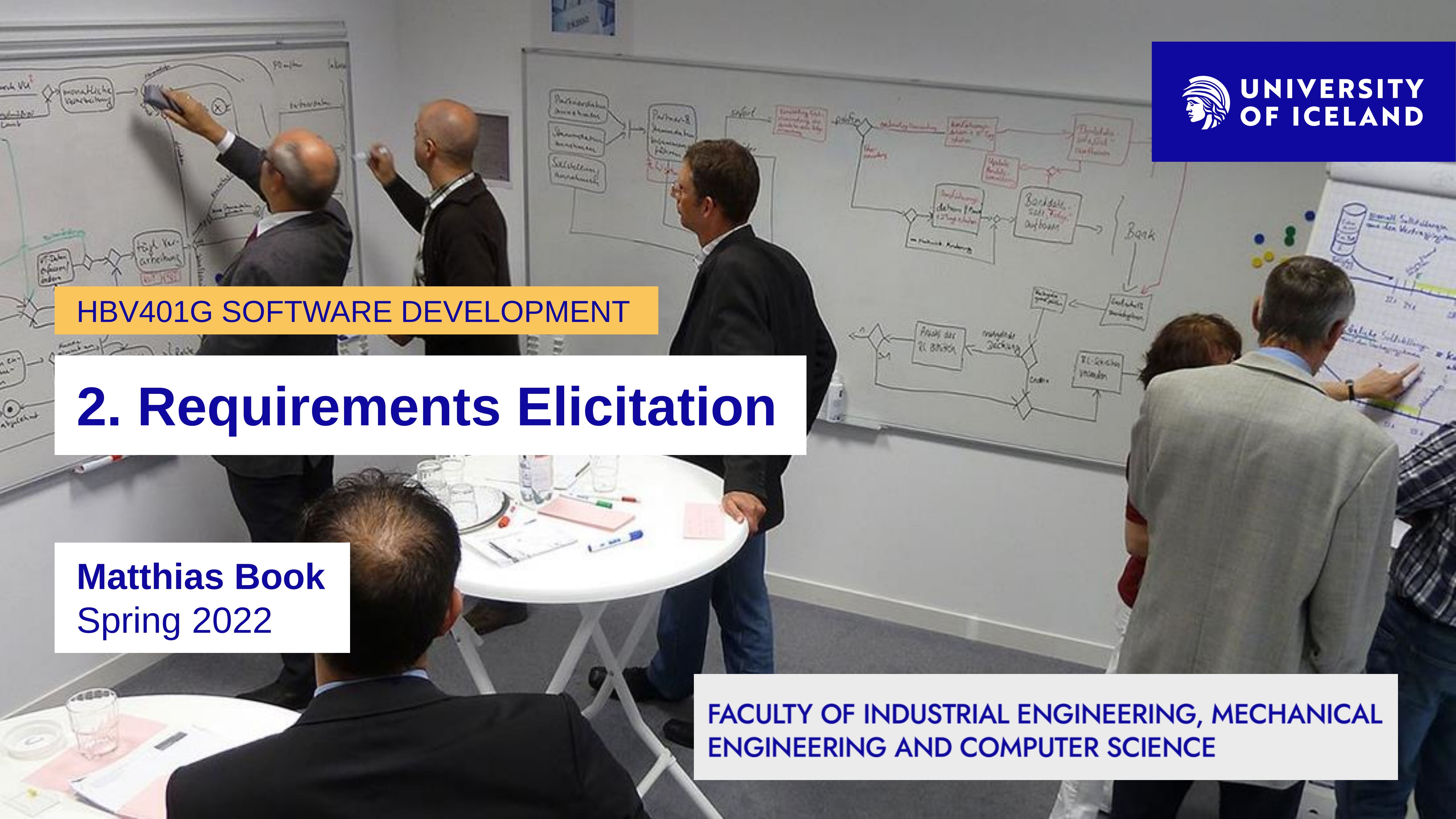


UNIVERSITY OF ICELAND

HBV401G SOFTWARE DEVELOPMENT

2. Requirements Elicitation

Matthias Book
Spring 2022

FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

Update: Team Consultation Schedule

Timeslots	Teams			
Wed 15:00-15:20	1D (1)	3D (4)	5D (4)	7D (4)
Wed 15:25-15:45	1F (4)	3F (4)	5F (4)	7F (4)
Wed 15:50-16:10	1H (4)	3H (4)	5H (4)	7H (4)
Wed 16:15-16:35	1T (4)	3T (4)	5T (0)	7T (0)
Wed 16:40-17:00	2D (4)	4D (4)	6D (4)	8D (3)
Wed 17:05-17:25	2F (0)	4F (4)	6F (4)	8F (4)
Wed 17:30-17:50	2H (4)	4H (4)	6H (4)	8H (4)
Wed 17:55-18:15	2T (0)	4T (4)	6T (3)	8T (2)

