the stakeholders
  - Closed interview: fixed list of questions
  - Open interview: no pre-defined agenda
  - Practice: mix of both
- Ethnography
  - Observing current systems/procedures

The focus of CSE1105 project is on end-users (among all the other users/stakeholders, the people using the product itself). But to get a main picture of the "domain" of the problem it is advised to talk to all the stakeholders of the project.



- Discovery: process of interacting with the stakeholder to identify the requirements. (i.e via interviews).
- Classification & organization: organize all the discovered requirements into different categories (user, system, functional, non-functional).
- Prioritization & negotiation: based on budget constraints. Identify conflicting requirements and reach a solution agreed with the stakeholders.
- Specification: Update documentation of the requirements

Must have: Without 1 of them the project is considered a failure. Implementing all must have results in a minimum viable product.

Should have: Additional value. Implement if resources allow it.
Could have: Additional value that has less impact.
Wont have: Discarded. Ignore even if budget is available.

The requirement elicitation can be documented via use-cases (also called stories/scenarios). In natural L.

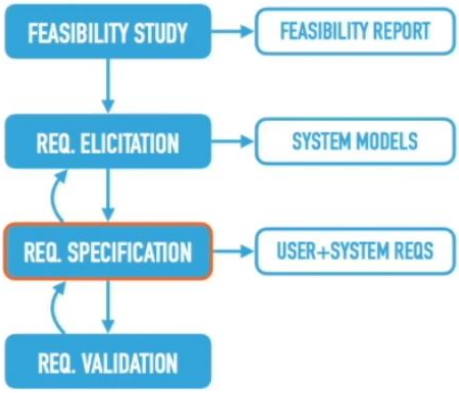
- Stories / scenarios
  - Describe how a situation is currently handled

- A structured scenario includes;
  - What the system/users expect at the start
  - Normal flow of events
  - Potential errors, and how to deal with them
  - Potential concurrent activities
  - System state at the end of scenario

REQUIREMENTS SPECIFICATION

- Translate information from previous phase into understandable document (for client).

- Describe **what** to deliver in a structured way
  - Decide on a format and use for all requirements
  - Use "shall" for mandatory req's, "should" for desirable
  - Text highlighting (bold/italic/underline) for key elements
  - Avoid use of jargon, abbreviations, acronyms
  - Preferably include a rationale (explain why, and by whom)



- Use the same format for all requirements
- Also include consequences from removing a requirement

Example

- The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes
  (Changes are relatively slow, so more frequent is unnecessary, […])

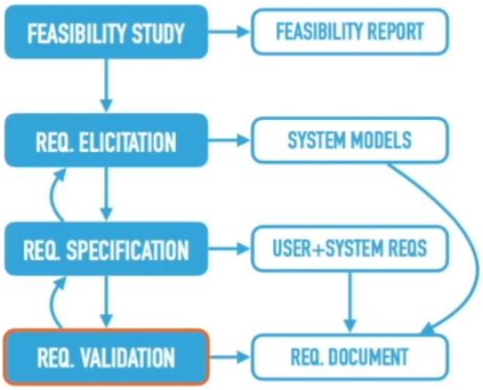
USER STORIES

- More common in Agile methodologies to describe "requirements"
- Focus on **stakeholders** and the **value** that is brought to them
  - Not all stakeholders are obvious: you'll learn about this in lecture 5
- Accompanied by
  - Acceptance criteria / "Definition of Done"
  - Failure cases
- Less suitable for non-functional requirements

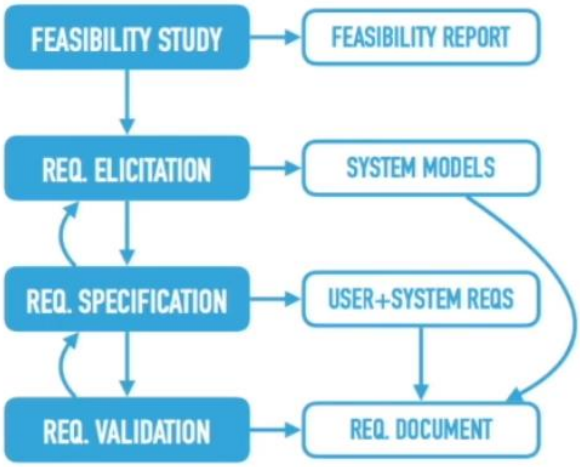
AS A <ROLE>, I WANT <DESIRED OUTCOME> SO THAT <REASON>

REQUIREMENTS VALIDATION

- Reach an **agreement** with the client upon the nature of the problem.
- Check requirements for;
  - Validity
  - Realism
  - Consistency
  - Completeness
  - Verifiability
- Repair errors that are found in this phase



- Make sure that requirements are measurable and testable
- After validation the backlog can we used as a contract to justify decisions



The requirements document is an organized version of the (user + system) requirements, with appendix of the system model (wireframe) and feasibility report (already done by the course) with a bit of legal jargon in which there could be clause where all parties agree and make the resulting document legally binding, which translates into the official backlog.

COMMON PROBLEMS

- Stakeholders don't know what they really want
- Stakeholders express requirements in their own terms
- Different stakeholders may have conflicting requirements
- Organizational and political factors may influence the system requirements
- New stakeholders may emerge

HOW DOES THIS TRANSLATE TO THE OOP PROJECT?

