Hugbúnaðarverkefni 2 / Software Project 2

13. Final Exam Preparation

HBV601G – Spring 2019

Matthias Book

HÁSKÓLI ÍSLANDS
VERKFRÆÐI- OG NÁTTÚRUVÍSINDASVIÐ
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI-
OG TÖLVUNARFRÆÐIDEILD

In-Class Quiz #10 Solution: Software Architecture Summary

- A (a) system architecture can be considered from several perspectives:
  - (b) **Module structure:** Design-time view of divisions and relationships of constructed modules
  - (c) **Component-and-Connector structure:** Run-time view of interactions of module instances
  - (d) **Allocation structure:** Design- and run-time view of mapping arch. elements to each other

- A system's architecture is **established through a series of** (e) **design decisions.**
  - e.g. allocation of responsibilities, coordination model, data model, resource management, mapping of architectural elements, variations and binding times, technology choices
- Design decisions **influence** (f) **quality attributes** of the system.
  - e.g. availability, interoperability, modifiability, performance, security, testability, usability
- Decisions affect & depend on technical, project, business, professional (g) **contexts.**

Recap: The Pivotal Role of Software Architecture

- Having a good software architecture early is key to a project's success because:
  - **An architecture will inhibit or enable a system's driving quality attributes.**
  - **Architecture analysis enables early prediction of a system's qualities.**
  - **A documented architecture enhances communication among stakeholders.**
  - **An architecture carries the earliest, fundamental, hardest-to-change design decisions.**
  - Architecture decisions allow you to reason about and manage system change and evolution.
  - An architecture defines a set of constraints on subsequent implementation.
  - An architecture dictates the structure of an organization, or vice versa.
  - An architecture can provide the basis for evolutionary prototyping.
  - An architecture is the key artifact enabling reasoning about cost and schedule.
  - An architecture can be a transferable, reusable model at the heart of a product line.
  - An architecture focuses attention on assembly rather than creation of components.
  - An architecture reduces design and system complexity by restricting design alternatives.
  - **An architecture can be the starting point for introducing team members to a project.**

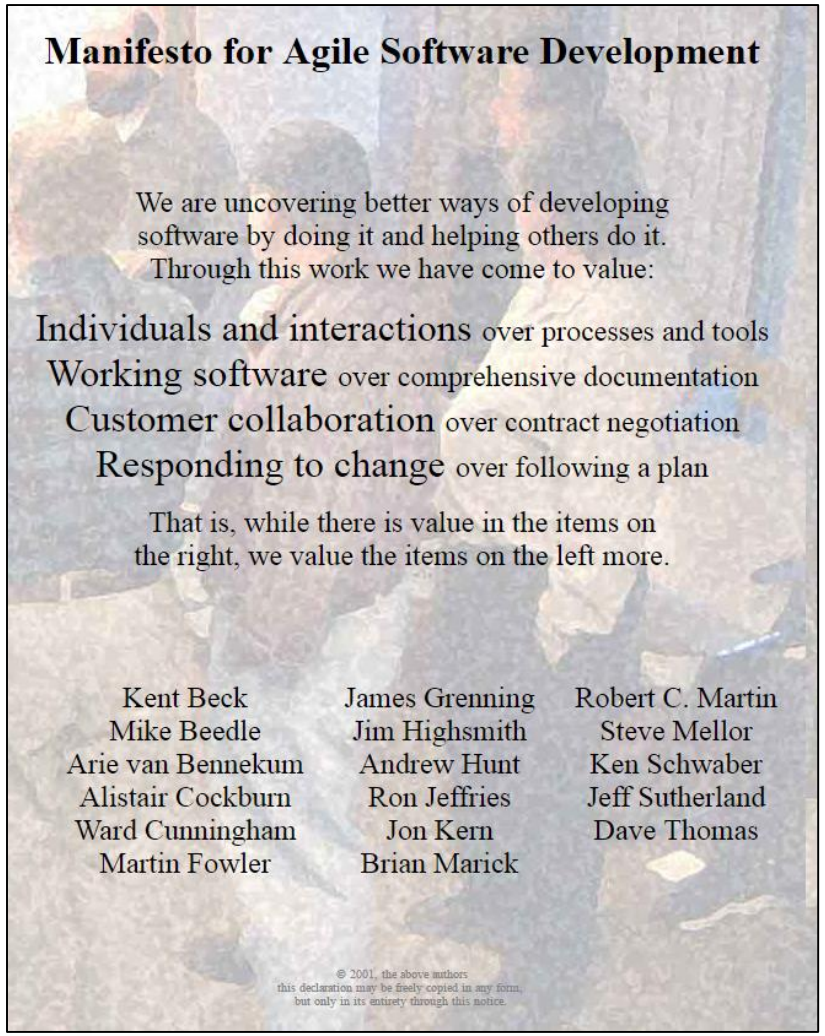
Agile Practices

A Critical Review

Further reading:

- Bertrand Meyer: Agile! The Good, the Hype and the Ugly. Springer, 2014

Criticism of the Agile Manifesto



Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

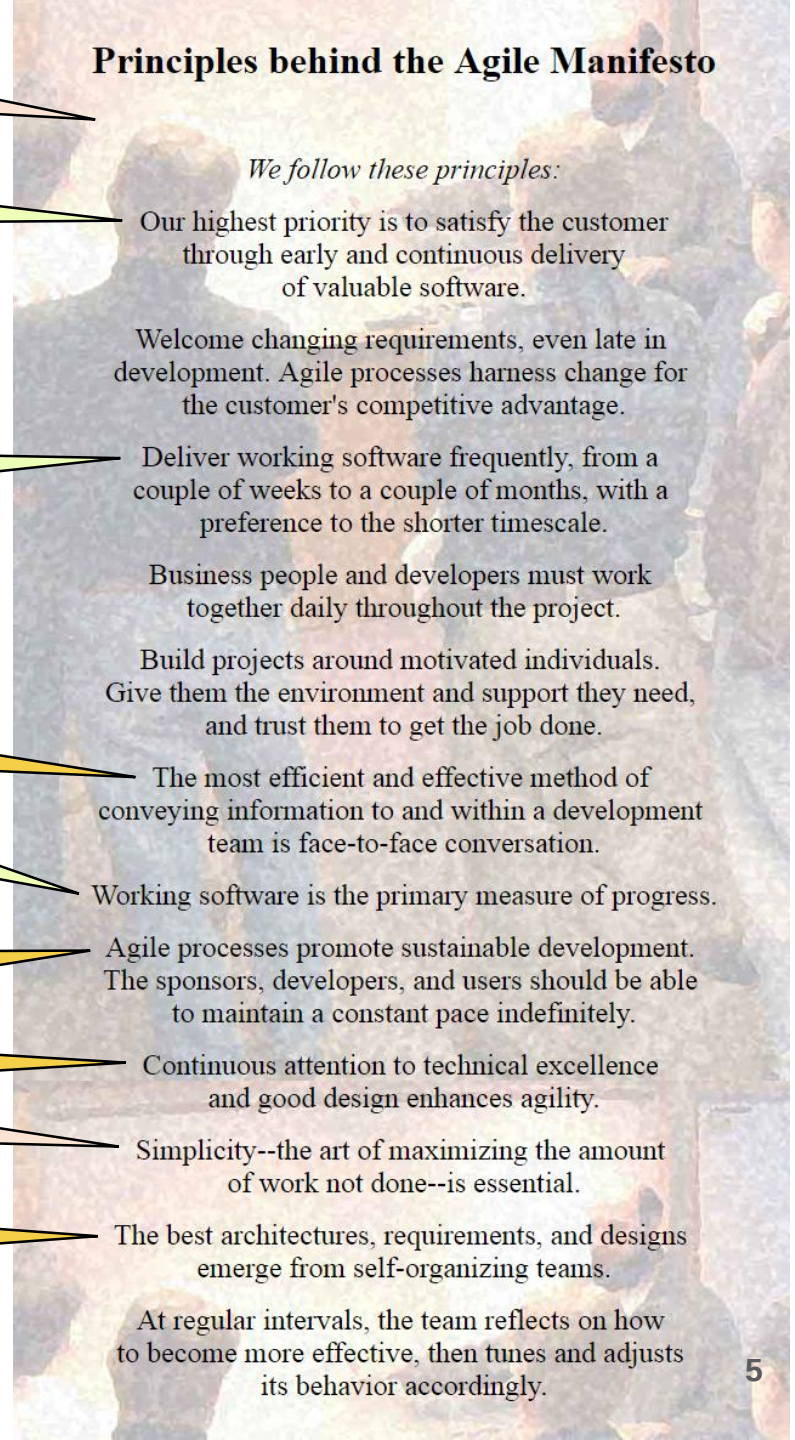
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

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

Overall impression:

- unclear
- partially unsubstantiated
- hard to operationalize



Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Testing…?