- Team IDs: [Cluster][Component] (e.g. Team 4H: Cluster 4, Hotels component)
- Please meet your tutor in your team's voice channel ("Team XX") on Discord at the scheduled time every Wednesday

Update: Course Support

- **Questions and feedback** welcome anytime! Preferably:
  - during the lectures on Zoom (just interrupt me anytime)
  - during the team consultations on Discord (just ask your tutor)
  - anytime on the Discord channels (just ask the @Teachers and/or your fellow students)

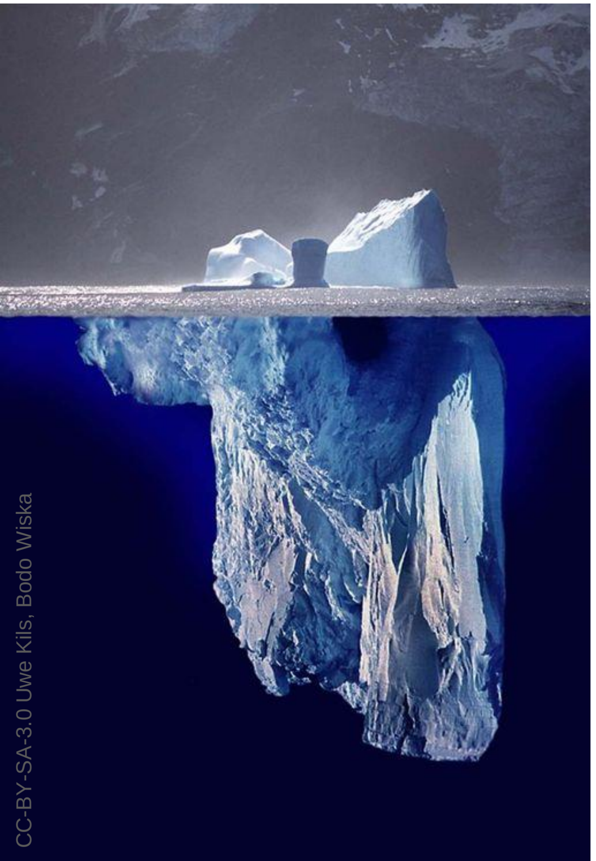
- **See Canvas for**
  - Lecture slides and recordings
  - Important course announcements