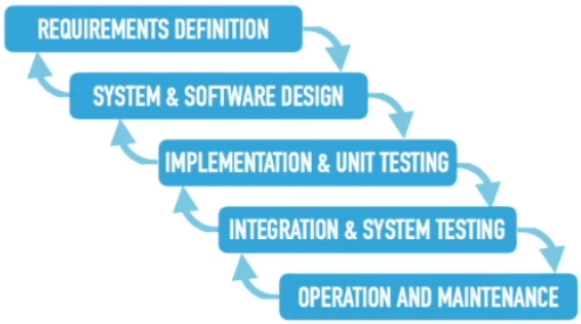
- OOP Project is a small project, so we do not need the entire (formal) process.
- For example, for government projects you'll typically need the full process + all deliverables.

- User story -> requirement -> feature -> gitlab issue -> git branch

Software development process

WATERFALL MODEL (ROYCE, 1970)

- Minimising risks by working **systematically** and **thorough**
- Consequences
  - Hard to elicit all req's from the beginning
    - req's change often
    - users don't know exactly "what they need"
  - In practice, implementation starts late
  - Client only sees a real ROI in the last phase
    - Cost of change is high then



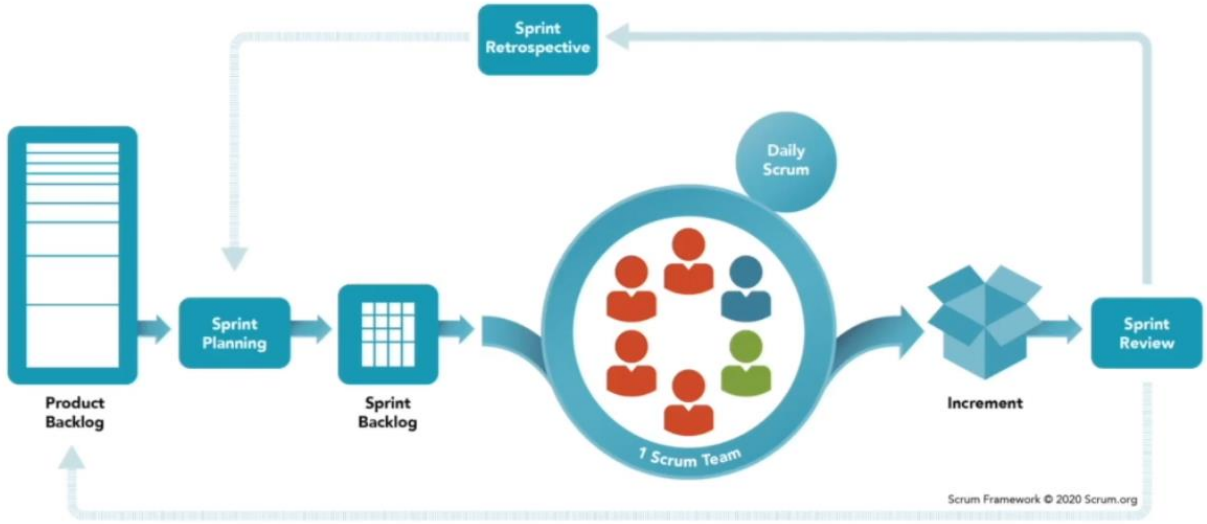
- Useful for projects that require strict planning/have tight budgets
  - For example: governmental projects

Scrum

INCREMENTAL DEVELOPMENT

- Fundamental part of **Agile** development methods
- Better method when project has **changing requirements**
  - This is the case for most software products
- Three major benefits compared to Waterfall
  - Cheaper/easier to implement **changed requirements**
  - Easier to get user **feedback** (using intermediate versions)
  - **Early delivery**/deployment possible (without all features)
- Agile doesn't mean: without a plan
  - Rather, easy to **change** the plan

SCRUM OVERVIEW



SCRUM TERMINOLOGY

- **Sprint**
  - Iteration (typically 2-4 weeks)
  - Produces a potentially shippable product (an "increment")
- **Product Backlog**
  - Prioritised list of user stories (~> features / bugs)
- **Sprint Backlog**
  - Selected items from the product backlog for the current sprint

After each spring the branches are merged into main.

A person may have multiple scrum roles.

In this course every student is both developer and project owner, whereas the TA is the scrum master.

Each sprint planning meeting reviews the product backlog and sets a new sprint backlog (a subset of items) for the week (duration of the sprint).

ROLES IN A SCRUM TEAM

- **Scrum master**
  - Educates the team
  - Helps to keep on track
  - Removes impediments
- **Project owner**
  - Maintains the backlog
  - Communicates with development team
  - Bridge to customer
- **Development team**
  - Develop
  - Estimate time for backlog items
  - Raise impediments
  - Reflect and adapt

Estimating time for stories

- Multiple methods possible, one being ***Planning Poker***;
  1. PO presents the user story
  2. Every shows a card that represents their estimate
     Unit can be days, ideal days, story points, …
  3. Lowest + highest bidders explain their choice
  4. Repeat until consensus

Sprint

1. Development time. 👩‍💻
2. Throughout sprint, keep others informed about status.
   Example flows are;
   - To Do ➡️ Doing ➡️ Done.
   - To Do ➡️ Testing ➡️ Review ➡️ Done.

