CS4505 Labwork

Image credit: Corné Bastiaansen www.rijksvastgoedbedrijf.nl

CS4505 Software Architecture, TU Delft, 2024/2025

Arie van Deursen & Diomidis Spinellis

Course Objectives: You'll Learn How To:

1. Structure a system's problem space
2. Architect a software solution that meets stakeholder needs
3. Manage evolving needs, keeping architecture aligned
4. Understand and assess architectures of existing software systems
5. Use proof-of-concepts for architectural decision making
6. Communicate architectural decisions
7. Fulfill the role of an architect

Image credit: Denis Andernach

Labwork Key Elements

- Work in teams of four
- Develop architecture for system of choice
  - Report R1: Analysis of problem domain
  - Report R2: Proposal for architecture
  - Proof-of-concept (PoC): code + README + docs
- Grow reports + PoC in three iterations (W3, W6, W9)
- Evaluate architectures from peer teams
- Embrace freedom to maximize your learning

Labwork Overview

	Problem Analysis			Architecture Development			Refinement and Consolidation		
	W1	W2	W3	W4	W5	W6	W7	W8	W9
Team formation and system selection						R1 final		Presentation days	
					R2 + PoC 80% submitted				R2 + PoC final
		R1: Problem analysis (80%)					Peer review		
				R1 refinement					

Team Composition Logistics

- Teams of (exactly!) four
- Target diversity in team to optimize **peer learning**
  - Interests in application domain, technical expertise, BSc, cultural background, writing fluency, presentation experience, ...
  - Avoid falling back to groups you've worked in before
- Teams created in Brightspace -> Collaboration -> Groups
- Teams finalized *in first week!*
- For each team, one TA will be their *coach*

LO6 communication
LO7, role

Week 1: Pick a System!

- (Modern) alternative to existing real-world system with concrete needs
- You / your team should be passionate about the system
- Diverse collection of stakeholders, challenging *context*
- Non-trivial / interesting technical challenges

System Selection Logistics

- Your team can propose a system on BrightSpace
  - BrightSpace -> Discussions -> Claim your team and system
- Each team must have *different* system
  - OK to work on clearly distinguishable subsystems of one larger system
  - Two teams must agree on distribution
  - Define / build interaction between the two proof-of-concepts
- Do *not* use existing open source system – too much bias on code
- System selected must be approved by teachers / TA.
- Teams must select a system in *first week!*

Suggested Theme: “Government goes Digital”

- *What if government could start from scratch?*
- Select a government service suitable for digitalization
  - Replacement of existing IT system?
  - Or brand-new service not yet imagined?
- EU, national, regional or municipal
- Netherlands or your favorite country
- (This is optional – non-govt systems also OK)

Example Government Systems?