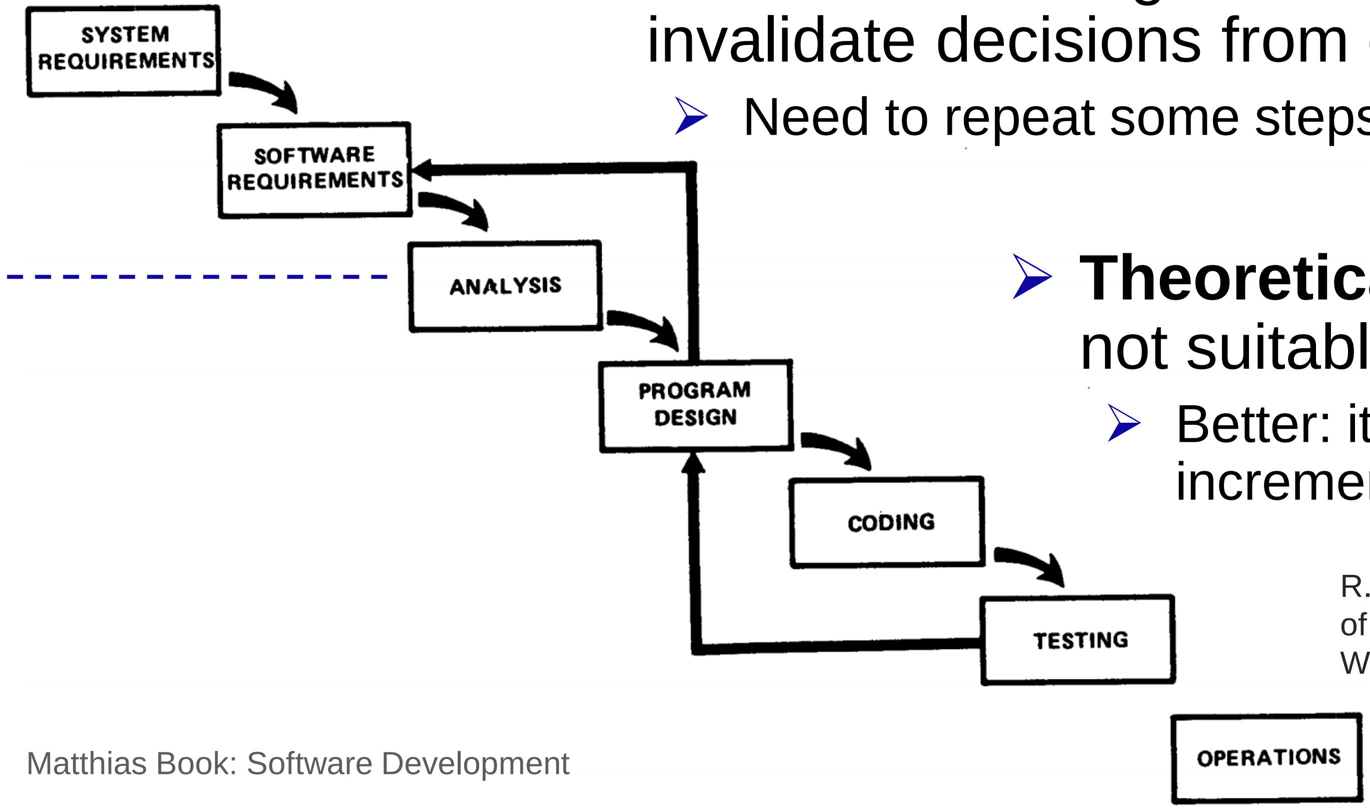
- **Teaching Staff**

Matthias Book	(book@hi.is)
Marcelo Felix Audibert	(mfa5@hi.is)
Jaan Jaerving	(jaj20@hi.is)
Tristan Freyr Jónsson	(tfj2@hi.is)
Valbjörn Jón Valbjörnsson	(vjv1@hi.is)

Recap: The Waterfall Model

- Comprises typical steps that can be found in any software process model
- **Problem #1:** Virtually impossible for a developer to get right without client input
  - Need frequent feedback from client





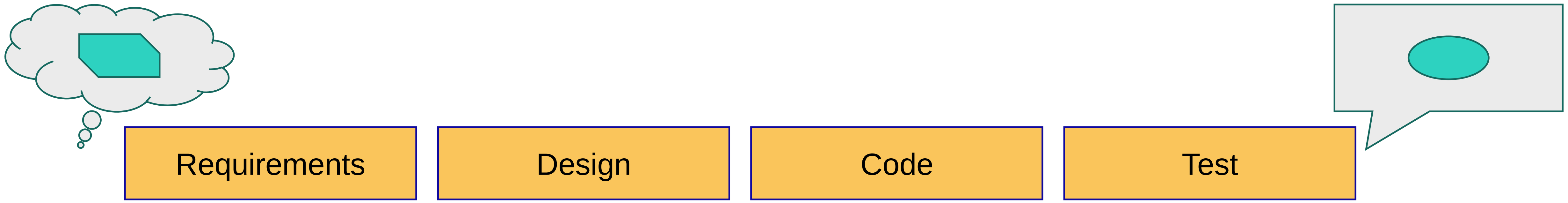
- **Problem #2:** Insights of one phase may invalidate decisions from earlier phases
  - Need to repeat some steps

- **Theoretical model,** not suitable for practice
  - Better: iterative-incremental model

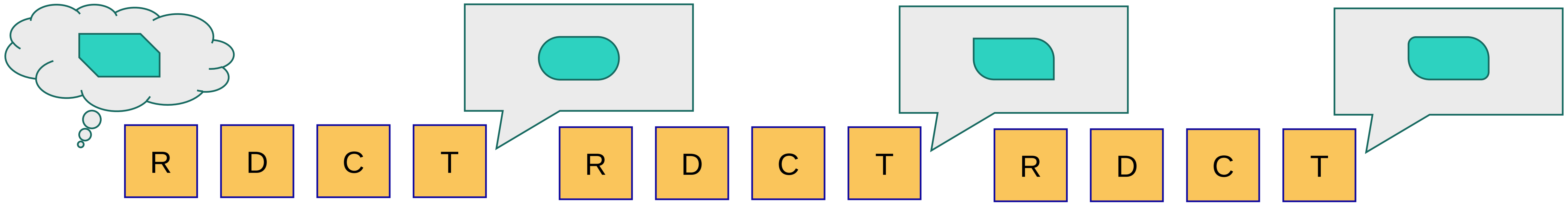
R. Winston: Managing the Development of Large Software Systems, Proc. IEEE WESCON 26, pp. 1–9 (1970)