Daily Scrum (~15 min)

- Inspect progress, and adapt backlog as necessary

Everyone working on the sprint backlog participates

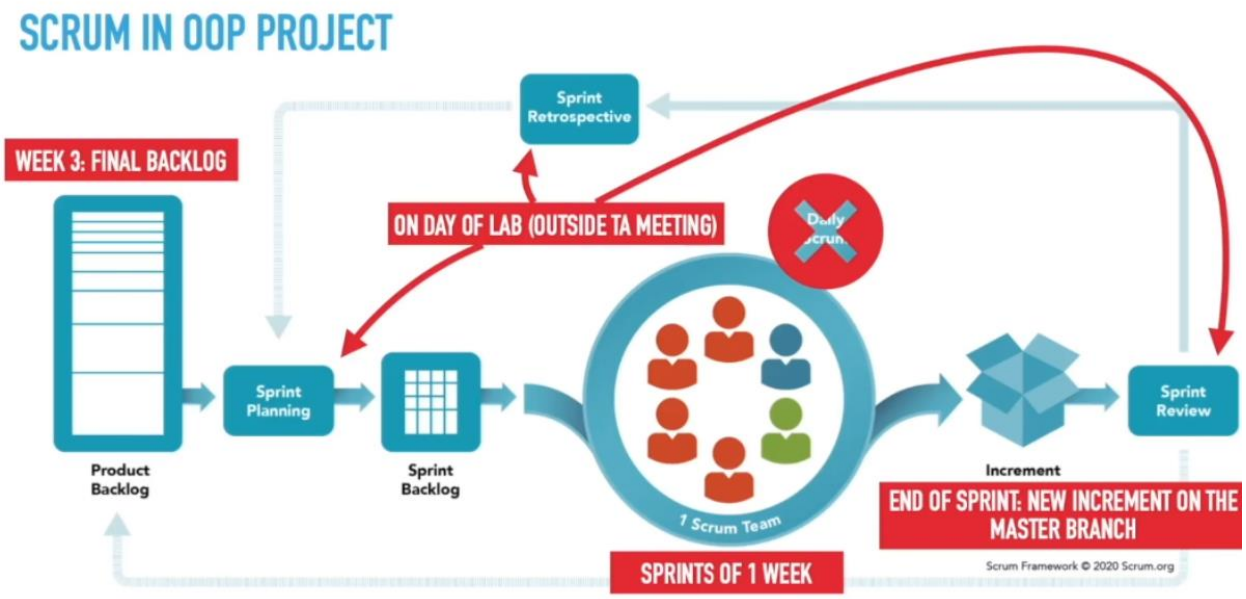
1. What have you done yesterday?
2. What are you going to do today?
3. Do I have any impediments?

Sprint Review

- Team presents result of sprint to key SHs
- PO checks for acceptance criteria
  - When is something "done"?
    Define "done" within your team
  - Only deliver done stories
- PO may propose new items for backlog
- PO + team: to ship or not to ship? 🚀 (with the TA)