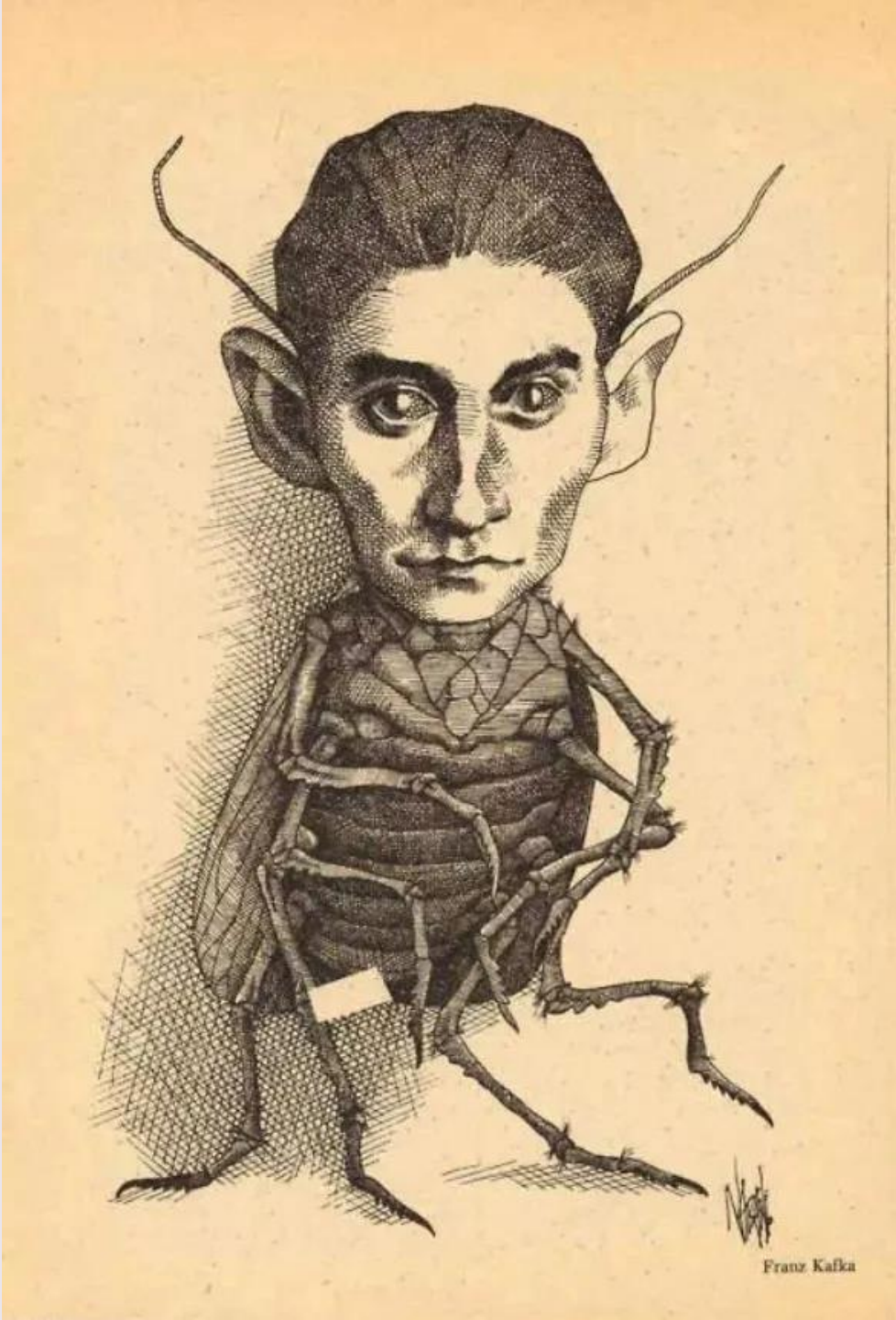
- Rethinking VAT (”BTW”)
- Social benefits
- Pensions
- Public health insurance
- (Integrated) public transport
- Air traffic control
- Transparent (open) govt
- Immigration
- Defense equipment management
- Police
- The judiciary
- Prison system
- Higher education
- Energy (transition)
- Cadastre
- Financial crime detection

Naam	Ministerie	Onderwerp	Projectfase	Projectduur	Totale kosten
European Rail Traffic Management System (ERTMS)	Infrastructuur en Waterstaat	Verkeer en vervoer	In uitvoering	11,6 jaar	€ 2.696,50 mln.
Programma iCAS	Infrastructuur en Waterstaat	Verkeer en vervoer	In uitvoering	10,1 jaar	€ 274,00 mln.
Programma uitrol iWKS	Infrastructuur en Waterstaat	Verkeer en vervoer	In uitvoering	7,1 jaar	€ 250,00 mln.
Digitaal stelsel omgevingswet (DSO)	Volkshuisvesting en Ruimtelijke Ordening	Overheid en democratie	In uitvoering	9 jaar	€ 177,50 mln.
Maritiem Operatiecentrum Kustwacht (MOC KW)	Defensie	Recht, veiligheid en defensie	In uitvoering	10 jaar	€ 155,00 mln.
CHARM	Infrastructuur en Waterstaat	Verkeer en vervoer	In uitvoering	[illegible]	[illegible]
Verwerving Datacenter	Sociale Zaken en Werkgelegenheid	Belastingen, uitkeringen en toeslagen	In uitvoering	7,1 jaar	€ 120,63 mln.

Example systems from https://www.rijksictdashboard.nl

Is Government IT Hard?

- Requirements:
  - Accessibility for all citizens
  - Strong requirements on privacy, security, integrity, …
  - Long life-times (possibly multiple decades)
- Process:
  - Limited “Digital Thinking” in law/policy making
  - Benefits not financial but *societal*,
    and often hard to measure
  - Complex governance of government IT systems
  - Competition for IT talent with industry



Week 2: Make a Plan!

- Create plan (in e.g. powerpoint) and present it to your TA
  - Upload pdf in Brightspace as assignment
- Selected system, solution direction, PoC ideas
- Logistics:
  - Division of work among team members
  - Meeting frequency
  - Communication channels (mattermost!)
- Personal and team objectives
  - Relevant knowledge you already have you may wish to apply
  - New knowledge and skills you would like to develop in this course

Weeks 1-3: 80% Version of Report R1

- Architecturally relevant requirements:
  - Scenarios, quality attributes
  - External data and services constraints
- Stakeholder analysis
- Sources of ambiguity
- Mission, vision, roadmap
- ~ 3000 words – more details in lecture 2
- Deadline end of week 3!