Recap: Iterative-Incremental Software Development

- ~~Big-Bang Approach~~



- Iterative Development



Recap: Software Process Models

- Some typical software project management considerations:
  - How long should an iteration take?
  - Who should be involved in an iteration?
  - What activities are performed in an iteration?
  - How much can be accomplished in one iteration?
  - How can I leverage iterations to deal with uncertainty and change?
  - How do I make sure I’m still on the right track in terms of budget and schedule?

- A **software process model** provides guidelines for…
  - Activities to execute
  - Artifacts to create
  - Roles to involve
  - Checks to perform
- …in building a high-quality software product.

Recap: Software Process Models

Focus of this course

Plan-Driven Models

- Attempt to understand and specify large parts of the system in detail at the beginning of the project
- Developers are assigned tasks
- Fixed schedule for component deliveries throughout project
- Small flexibility for midway changes
- Higher (perceived) control of project progress and outcome
- e.g. Spiral Model, RUP, V Model

Agile Models

- Start with a rough project vision, refine through many prototyping and customer feedback cycles
- Developers pick their own tasks
- Component deliveries can be re-prioritized and rescheduled anytime
- No fixed specifications or plans
- Lower (perceived) control of project progress and outcome
- e.g. XP, Scrum, Kanban

Quiz #1 Solution: Software Process Models

- **Indicate which of the following statements apply to plan-driven models (P) and which ones apply to agile models (A):**

a) Attempt to understand and specify large parts of the system in detail at the beginning of the project (P)

b) Component deliveries can be re-prioritized/rescheduled anytime (A)

c) Developers are assigned tasks (P)

d) Developers pick their own tasks (A)

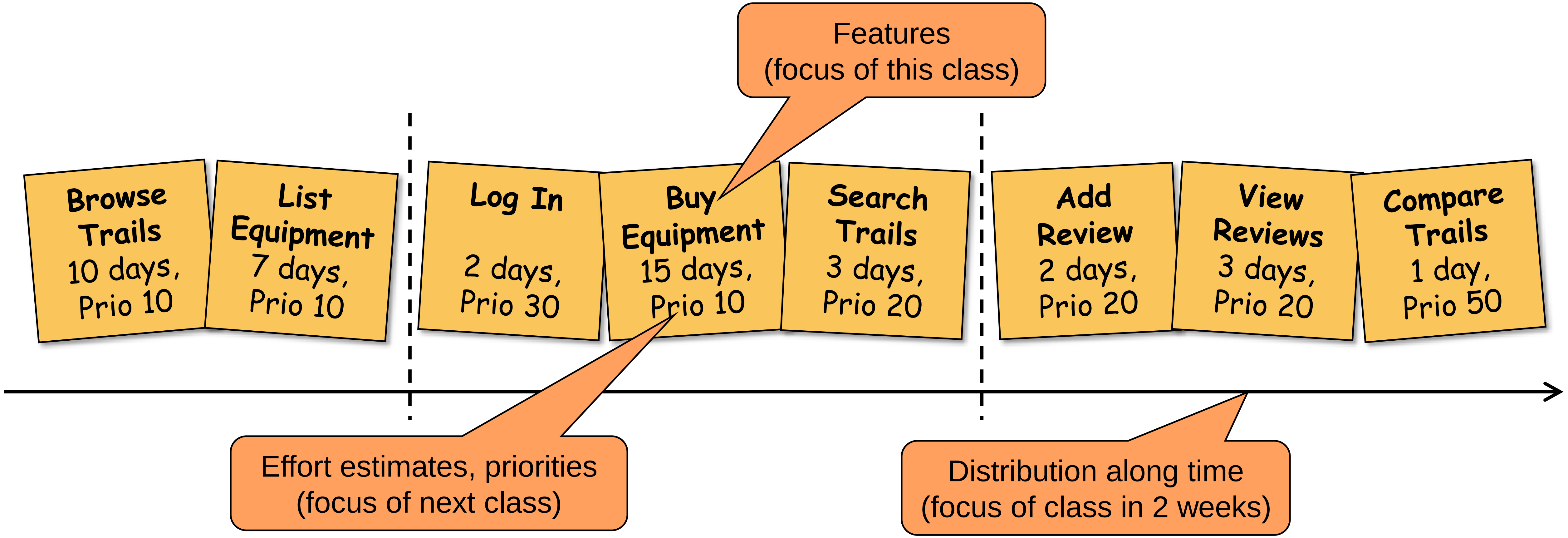
e) Fixed schedule for component deliveries throughout project (P)