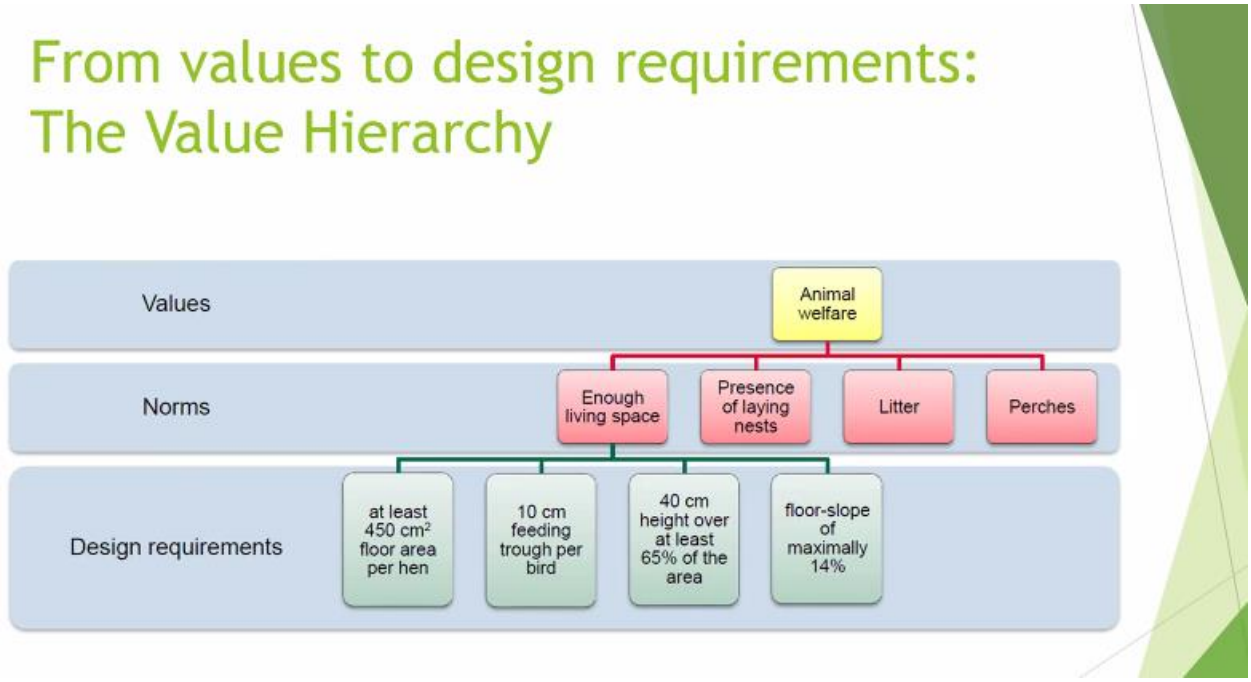
Sprint Retrospective

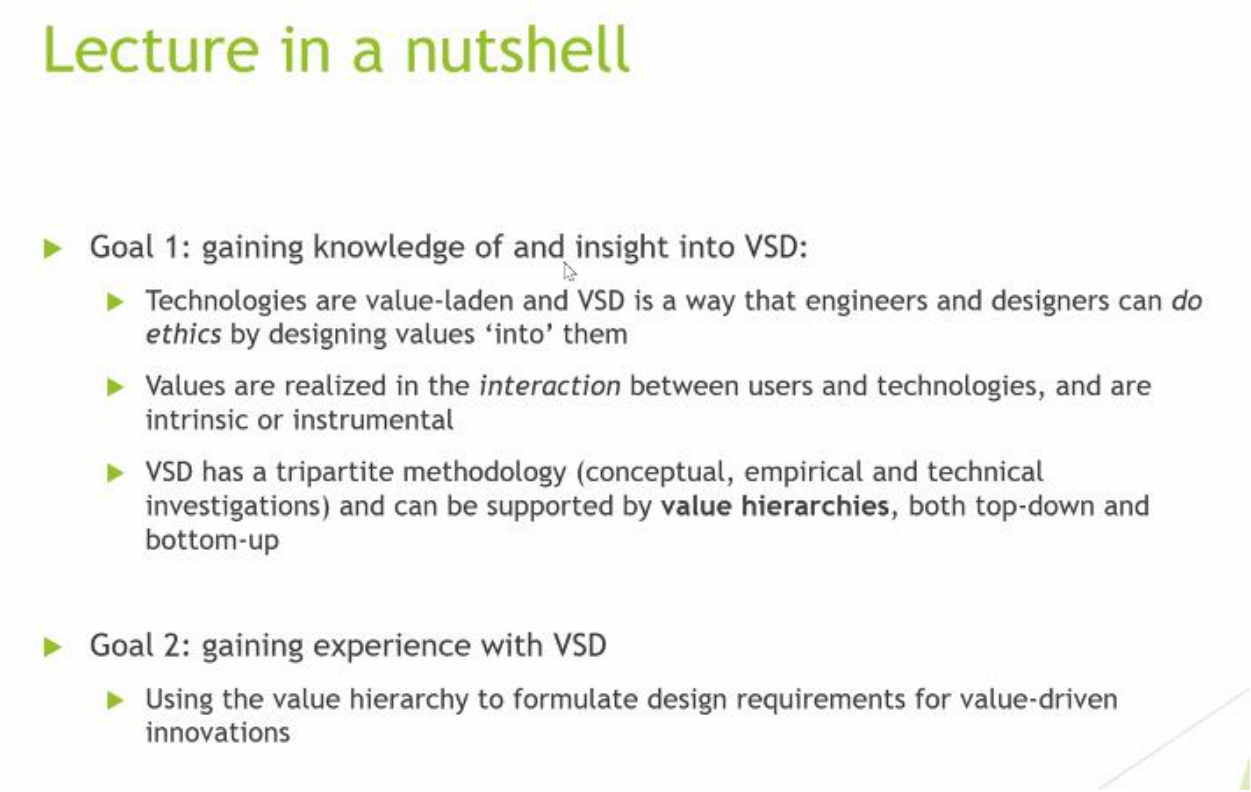
- Reflect on team's behaviour
- What went well? What can be improved?
- Actionable points
- Review improvement points from last retrospective



- The master branch shall only be updated with WORKING versions of the product

Lecture 5. Responsible Computer Science (Design, values)

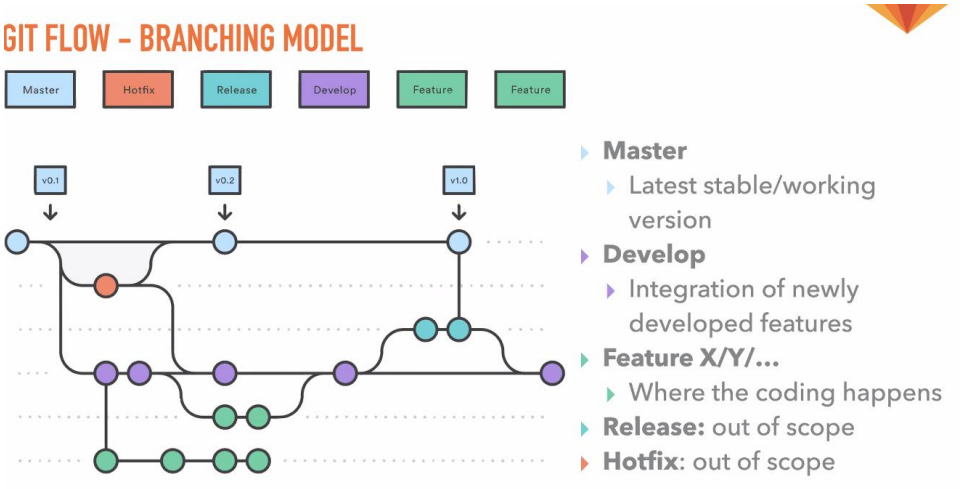




Values used in class

Group numbers	Value
1,9,17,25,33,41	Freedom
2,10,18,26,34,42	Friendship
3,11,19,27,35,43	Health
4,12,20,28,36,44	Equality
5,13,21,29,37,45	Safety
6,14,22,30,38,46	Curiousity
7,15,23,31,39,47	Privacy
8,16,24,32,40,48	Transparency

Lecture 6. Gitlab



Before merging there should be a test branch (release)

Lecture 7. Information Literacy

Where to find information

http://informationliteracy.tudelft.nl/Informatievaardigheden1/NED/L2O2_Tabel_Vergelijking_Google_Scholar_WorldCat_Discovery_and_Scopus.pdf