LO1, problem analysis

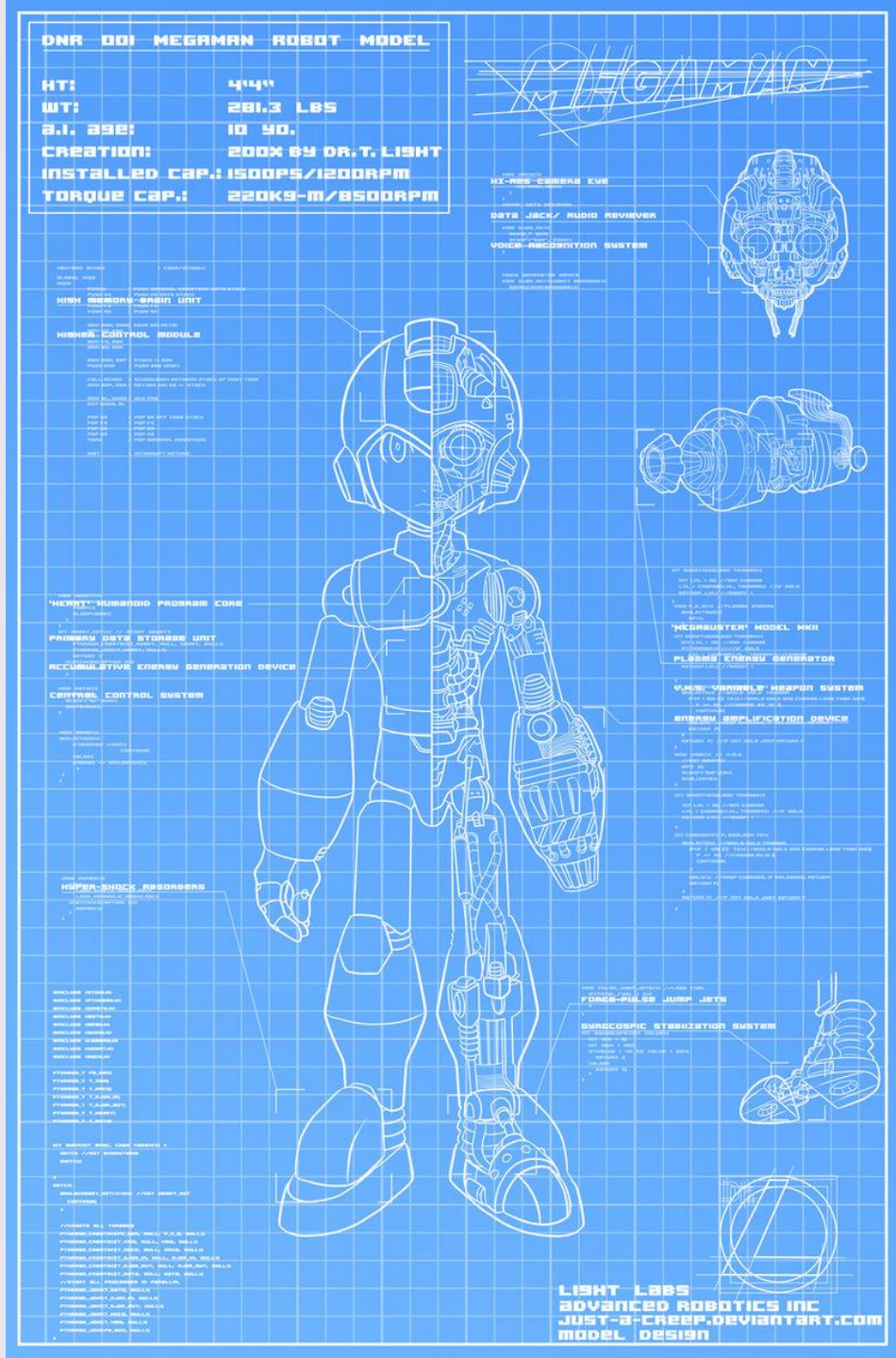
Week 4: Domain Refinement Meeting

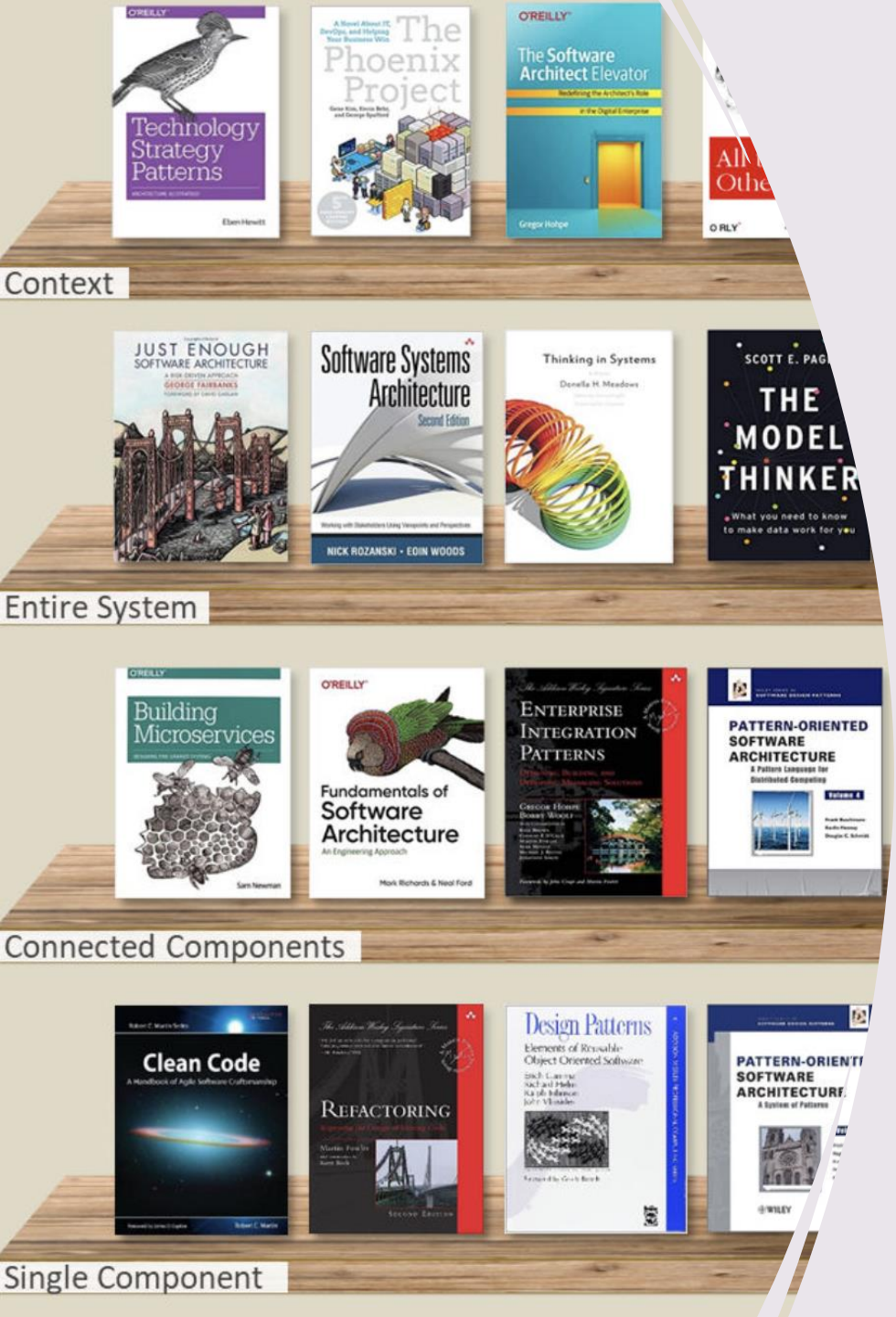
- Given 80% R1 / written domain analysis:
- Week 4: Meeting with TA *and another team*
  - Teams present their results to each other
  - Teams suggest changes to each other's domain / requirements
  - Teams discuss how discuss how system could integrate changes
  - TA facilitates
- Afterwards: Refine R1 report to reflect changes (= ambiguity) and their architectural impact

LO3, evolution
LO4, assessment

Weeks 4-6 & 7-9
R2: Develop Architecture

- Key scenarios and quality attributes
- Architectural views: modules, data, concurrency, components & connectors, …
- Architectural styles: pipes and filter, model-view-controller, (micro)-services, …
- (UML, SysML) models
- External interfaces; (open source) libraries
- Quality assurance and software testing
- C4/C5, arc42 guidelines & templates