Redundant

Redundant

Assertion

Redundant

Assertion

Assertion

Wrong

Assertion

Sources: agilemanifesto.org;
Bertrand Meyer: Agile Methods –
The Good, the Hype and the Ugly (2014)

Agile Practices: “The Ugly”

- **Rejection of up-front tasks, particularly: no up-front requirements**
- **User stories as a replacement for abstract requirements**
- **Feature-based development and ignorance of dependencies**
- Tests as a replacement for specifications
- Test-driven development
- Embedded customer
- Coach and method keeper (e.g. Scrum Master) as a separate role
- Dismissal of traditional manager tasks
- **Dismissal of a priori architecture work**
- **Dismissal of a priori concern for extensibility**
- **Dismissal of a priori concern for reusability**
- Dismissal of auxiliary products and non-shippable artifacts

Source: Bertrand Meyer: Agile Methods – The Good, the Hype and the Ugly (2014)

Agile Practices: “The Indifferent”

- Pair programming
- Open-space working arrangements
- **Self-organizing teams**
- **Maintaining a sustainable pace**
- Producing minimal functionality
- **Planning game, planning poker**
- **Cross-functional teams**

Source: Bertrand Meyer: Agile Methods – The Good, the Hype and the Ugly (2014)

Agile Practices: “The Good”

- **Acceptance of change**
- **Frequent iterations**
- Emphasis on working code
- Tests as one of the key resources of the project
- Constant test regression analysis
- No branching
- Product (but not user stories!) **burndown chart**
- **Daily meeting**

Source: Bertrand Meyer: Agile Methods – The Good, the Hype and the Ugly (2014)

Agile Practices: “The Brilliant”

- **Short iterations**
- **Closed-window rule**
- **Refactoring** (but not as a substitute for design)
- **Associating a test with every piece of functionality**
- **Continuous integration**

Source: Bertrand Meyer: Agile Methods – The Good, the Hype and the Ugly (2014)

Suggestion: More Actionable Agile Principles

Organizational Principles

1. Put the customer at the center
2. Accept change
3. Let the team self-organize
4. Maintain a sustainable pace
5. Produce minimal software
    1. Produce minimal functionality
    2. Produce only the product requested
    3. Develop only code and tests

Technical Principles

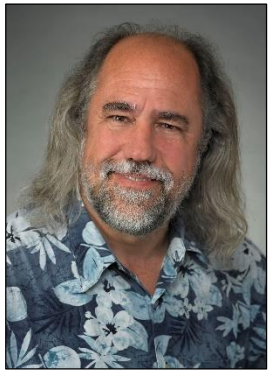
8. Develop iteratively
    1. Produce frequent working iterations
    2. Freeze requirements during iterations
9. Treat tests as a key resource
    1. Do not start any new development until all tests pass
    2. Test first
10. Express requirements through scenarios

Source: Bertrand Meyer: Agile Methods – The Good, the Hype and the Ugly (2014)

Food for Thought

- Meyer's preceding critical review is not established "textbook" knowledge, but a contribution to an ongoing academic and professional discussion.
- Some of his criticism will weigh differently depending on the type of project, the composition of the team, etc.
- It is provided here to spark and encourage your own critical thinking about the software engineering practices you have learned about.
- **How do you perceive the agile practices?**
  - Which ones would you adopt?
  - Which ones would you modify?
  - Why?

Grady Booch: The Future of Software Engineering
(A Retrospective and Outlook on Our Discipline)



International Conference on Software Engineering 2015 Keynote

- No matter what future we may envision, it relies on software that has not yet been written. The nature of the systems we build continues to change, and as they weave themselves into our lives, we must attend not only to the technical elements of software development, we must also attend to human needs. This talk looks at the history of software engineering and offers some grand challenges for the future.

- **Grady Booch** is Chief Scientist for Software Engineering at IBM Research. Having originated the term and the practice of object-oriented design, he is best known for his work in advancing the fields of software engineering and software architecture. He is a co-author of the Unified Modeling Language (UML), a founding member of the Agile Alliance and the Hillside Group, has been awarded the Lovelace Medal and given the Turing Lecture for the BCS.

https://youtu.be/h1TGJJ-F-fE?t=7m30s

Grading Recap and Final Exam

Recap: Assignment 4: Schedule and Deliverables

- This afternoon, **demonstrate** and **explain** your product to your classmates in a 10- to 15-minute-presentation:
  1. **Product:** What does your product do? Demonstrate the key features of your system.
  2. **Architecture:** How does your product work? Explain architecture & key design decisions.
  3. **Process:** How did you build the product? Relate and interpret challenges you faced.