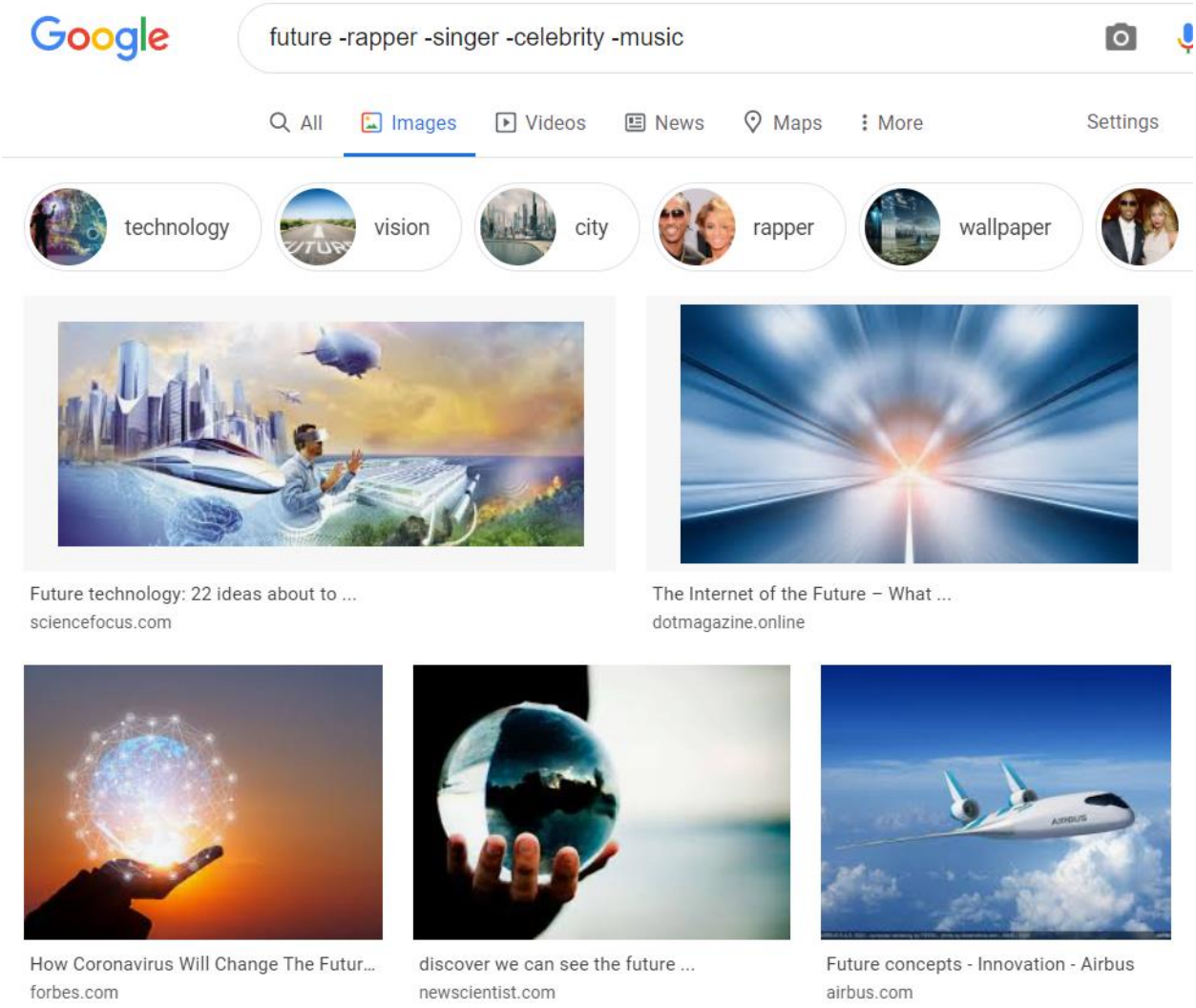
There is a huge number of different information sources available, but you won't need all of them right away. Where do you find what?

What?	**Where?**
Books, journals and other material that can be retrieved and studied at TU Delft Library	WorldCat Discovery
Scientific articles	Scopus* or Google Scholar
Standards (*agreements on the basis of consensus concerning the specifications of a product, service or business process. Also called norms.*)	NEN Connect
Patents (*a form of intellectual property. It consists of a set of exclusive rights granted by a sovereign state to an inventor or their assignee for a limited period of time in exchange for the public disclosure of an invention.*)	Espacenet
Websites	Google
Information sources relevant to your field of study	TU Delft Library website

** Scopus is just one example of a scientific database. In the collection of the TU Delft Library you can find many other scientific databases as well.*

Search shortcuts

- Use "double quotes for literal string order"
- Use -categoryToIgnore to remove irrelevant results
- https://tulib.tudelft.nl/searching-resources/search-operators/



Lecture 8. Spring & Testing

Dependency Injection

Use Guice For Dependency Injection

```
<dependencies>
    <dependency>
        <groupId>com.google.inject</groupId>
        <artifactId>guice</artifactId>
        <version>4.2.3</version>
    </dependency>

    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Add dependency to pom.xml

```
public class MyConfig implements Module {

    public void configure(Binder binder) {
        binder.bind(IFileUtils.class)
                .toInstance(new FileUtils("somefile.txt"));
    }
}
```

Create Config

```
Injector injector = Guice.createInjector(new MyConfig());
ContentProcessor cp = injector.getInstance(ContentProcessor.class);
cp.run();
```

Use Injector

Testability: Create DOC Replacement

```
public class TestFileUtils implements IFileUtils {

    public List<String> calls = new LinkedList<>();
    public String toReturn = "";
    public String lastWriting;

    @Override
    public String readContentFromFile() {
        calls.add("read");
        return toReturn;
    }