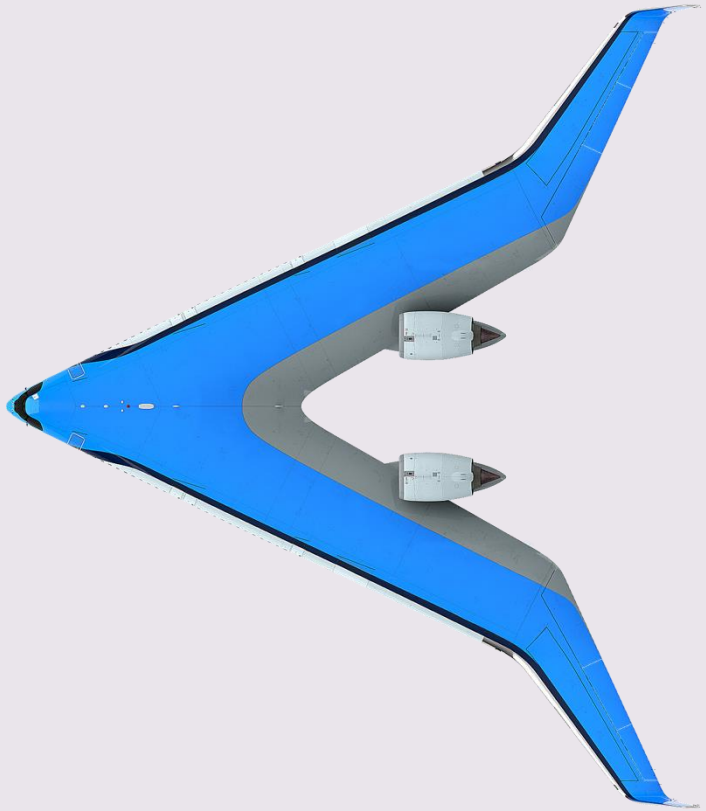
R2: Deepening your Knowledge

- For system, pick relevant architectural topic from literature you want to explore deeper
- Use lectures as starting point
- Find and apply additional books / papers / blogs / resources
- Describe (cite) in your report