f) No fixed specifications or plans (A)

g) Small flexibility for midway changes (P)

h) Start with a rough project vision, refine through many prototyping and customer feedback cycles (A)

Recap: Planning an Iterative Software Project



- **Plan-driven models** try to figure this out in the beginning, and stick to it
- **Agile models** write and revise the plan for the next few iterations as they go

Gathering Requirements

see also:

Head First Software Development, 1st half of Chapter 2

Distinguishing Problem Domain and Solution Domain

$$\begin{tikzcd}
& \textbf{Problem} \arrow[dr, "\textit{triggers}"] & \\
\textbf{Solution} \arrow[ur, "\textit{solves}"] & & \textbf{Requirement} \arrow[ll, "\textit{shapes}"]
\end{tikzcd}$$

- Problems, requirements and solutions are often mixed up in talks with clients
  - “A warning light should turn on if tire pressure is too low, to prevent the tire from bursting.”
- Better way of thinking:
  - **Problem:** “Tires may burst if continuously run on too low pressure.”
  - **Requirement:** “The driver must be warned if tire pressure drops below a critical value.”
  - **Solutions:** Measure tire pressure/ Calculate tire pressure — turn on warning light/ play spoken warning — if pressure < 30 psi/ if difference b/w tires > 2 psi

Types of Requirements

- **Functional requirements** define features that the system must provide.
  - e.g. “The driver shall be warned if tire pressure drops below a critical value.”
- **Quality requirements** define qualitative characteristics of individual features or whole systems.
  - reliability, usability, efficiency, changeability, adaptability
  - e.g. “The tire pressure shall be determined within at most 3 minutes of driving time.”
- **General conditions** comprise organizational, technical, legal… prerequisites.
  - e.g. “Tire-pressure sensors communicate via Bluetooth.”

Underspecified Requirements

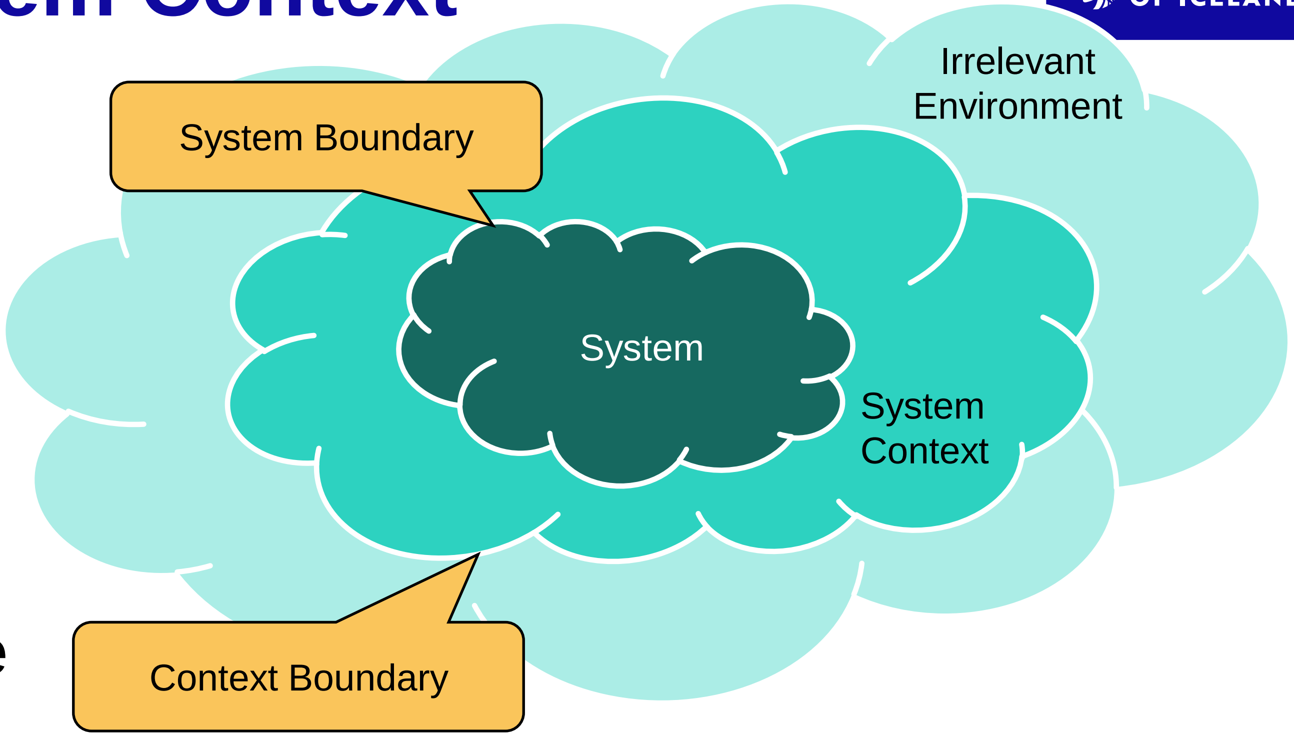
- Requirements should be concrete and precise in describing the WHAT
  - but without prescribing solutions, i.e. the HOW