- On **Sun 14 Apr**, submit your **final product** in Ugla, including:
  - Complete source code and installation instructions
  - Slides of your final presentation

- Your product does not need to satisfy all the criteria you listed in your initial requirements document, but the key features should work.

Recap: Assignment 4: Final Presentation Format

- All teams will present their work in the common area on the ground floor of Tæknigarður.

- Each team will get a table to set up a laptop and/or mobile device(s) for an hour to demonstrate their product to other teams and tutors visiting their table
  - Teams 1-10: 13:20-14:20 (setup begins 13:00)
  - Teams 11-20: 14:25-15:25
  - Teams 21-30: 15:30-16:30

No power cords available – ensure your devices are charged!

- During their timeslots, teams repeat their 10- to 15-minute presentation 3-4 times as different teams are visiting their table.

- Please attend the other timeslots as well to visit other teams' tables and learn about their product ideas, technologies used, and challenges they experienced.

Recap: Assignment 4: Grading

- Your product will be graded during the presentation attended by your tutor.

- Team grade refers to the visible parts of the presentation
  - i.e. information on slides, impression the product makes, etc. – Criteria:
    - Final product implements key features, and is running smoothly (75%)
    - Critical retrospective given of chosen process, architecture and technology (25%)

- Presenter grade refers to the audible parts of the presentation
  - Criteria: how well the presenter explains things, answers questions etc.
  - Presentation to tutor must be given by team member who has not presented any assignments yet (the other presentations may be given by other team members)
  - If all team members have gotten a presentation grade already, the presentation quality will be reflected in the team grade.

Recap: Overall Grading Scheme

- All contributing factors are graded on a scale of 0…11
  - Grading criteria for assignment deliverables
  - Individual presentation quality
  - Questions on in-class quizzes
  - Questions on final exam
- All factors are averaged using the published weights, without rounding
  - Exceptional performance (11) on some factors can outweigh lower performance elsewhere
- Resulting final grade is rounded to nearest half point
  - In case of a passed exam, final grade is capped at 10.0
  - In case of a failed exam, final grade is capped at 4.5

- Need a passing project grade to be allowed to final exam
  - If project grade is not sufficient, need to do a project again next year
- Need a passing project and exam grade to pass the course
  - Failed exam can be re-taken during the resit period, and if failed again, the year after
  - No need (and not possible) to redo a passed project
    - Project grade remains valid for only one year though

Recap: Project Grading

- The project grade depends on the **deliverables** submitted and the **presentation** given for each assignment.
  - Grading criteria will be published together with assignment.
- All team members receive same grade for **deliverables** submitted for an assignment
  - Each assignment weighs **20%** of project grade
- Over the course of the semester, each team member must lead the **presentation** of at least one assignment to tutor
  - Focus: Don't just tell us what you did, but *why* you decided to do it this way.
  - The presenting team member receives an individual grade for their presentation ("5th assignment", weighs **20%** of project grade)
    - If someone gives several presentations, the best presentation grade counts
- The resulting project grade weighs 30-70% of the final course grade.

Recap: Grade Weights

- Peer assessment of team mates' contributions to project
  - At the end of the semester, all team members assess how much each of their team mates contributed to the project.
  - Contribution votes are normalized to obtain each team member's contribution factor.