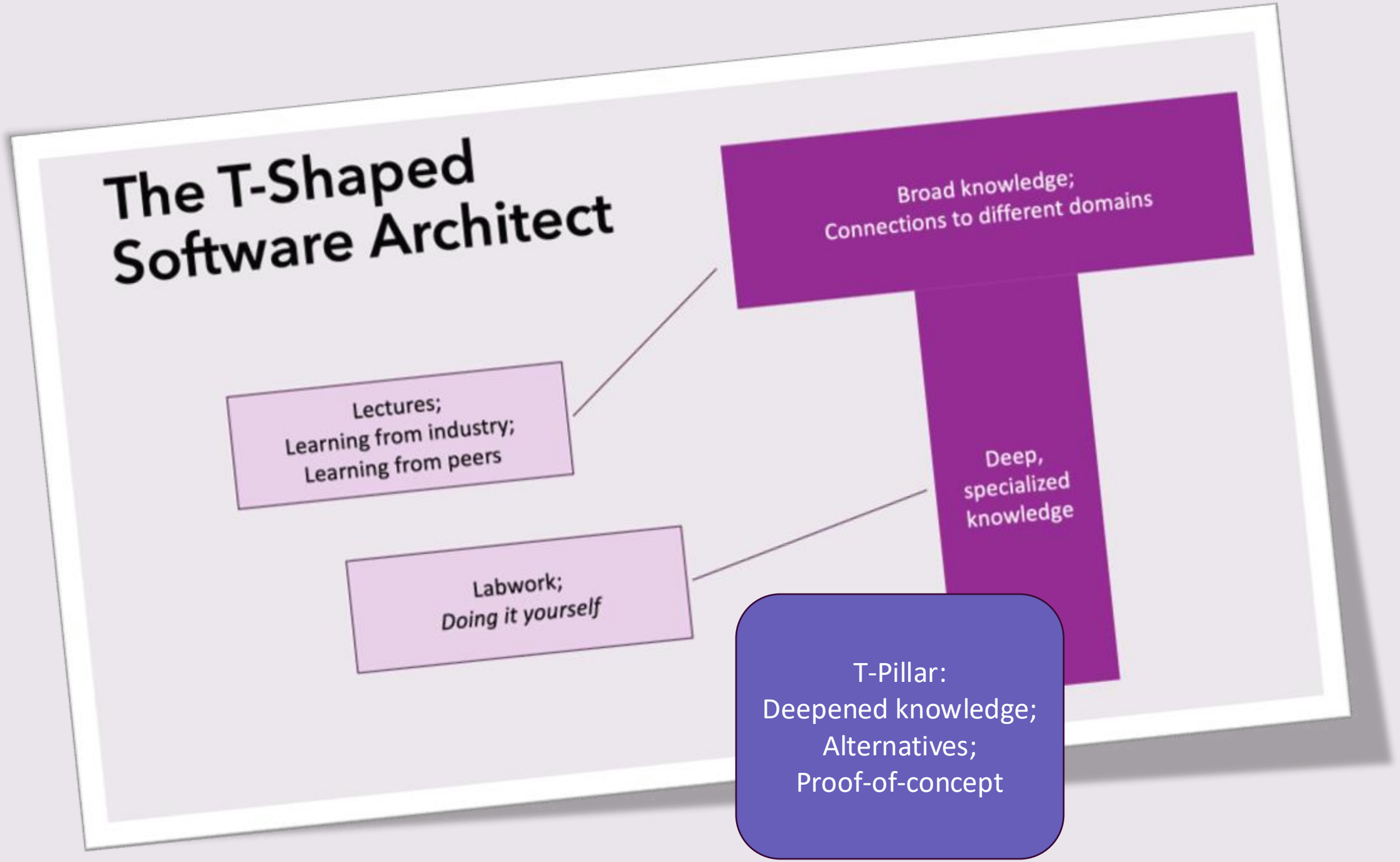
R2: *Alternatives*

- Architecture is about *decisions*
- A decision is a choice among *alternatives*
- In R2, for one (or more) key decisions, formulate and elaborate:
  - Four serious options
  - An analysis of trade-offs
  - A motivation for picking one as favorite
- [Build on deepened knowledge]

Week 4-6, 7-9:
Build Proof of Concept



- Gitlab repository per team
- Clear argument how proposed proof-of-concept will help take / support architectural decision(s)
- Proof-of-concept enables conducting experiments (measurements, assessment of API usability, …)
- Critical analysis of experimental results, using relevant diagrams
- Sound software engineering best practices (git, testability, documentation, continuous integration, release planning, ...)
- README + relevant docs explaining the PoC.


The T-Shaped Software Architect
Broad knowledge;
Connections to different domains
Deep, specialized knowledge
Lectures;
Learning from industry;
Learning from peers
Labwork;
Doing it yourself
T-Pillar:
Deepened knowledge;
Alternatives;
Proof-of-concept

Week 7: Reports + PoC for Review

- R1 + R2 reports and PoC will be subjected to peer review
  - Problem analysis (R1 final report, 100% finished)
  - Architectural development report (R2 solid draft, 80% version)
  - PoC up and running (README, 80% version, ongoing)
- Details that are still in progress should be clearly marked
- PDFs submitted to TU Delft “peer” system
- Peer review by students from other teams
  - Questionnaire / rubric in Peer.

LO4, Architecture Evaluation

Week 9: Presentation Days!

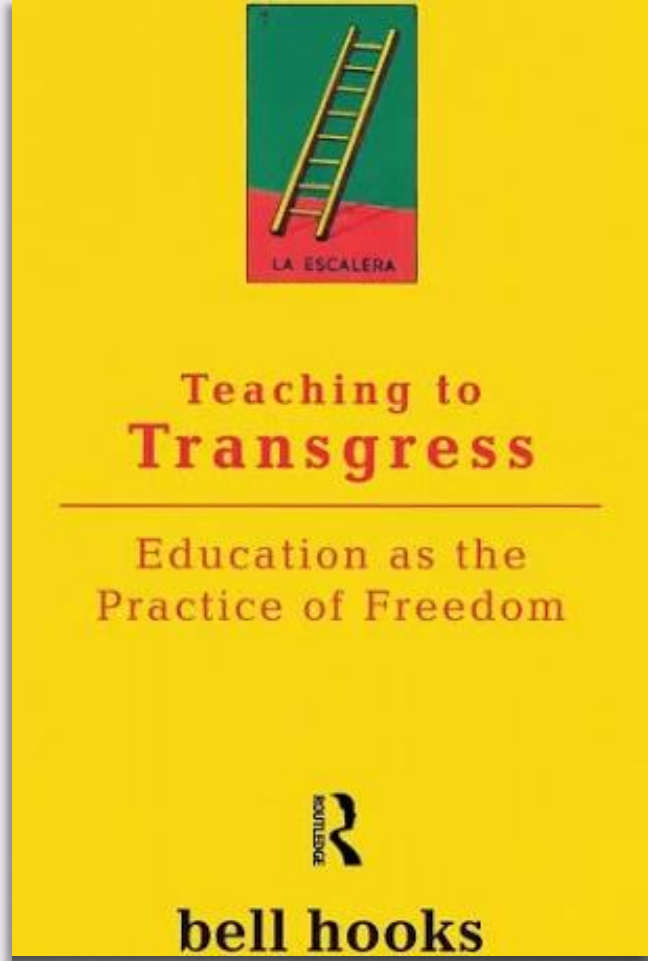
- 15 minute presentation + 5 min Q&A
- Includes demonstration of proof-of-concept
- Everyone present in person
  - (health-related exceptions via academic counselor granted)
- Two mornings, two parallel sessions, 4 sessions in total
- Each session has around 10-15 team presentations
- Audience feedback via online form