- Bad example:
  - “The system must be secure.”
- Better:
  - “The system shall check the user’s name and password before allowing access to X.” (functional requirement)
  - “All communications shall use strong asymmetric encryption.” (quality requirement)

- General conditions must be turned into requirements at some point:
  - “All senior executives shall use two-factor authentication.”
  - Not enough to just state this – which requirements does it translate into for our system?

System Boundary and System Context

- The **system boundary** separates the planned system from its environment. It separates that part of reality which can be defined and changed by the development process from the part of reality that cannot be changed.
- The **context boundary** separates the part of the environment that can influence the system under construction from the part that has no influence on the system.
- *These boundaries are typically not fixed, but emerge during the project!*



- The **system context** is the part of the environment of a system that is relevant for the definition and understanding of the system's requirements.

Requirements Elicitation

- **Goals:**
  - Identify sources of requirements
  - Elicit requirements from the sources and the system context
- **Risks** of an insufficient requirements elicitation:
  - Key requirements overlooked
  - Key requirements discovered late in the project (→ more costly to implement)
  - User acceptance of system endangered
- **Note:**
  - Requirements do not exist per se, but must be worked out!

Sources of Requirements

- **Stakeholders**
  - Client, user, operator, architect, developer, workers’ council, privacy officer, …
- **Documents**
  - Laws, norms, standards
  - Domain-/organization-specific documents
    (regulations, best practices, work procedures, business process models…)
- **Existing systems**
  - Operational/legacy systems
  - Competing systems
  - Systems and interfaces of the system context

Three Simple Requirements Elicitation Techniques

- **Brainstorming**
  - Get as many *different* stakeholders involved as possible
  - Collect everyone's ideas, no matter how trivial or crazy – no filters initially
  - Qualify feature ideas later: How important is it? How much effort, how much risk is involved?
- **Observation**
  - Unobtrusively watch people perform the tasks that you want to support
  - Observe how they deal with normal cases, unusual cases, mistakes
  - Observe which general conditions they are bound by
  - Identify inefficiencies, potential optimizations (but confirm your assumptions!)
- **Role playing**
  - Run through a feature in dialog form – one person plays the user, the other plays the system
  - Note assumptions the system makes – how to base them on known data?
  - Everything the system does must be implemented, i.e. is a requirement

Quiz #2: Requirements Types

- **Indicate if the following are (F)unctional requirements, (Q)uality requirements or (G)eneral constraints:**

a) Attach a file to an e-mail
b) E-mail addresses must not contain spaces
c) E-mail attachments are always sent MIME-encoded
d) Enable sender to encrypt and sign e-mails
e) E-mails exchanged between users of the same mail server shall arrive within 10 seconds
f) Prevent unauthorized access to a user's e-mail account
g) Send an e-mail to several recipients
h) Server must have <5% downtime

UNIVERSITY OF ICELAND

Break

Samples of Initially Collected Coarse Features

Book flight
A user will be able to book a flight on a specified date and time.

Review flight
A user will be able to leave a review for a flight they have been on.

Choose seating
A user will be able to choose aisle or window seating.

Use Ajax for the UI
The user interface will use Ajax technologies to provide a cool and slick online experience.

Pay online
Users will be able to pay for their bookings by credit card or PayPal.