    @Override
    public void writeContentsToFile(String content) {
        calls.add("write");
        lastWriting = content;
    }
}
```

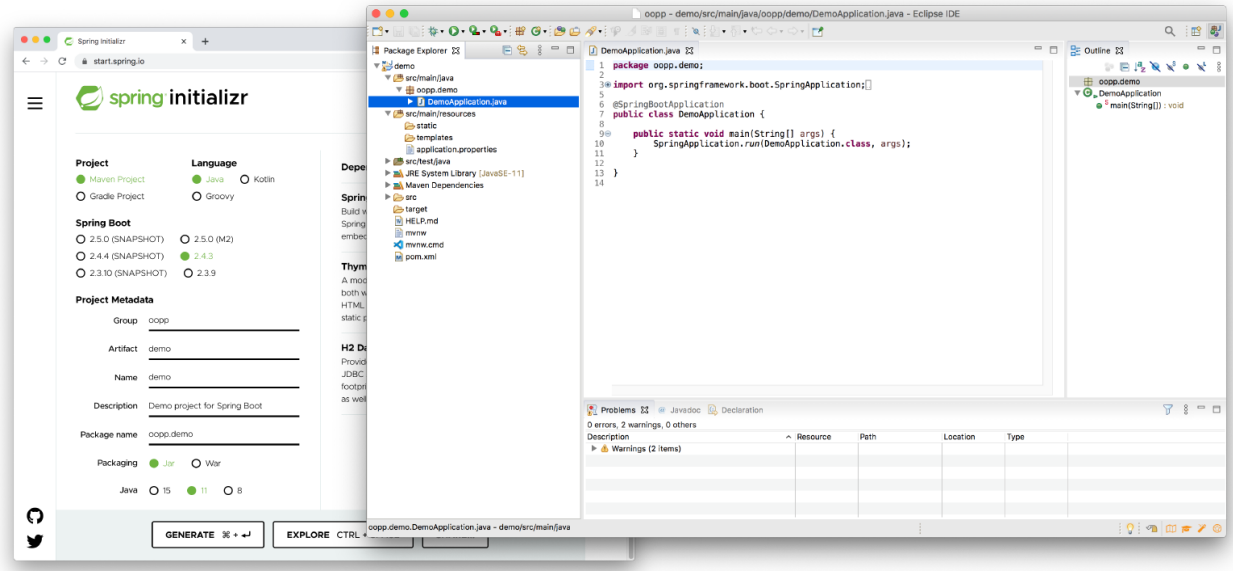
Test Example: Indirect Input/Output

```
public class ContentProcessorTest {

    private TestFileUtils doc;
    private ContentProcessor sut;

    @Before
    public void setup() {
        doc = new TestFileUtils();
        sut = new ContentProcessor(doc);
    }

    @Test
    public void textShouldBeLowered() {
        doc.toReturn = "aBc";
        sut.run();
        String actual = doc.lastWriting;
        String expected = "abc";
        assertEquals(expected, actual);
    }
```

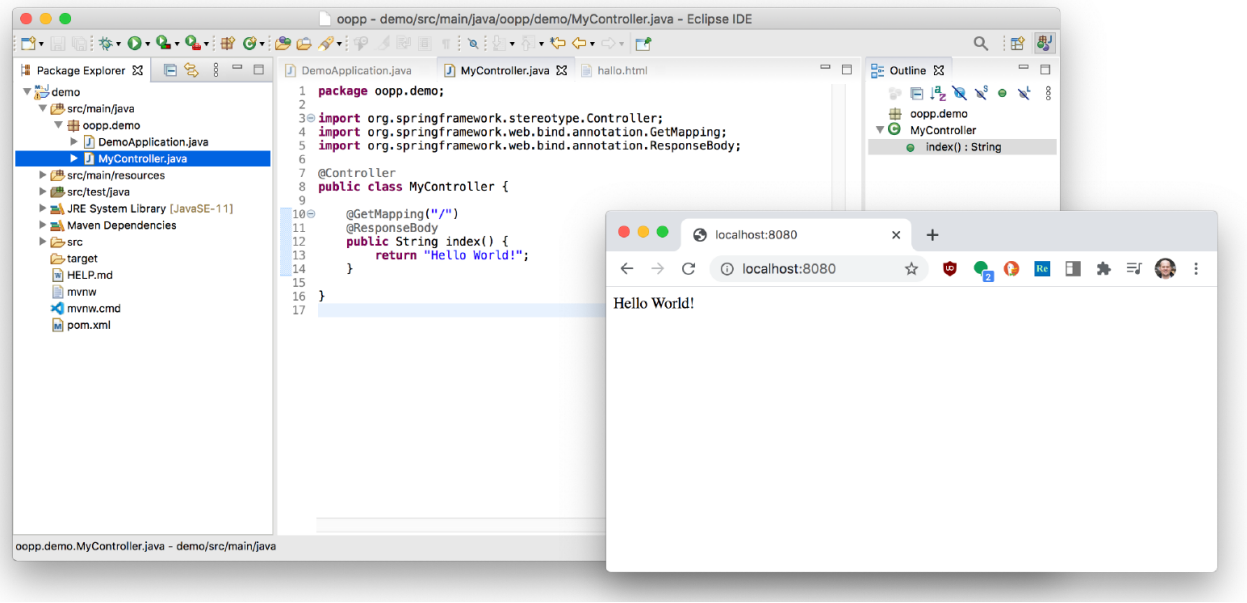
Test Example: Behavior

```
@Test
public void correctCallOrder() {
    sut.run();