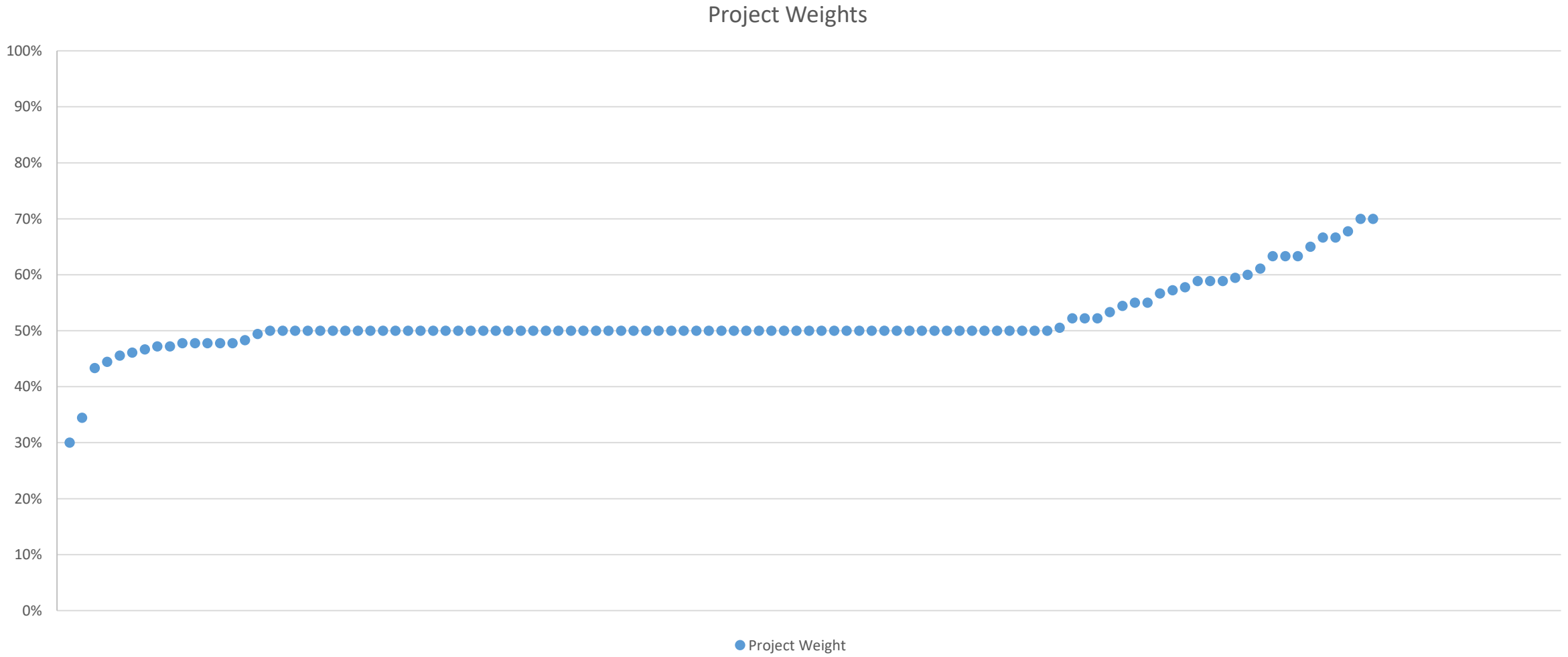
- Depending on team members' project contributions, the weight of their project and final exam grades will be individually adjusted between 30% and 70%:
  - Below-average contribution → lower weight of project grade, higher weight of exam grade
  - Average contribution → equal weight of project and exam grade (50:50)
  - Above-average contribution → higher weight of project grade, lower weight of exam grade

- In rare cases where the above rules would punish someone who contributed above average, or reward someone who contributed below average, the traditional 50:50 weight distribution is used instead.

Peer Assessment

- Each team member can assess the contribution of each of their team mates:
  - +++ : contributed **much more** than others
  - ++ : contributed **more** than others
  - + : contributed **slightly more** than others
  - o : contributed **same** as others
  - - : contributed **slightly less** than others
  - -- : contributed **less** than others
  - --- : contributed **much less** than others
- Assessments for each team member will be averaged, and the assessments will be normalized across all team members
  - ➢ if you and your team mates all rate each other as “+++”, your normalized contribution will be “o”
- Ratings should reflect a fair assessment of your own and your team mates’ contribution.
  - Ultimate decision of how assessments are counted remains with tutors.
- Submit your ratings under the “Peer Assessment” assignment in Ugla by **Sun 14 Apr**.

Example: A Past Course's Project Weights based on Peer Assessment



Recap: Final Exam

- **Date & Time:** Mon 29 Apr, 13:30-16:30
- **Focus:** Understanding of software engineering concepts and methods
- **Scope:** Lecture slides (i.e. contents of Námsefni folder)
  - Note: The soundtrack is relevant!
- **Style:** Written exam
  - Write into given spaces on exam sheets
  - Mark exam only with your exam number, *not* your name
- **Weight:** 30-70% of course grade
- **Tools:**
  - One sheet of handwritten material allowed
    - i.e. blank A4 sheet with only your own ink (double-sided use ok)
    - no photocopied notes or printed material
  - Dictionary allowed (in book form)
  - No electronic devices allowed
- **Questions:**
  - 10-12 questions weighing 5-12% each
  - No optional questions (except quiz result)
  - But answers that exceed expectations can make up for deficiencies elsewhere
- **Answers:**
  - in English, in your own words
  - short paragraphs of whole sentences
  - possibly small models

Recap: Optional In-Class Quizzes

- To encourage class attendance: Small quizzes in most lectures
  - Solved and handed in during class
  - Graded on usual scale, with 0 for any unanswered questions, 5…11 for answered questions
    - 2 worst quiz grades (e.g. from skipped classes) will be ignored