Support 3000 concurrent users
Traffic is expected to reach 3000 users using the site simultaneously.

Ask More Questions!

- Validate the requirements you come up with by running them by the client.
- Be aware of assumptions you make; validate these with the client as well.
- Ask for clarification of domain aspects that you are uncertain about.
- Ask about rationales for difficult requirements, to help find alternatives.
- Ask about additional requirements that may have come up meanwhile.
  - But make sure the client is aware that additional features cost additional money
  - and need to compete with existing requirements for inclusion in the project schedule.

Possible Conflicts

- **Subject conflict**
  - Missing or incorrect information, diverging interpretation of facts
- **Conflict of interest**
  - Different interests, agendas or goals of stakeholders
- **Value conflict**
  - Different values associated with certain items (e.g. due to cultural differences)
- **Social conflict**
  - Negative social behavior (e.g. rudeness, uncooperative behavior, withholding information)
- **Structural conflict**
  - Unbalanced distribution of power and authority

- Actual conflicts are usually a mix of the above.

Conflict Resolution Techniques

- **Agreement:** Parties agree on a common point of view.
- **Compromise:** Both parties depart from their point of view and “meet halfway”.
- **Vote:** A decision is made by vote of eligible stakeholders.
- **Variants:** Variants reflecting both points of view are built into the system.
- **Authority:** A decision is made by a higher authority.

Refining Coarse Features into User Stories

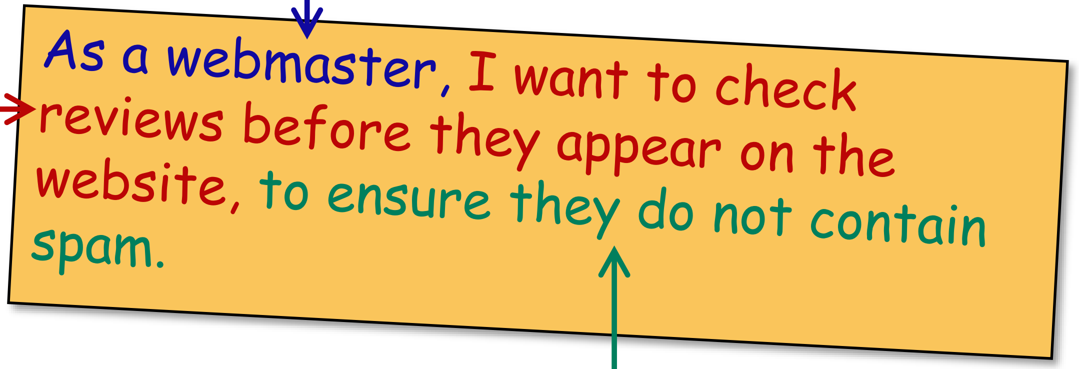
- To plan your work based on the coarse features you collected, translate them into more fine-grained **user stories**.

- User stories should
  - describe one thing that the software needs to do for the client
  - be written using language and terms that the client understands
  - be written from the client's perspective
  - be short (1-2 sentences).
- User stories should not
  - mention specific technologies or technical solutions
  - Mention general conditions or quality requirements that don't translate into features

- Both you *and your client* must understand what a user story means!

Elements of a User Story

- **Role/Perspective**
  - Who will be using this feature?
- **Goal/Function**
  - What task is the system supposed to support?
- **Rationale/Benefit** (optional)
  - What do we want to accomplish by this feature? Why is it important?


As a webmaster, I want to check reviews before they appear on the website, to ensure they do not contain spam.

Requirements that Cannot be Turned into User Stories

Book flight
Users must be able to book a flight on a specified date and time.

Review flight
Users should be able to review flights they have been on.

Order in-flight meals
Users should be able to specify the meals and drinks they want during a flight.

~~**Use Ajax for the UI**~~
~~The user interface will use Ajax technologies to provide a cool and slick online experience.~~

Pay online
Users must be able to pay for their bookings by credit card or PayPal.

~~**Support 3000 concurrent users**~~
~~Traffic is expected to reach 3000 users using the site simultaneously.~~

Too technical – part of the solution domain, not the problem domain

Not a functional requirement, but a quality requirement to consider in designing the system architecture

Sample User Stories for a Feature

Review flight
Users should be able to review flights they have been on.

As a passenger, I want to submit a review for a flight I have been on, in order to express my commendation or frustration to someone.

As a passenger, I want to choose whether my review is public or private, so I can provide more candid details if necessary.