    List<String> expected = new LinkedList<>();
    expected.add("read");
    expected.add("write");
    assertEquals(expected, doc.calls);
}
```

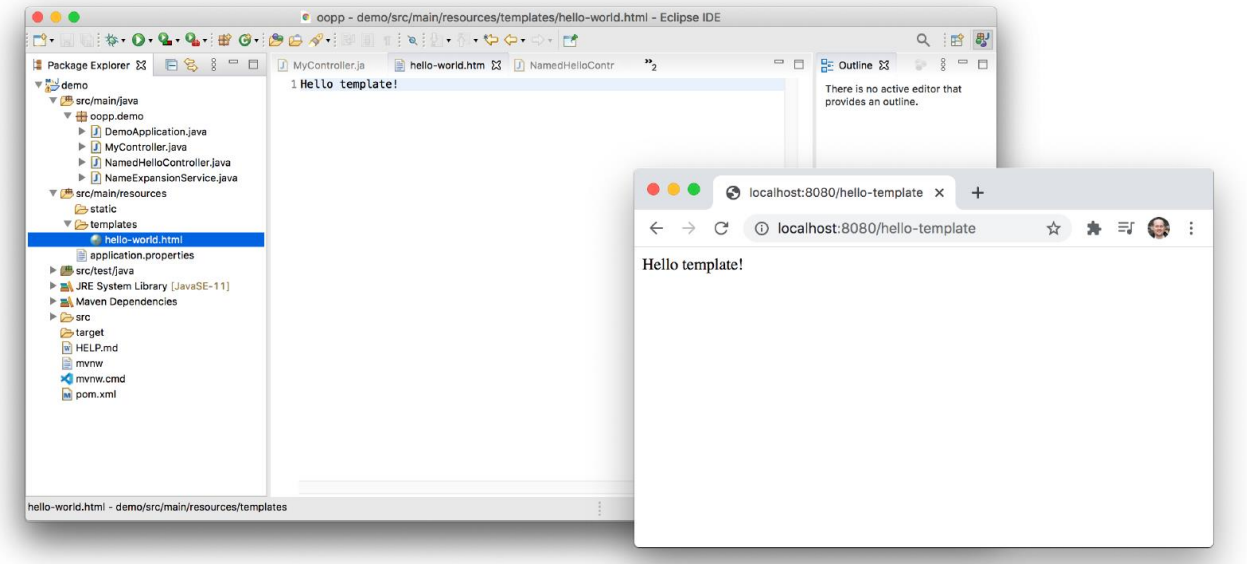
Spring Initializer



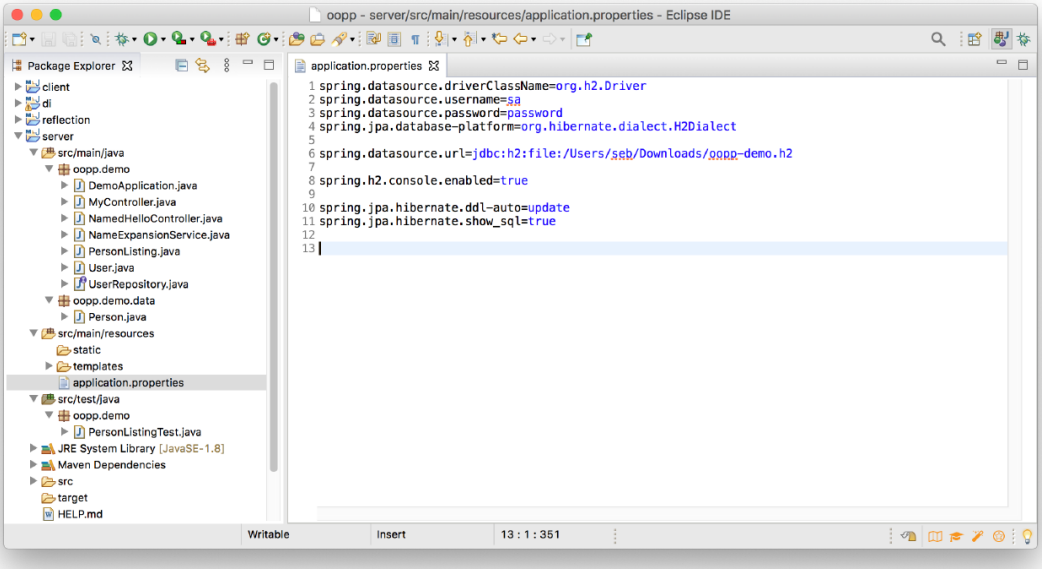
Spring Controller



Use Thymeleaf As a Simple HTML Template Engine



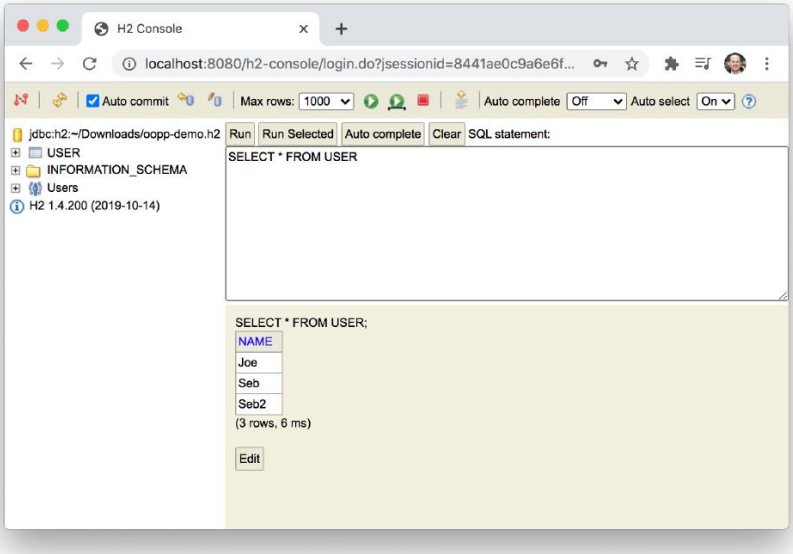
Database Access: Spring Configuration



Database Access: Use Repository