- Grades of all in-class quizzes will be averaged into one final **quiz grade**
- Quiz grade can improve your final exam grade:
  - All the normal final exam questions add up to 100% (as usual)
  - Quiz grade will be counted as an optional additional final exam question worth 7.5%

➢ If you don't participate in any quizzes, you can still get top marks on the exam

➢ If you do participate in quizzes, the quiz grade can improve your exam grade by up to 11 on a 7.5% question, and thereby make up for deficiencies elsewhere

Sample Exam Questions

- The following review questions are representative of the types of questions that could be on an exam.
- The answers to some of the following review questions can be found immediately on the lecture slides.
  - Caution: An open-book exam will obviously contain very few of these!
- Most questions require more individual reasoning and/or application of learned knowledge.
  - Be precise, justify your answers.

- **Read exam questions carefully**
  - Are you asked to explain or give an example or both?
  - Are you asked to argue for or discuss an issue?
  - Are you asked to explain or discuss an issue, or both?
  - How many concepts are you asked to consider?

- **Answer precisely**
  - i.e. brief, focused, comprehensive

Types of Questions

- **“Explain…”**
  - Give an explanation of what something is or how something works in general.
  - Note: A concrete example is not a substitute for a general explanation.
- **“Argue…”**
  - Make up your mind about a certain issue.
  - Present arguments for your opinion on the issue.
- **“Discuss…”**
  - Consider both sides of an issue.
  - Present arguments for both sides.
  - You can state your own opinion, but should present counter-arguments as well.
- **“Suggest…”**
  - Come up with a solution for an issue.
  - Briefly explain what should be done.
- **“Give an example…”**
  - Give a brief example that illustrates the considered concept.
  - When giving examples for several distinct concepts, make sure the examples highlight the distinguishing characteristics.
- **“Point out issues…”**
  - Examine a given artifact, and explain what is wrong with it.
  - Don’t just say *what* is wrong, but *why* it is.
- **“Draw…”**
  - Draw a UML diagram reflecting the relevant aspects of the given scenario.
  - Pay attention to proper syntax!
- **“Implement…”**
  - Write/modify Java code. For modifications, place question sheet into answer book.
  - Pay attention to proper syntax!

Sample Questions: Software Requirements

- Argue why neither “Let’s just start coding and see what the customer says” nor “Let’s get the complete and precise requirements from the customer first” are effective approaches to requirements engineering.
- Explain the difference between needs, features and requirements.
- Initial versions of user stories that a team comes up with are often too broad. Explain two patterns for refining user stories by splitting them into more stories.
- Argue how much emphasis should be placed on requirements engineering in plan-driven projects and in agile projects.
- Discuss whether a traditional product manager could fill the role of an agile product owner.

Sample Questions: Software Estimation

- Argue why single-point estimates can be misunderstood, and how estimates should be phrased instead.
- Explain how much divergence between an estimate and a target can typically be coped with through project management, and argue what should be done when the divergence is bigger.
- For the following scenarios, calculate the projected effort, or argue why a projection is not possible: [...]
- Assume that for a certain feature, an expert came up with a best-case estimate of … person-days (PD), a most-likely-case estimate of … PD and a worst-case estimate of … PD. Calculate the expected-case estimate according to the basic PERT formula.

Sample Questions: Software Estimation

- Argue whether it is a good sign if about half of your estimates are above and about half are below the actual values.
- Explain the differences between Wideband Delphi and Planning Poker.
- Argue whether you would use the T-shirt sizing method for sprint planning.

Sample Questions: Android Development

- Explain the relationship between an activity and a layout.
- Explain why anonymous inner classes are frequently used in Android development.
- Explain how you can avoid losing an activity's state when the device is rotated.
- Explain the difference between explicit and implicit intents.
- Explain why the following diagram does not show an elegant Android design, and draw an improved version.

Sample Questions: Android Development

- Explain for what kinds of data you would use a) a **`SharedPreferences`** file, b) a binary file, and c) an SQLite database in your app's internal storage directory.
- Explain the difference between internal and external file storage.
- Explain the role of **`@DAO`** objects when using the Room persistence framework.
- Explain the difference between **`AsyncTask`**s and **`HandlerThread`**s.

Sample Questions: Software Architecture

- Explain the difference between the module structure and the allocation structure of a software architecture.
- Name two types of design decisions shaping the module structure of a software architecture, and give an example for each type of decision.
- Assume you want to improve the performance of a software system. Give two examples of design considerations that would shape your technology choices.

Sample Questions: Software Engineering Practices

- Consider Meyer's classification of agile practices into "the ugly", "the indifferent", "the good" and "the brilliant". Pick one practice whose classification you agree with and one that you disagree with, and argue why you agree or disagree with them.