Week 9:
Final PoC & Architectural Report

- Digest all feedback from peer reviews and presentations
- Submit final (100%) versions of
  - architectural document R2
  - PoC + report R3
- PoC submission: merged commit + gitlab tag before deadline
- Report submission:
  - Merged commit before deadline + pdfs uploaded to BS

Open Learning; Open Writing

- Intent: *Learn from each other*
- Write for your fellow students
  - Better still: Write for the outside world!
- Make it as easy as possible to enjoy your writing
  - Write reports in the format of a blog post!
- Collectively, we work towards a public web site showing our joint creativity



localhost:1313

Delft Students on Software Architecture

A blog on the architecture of (government) software systems written by students from Delft University of Technology

Nestorix Proof of Concept

Nestorix — January 29, 2024

We provide a proof of concept demonstrating the product line architecture third party pension funds can use to enrich the user experience of their customers. We provide four alternatives architectures with different binding times (static generation, dynamic loading, runtime user configurable) and assess their pros and cons for different usage scenarios (web, mobile) We illustrate our proof of concept using variability models, and API documentation.

Read more »

Nestorix Proposed Architecture

Nestorix — January 28, 2024

The backbone of Nestorix will be a cloud-native micro-serivces architecture, with separate micro-services for, e.g., premium computations, payment computations, and scenario analysis. The web portal will make use of a standard model-view-controller architecture, relying on the micro-services for computations. Scaleability will be achieved through an event-based architecture, where payments are pushed to queues.

Read more »

Nestorix: Rethinking Pension Provisioning

Nestorix — January 27, 2024

In this post, we propose a radical rethinking of IT support for pension systems, through a new system called Nestorix. The primary functionality of Nestorix is giving citizens authoritive information about the status of their pension – today, tomorrow, and many years from now. The system connects to third party pension fund suppliers, to obtain accurate pension data. Furthermore, it allows third party suppliers to specialize the Nestorix user experience so that it includes unique features of the pensions offered.

Read more »

localhost:1313/projects/pensions/

NESTORIX

This is an example project, for a hypothetical government system to manage pensions, called Nestorix.

Image source: https://asterix.com

PROJECT SITE | PROJECT SOURCE

AUTHORS

Nestorix Proof of Concept

We provide a proof of concept demonstrating
pension funds can use to enrich the user exp
alternatives architectures with different bind
loading, runtime user configurable) and asse
scenarios (web, mobile) We illustrate our pr
API documentation.

Nestorix Proposed Architecture

The backbone of Nestorix will be a cloud
separate micro-services for, e.g., premiu
scenario analysis. The web portal will ma
architecture, relying on the micro-servic
achieved through an event-based archi

localhost:1313/projects/pensions/posts/2024-01-27-problem-analysis/

ESOSA 2024

Nestorix: Rethinking Pension Provisioning

Nestorix — January 27, 2024

Authors
Arie van Deursen

Print this article

In this post, we propose a radical rethinking of IT support for pension systems, through a new system called Nestorix. The primary functionality of Nestorix is giving citizens authoritive information about the status of their pension – today, tomorrow, and many years from now. The system connects to third party pension fund suppliers, to obtain accurate pension data. Furthermore, it allows third party suppliers to specialize the Nestorix user experience so that it includes unique features of the pensions offered. The initial market place for this system is The Netherlands, but the ultimate vision is to make this a European pension system, supporting the free movement of people and their pensions through Europe.

Pension Systems

Problem analysis:

System vision

Main function delivering external value

Monthly payments of pensions. Single portal for people to manage their pensions.

Start (birthday, early retirement), stop (passing away, moving)