```java
@Controller
@RequestMapping("/named")
public class NamedHelloController {

    private NameExpansionService nes;
    private UserRepository users;

    public NamedHelloController(NameExpansionService nes, UserRepository users) {
        this.nes = nes;
        this.users = users;
    }

    @GetMapping("/name/{name}")
    @ResponseBody
    public String named(@PathVariable("name") String name) {

        if (!users.existsById(name)) {
            User u = new User();
            u.name = name;
            users.save(u);
        }

        return "Hello " + nes.expand(name) + "!";
    }
```

Database Access: Use Console For Exploration



Use An Object Mapper (Jackson, GSON, JSON.simple, ...)

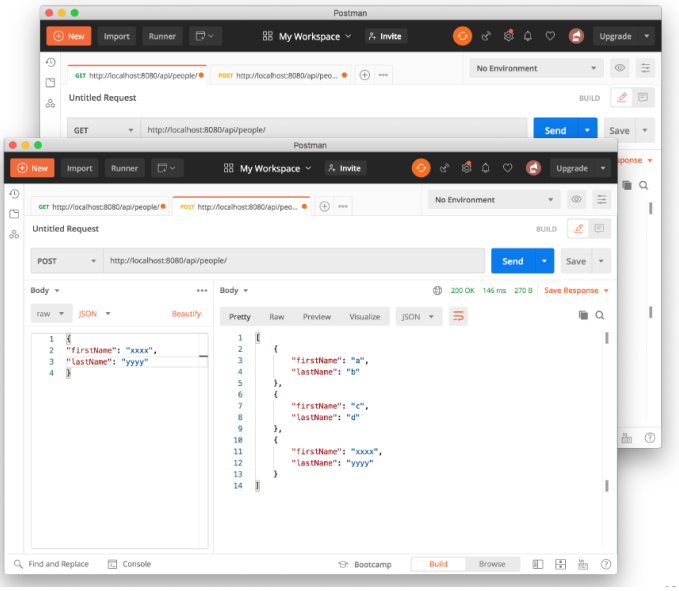
```
public class Person {

    public String firstName;
    public String lastName;

    public Person(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    // either provide parameter-free constructor for OM...
    private Person() {}

    // ... or do not use an explicit constructor
    public static Person create(String first, String last) {
        Person p = new Person();
        p.firstName = first;
        p.lastName = last;
        return p;
    }

    public int hashCode() {

    public boolean equals(Object obj) {

    public String toString() {
}
```

```
ObjectMapper OM = new ObjectMapper();

Person a = new Person("Sebastian", "Proksch");
String json = OM.writeValueAsString(a);

System.out.println(json);

Person b = OM.readValue(json, Person.class);

if (a != b && a.equals(b)) {
    System.out.println("equals!");
}
```

```
{"firstName":"Sebastian","lastName":"Proksch"}
equals!
```

Spring REST Controller

```
@RestController
@RequestMapping("/api/people")
public class PersonListing {

    private List<Person> people = new LinkedList<>();

    public PersonListing() {
        people.add(new Person("a", "b"));
        people.add(new Person("c", "d"));
    }

    @GetMapping("/")
    public List<Person> list() {
        return people;
    }

    @PostMapping("/")
    public List<Person> add(@RequestBody Person p) {
        if (!people.contains(p)) {
            people.add(p);
        }
        return people;
    }

}
```



Add Dependencies For Jersey And Jackson To Client

```
<dependencies>
    <!-- depend on the other project to get access to data structure -->
    <dependency>
        <groupId>oopp</groupId>
        <artifactId>server</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>

    <!-- All Jersey dependencies for the REST requests -->
    <dependency>
        <groupId>org.glassfish.jersey.core</groupId>
        <artifactId>jersey-client</artifactId>
        <version>3.0.1</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.inject</groupId>
        <artifactId>jersey-hk2</artifactId>
        <version>3.0.1</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.media</groupId>
        <artifactId>jersey-media-json-jackson</artifactId>
        <version>3.0.1</version>
    </dependency>
</dependencies>
```

This is a customized Jackson release for Jersey… for "Stand-alone Jackson", look out for FasterXML Jackson.

(This is an excerpt of a Maven pom.xml)

Jersey

```
Person p = new Person("jersey", "client");

Entity<Person> requestBody = Entity.entity(p, MediaType.APPLICATION_JSON);

GenericType<List<Person>> responseBodyType = new GenericType<List<Person>>() {
};

List<Person> people = ClientBuilder.newClient()//
        .target("http://localhost:8080/api/people/")//
        .request(MediaType.APPLICATION_JSON)//
        .accept(MediaType.APPLICATION_JSON)//
        .post(requestBody, responseBodyType);


System.out.println(people);
```

Declare the serialization format, like JSON, XML, …

Define the entity that should appear in request body.

Define the returned type. Normally you can just provide the .class, e.g., List.class, but for generic types, a helper object is needed.

https://spring.io/guides/gs/testing-web/

After this lecture, you should should be able to...

- create a testable design for complex systems
- test indirect input and output of dependent-on components
- use dependency injection to decouple your system components
- setup a basic Spring app with REST endpoints
- consume REST endpoints on clients through Jersey
- use object mappers and stop worrying about serialization formats
- understand the underlying mechanism of many frameworks
- write proper unit tests for your Spring applications