- Pick a software engineering practice that you learned about in HBV401G, HBV501G or HBV601G. Argue under which conditions you believe it would work well, and suggest how it could be modified for different conditions.

Wrap-up

Special Thanks

Andri Valur Guðjohnsen

Daníel Páll Jóhannsson

Sigurður Gauti Samúelsson

Kennslukönnun

Evaluate this course on Ugla!

(until 24 April)

Long-Term Feedback

- Over the course of your future academic and professional career, you'll be exposed to challenges
  - that these courses prepared you for
  - that these courses didn't prepare you for

- I'd appreciate if you could let me hear about these in 1, 2, 3… years so we can keep tailoring the curriculum better to actual demands.
  - What could you use?
  - What couldn't you use?
  - What do you wish you had been taught?
  - What do you wish you had taken more seriously?

- Contact me at **book@hi.is** – I look forward to keep hearing from you!

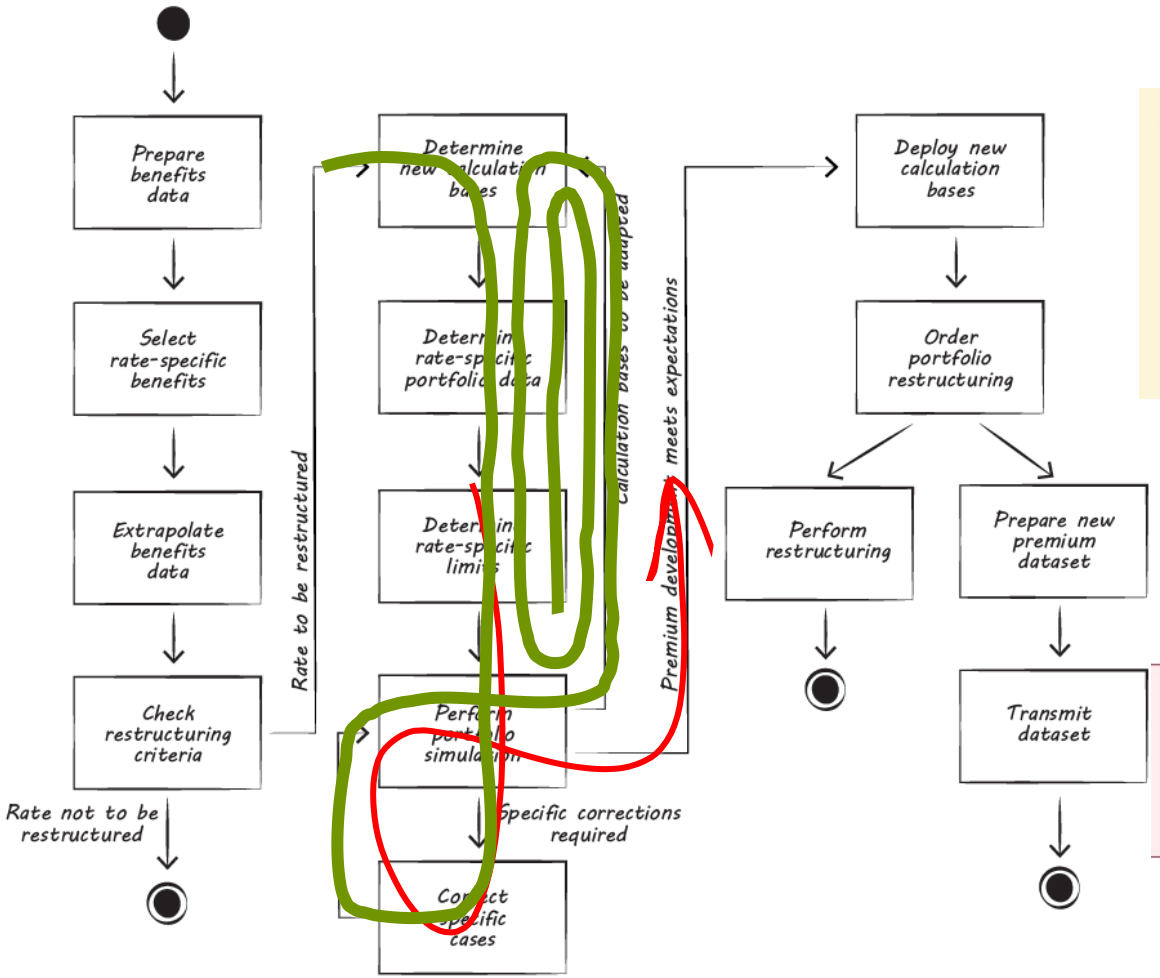
Teaching Assistants Wanted

- I am looking for TAs for next semester's course:
  - **HBV501G Software Project 1**
- If you are interested or know someone who is, please contact me at **book@hi.is.**