As a webmaster, I want to check reviews before they appear on the website, to ensure they do not contain spam.

As a marketing manager, I want to reply to reviews in public or private, in order to address any mentioned concerns.

As a website visitor, I want to rank reviews I read, in order to help other visitors to find the most helpful reviews.

Writing Good User Stories

- User stories are written in the language of the client
  - Requires domain knowledge
  - Helps to spread domain knowledge in the team

- User stories describe requirements, not solutions

- User stories do not describe technology
  - “~~As a user, I want the application to run on two servers.~~”

- User stories do not describe user interfaces
  - “~~As a user, I want to print the report by clicking the green button in the lower left corner.~~”

Improving User Stories

Bad Examples

- “The application runs on two servers.”
- “The Search feature must be available 24/7.”
- “The reservation engine must be compatible with the Amadeus system.”

Better Examples

- “As a user, I want to be able to use the system without delay even under high load.”
- “As a user, I need access to the Search feature at any time.”
- *Not a functional requirement*
  - ➢ *Don’t express as user story*
  - ➢ *State in technical specification*

Characteristics of User Stories: The INVEST Model

- **Independent**
  - They ideally should not assume details/scheduling of other user stories
  - They imply all necessary work throughout all application layers (interface, logic, database)
- **Negotiable**
  - The client should be able to decide if and when to implement them
  - The team’s effort estimates are not negotiable though
- **Valuable**
  - They should have a justification and add value (as expressed e.g. by their “rationale” part)
- **Estimatable**
  - They should be concrete enough to estimate design, implementation and integration effort
  - But without prescribing a particular technical solution
- **Small**
  - Their realization should not require large amounts of the project resources
  - Estimates over 15 person-days often indicate a story is too complex
- **Testable**
  - It should be clear by which criteria we can confirm that a story has been implemented satisfactorily.

Discussion

- If I include some technical ideas in my user stories,
  won't that make them more useful to my team?

- Am I supposed to do all this refinement of requirements into user stories together with the client?

- That's a lot of up-front requirements work. What about things changing later?

Team Assignment 1a: User Stories

- By **Sun 30 Jan**, submit a **list of user stories** in Canvas describing all the features that your component will offer:
  - Come up with at least 10-20 user stories
  - Try to describe each feature as precisely as possible within the user story format
  - Include roles, functions, rationales in the user story
- On **Wed 26 Jan**, discuss a **draft** of your user stories with your tutor.
- On **Wed 2 Feb**, present and explain your **submitted** user stories to your tutor:
  - Why did you include exclude certain features?
  - Did you consider different stakeholders / usage scenarios etc.?
- **Grading Criteria:**
  - User stories are in the correct format (20%)
  - User stories precisely describe features (50%)
  - User stories cover plausible component scope (30%)

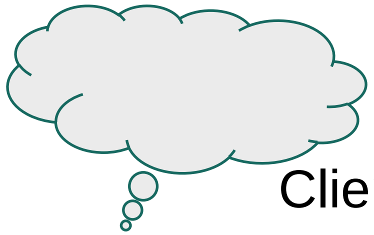
Preview on Team Assignment 1b: Effort Estimates

- In the consultation on Wed 2 Feb, your tutor will give you feedback on the user stories, to make sure your project has the right scope for subsequent assignments.

- Based on this feedback, you will
  - estimate the effort required for your user stories
  - plan the first sprint
- The following slides will cover how to do this.

- The second part of the assignment (effort estimates and product backlog) then needs to be submitted by Sun 13 Feb.

Summary and Outlook: The User Story Refinement Cycle



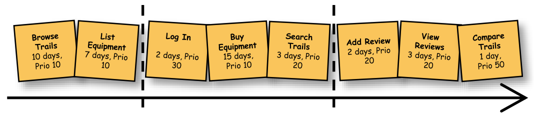
Client ideas

Use requirements elicitation techniques to gather client ideas and requirements

Construct user stories from requirements with client

Fill gaps and add clarity on details with client feedback

Estimate and prioritize user stories


Browse Trails 10 days, Prio 10
List Equipment 7 days, Prio 10
Log In 2 days, Prio 30
Buy Equipment 15 days, Prio 10
Search Trails 3 days, Prio 20
Add Review 2 days, Prio 20
View Reviews 3 days, Prio 20
Compare Trails 1 day, Prio 50

Refine stories into tasks for sprint planning and progress monitoring

Thank you!

book@hi.is