- **Tasks**
  - Advising student teams
  - Scoring assignments
- **Opportunities**
  - Help to shape the course
  - Brush up your CV
- **Requirements**
  - Successful completion of respective course
  - Ability to advise and evaluate student teams

Recap: Sketch-based Software Engineering

in cooperation with
UNIVERSITÄT DUISBURG ESSEN
UCI University of California, Irvine
funded by rannís Icelandic Research Fund



Goal: Enable developers to use sketching

- **not just to help them think**
  about software engineering activities
- **but as an interaction modality**
  to perform and complete activities directly

```
        <constraint
            firstAttribute="trailing"
            secondItem="LFs-rG-Cv6"
            secondAttribute="trailing"
            id="w6V-ZV-c3M"/>
    </constraints>
</view>
<connections>
    <outlet property="displaySubtitle"
        destination="PPq-nc-dK0" id="p3z-
        Mf-llw"/>
    <outlet property="bottomView"
        destination="7Wa-hI-bwz" id="wcM-
        eJ-2e5"/>
    <outlet property="finishSuiDuButton"
        destination="dWz-Xu-8U4" id="ItA-
        CN-vF0"/>
    <outlet property="mainView"
        destination="LFs-rG-Cv6" id="OZC-
        zg-yYF"/>
    <outlet property="numberLabel"
        destination="zog-c3-fbX" id="wJq-
        Xf-MiY"/>
    <outlet property="weiTiaoView"
        destination="58Q-XU-7cJ" id="fre-
        Wb-wsR"/>
</connections>
</viewController>
<placeholder
```

```
188
189
190
191
192
193
194
195
196
197
198
199
```

```
        <constraint firstAttribute=
            "trailing" secondItem="LFs-rG-
            Cv6"
            secondAttribute="trailing"
            id="w6V-ZV-c3M"/>
    </constraints>
</view>
<connections>
    <outlet property="bottomView"
        destination="7Wa-hI-bwz" id="wcM-
        eJ-2e5"/>
    <outlet property="finishSuiDuButton"
        destination="dWz-Xu-8U4" id="ItA-
        CN-vF0"/>
    <outlet property="mainView"
        destination="LFs-rG-Cv6" id="OZC-
        zg-yYF"/>
    <outlet property="numberLabel"
        destination="zog-c3-fbX" id="wJq-
        Xf-MiY"/>
    <outlet property="weiTiaoView"
        destination="58Q-XU-7cJ" id="fre-
        Wb-wsR"/>
</connections>
</viewController>
<placeholder placeholderIdentifier=
    "IBFirstResponder" id="bFU-wk-q91"
    userLabel="First Responder"
    sceneMemberID="firstResponder"/>
```

Example: Sketch-based Test Case Specification

Example: Sketch-based Code Merging

60 ECTS HBV/TÖL MSc Project Topics in Sketch-based Software Engineering

in cooperation with UNIVERSITÄT DUISBURG ESSEN, UCI University of California, Irvine

funded by rannís Icelandic Research Fund

- *Work with a PhD student on the following research goals:*

- **Developing sketch-based micro-notations** that can express developers' intentions for different purposes
  - i.e. sketching gestures for testing, code merging, refactoring etc.
- **Turning software artefacts into base layers** that can be sketched on (models, source code, running applications, etc.)
  - Semantic challenge: To correctly interpret user intentions, tools need to understand semantics of the user's sketches and elements of the base layer (i.e. source code)
  - Technical challenge: Reinvention of current stand-alone sketching tools' architecture, as sketching will turn into one interaction layer among others in general IDEs
- **Evaluating** how the sketch-based interaction can be integrated seamlessly with the developers' thoughts and ways of approaching software engineering tasks

- *If you are interested, please send your CV and grade transcript to **book@hi.is**.*

Keep in Touch!

- For feedback, questions, projects, theses, events, guest talks, collaboration…

book@hi.is

http://is.linkedin.com/in/matthiasbook

http://www.facebook.com/matthiasbook

http://www.twitter.com/matthiasbook

Takk fyrir
og gangi þér vel!

book@hi.is

HÁSKÓLI ÍSLANDS
VERKFRÆÐI- OG NÁTTÚRUVÍSINDASVIÐ
IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI-
OG TÖLVUNARFRÆÐIDEILD