Key Scenarios

Scenarios that define the architecture

Quality Attributes

Scalable to 10 million registered pensionados, correct payments each month

External Dependencies

Systems that we assume that are live

Stakeholder Analysis

Collaborative Writing

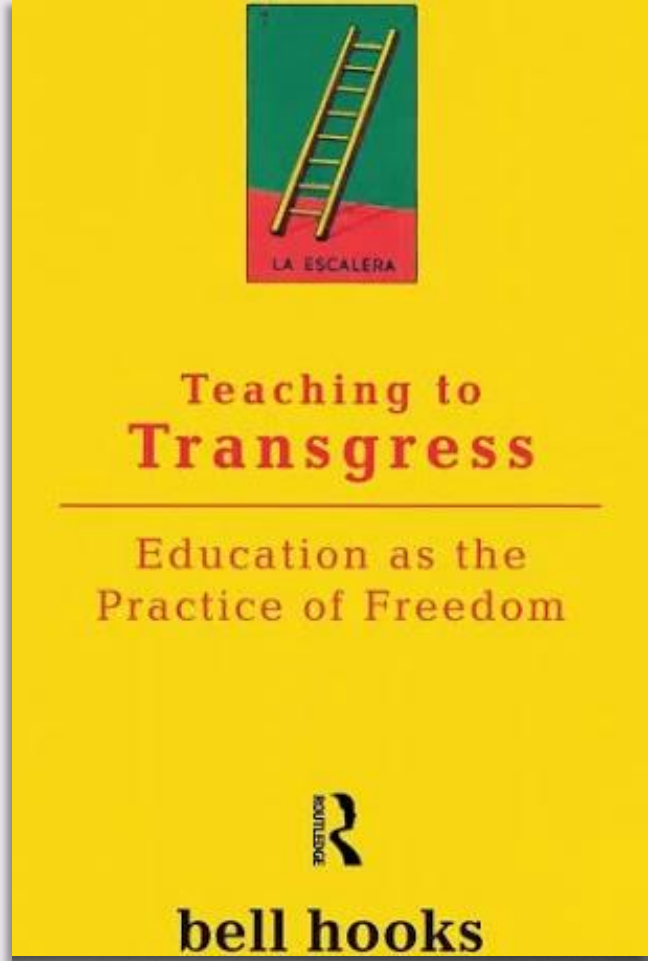
- Markdown: GitLab Hugo + GitLab Pages
- For all reports: Single gitlab repository for whole course!
  - README has markdown suggestions
  - Local build via Docker
  - Pages deployment via gitlab CI
- Per team: One subfolder with overview and two reports
  - *Only make changes to your own subfolder*
  - Push to main only via merge requests

Report Logistics

- Word limits for final versions:
  - R1: max 3000 words; R2: max 6000 words
  - Appendices permitted, and don’t count to word limit.
  - Reports must be readable *without consulting appendices*
  - Appendices may be ignored during grading
- Commits merged to main branch before deadline
- Pdfs uploaded to Brightspace assignments

Open Learning;
Open Communication

- Intent: *Learn from each other*
- **All course communication in Mattermost**
  - Per team: dedicated channel
  - Avoid whatsapp, mail, discord, …!
- **All writing in GitLab (markdown)**
  - Git commits
  - Discussion in issues / merge requests

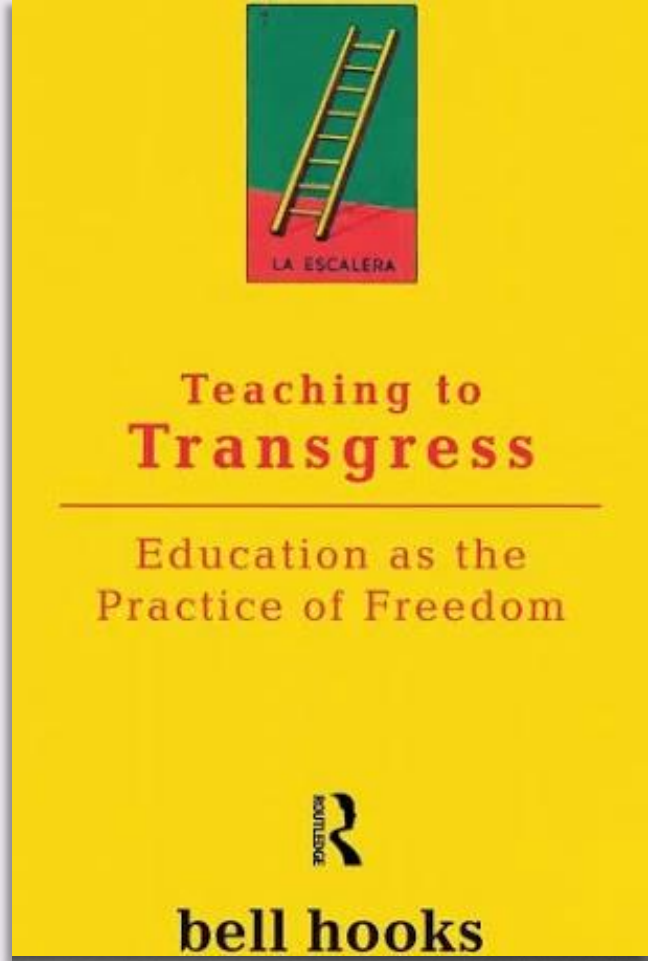


Mattermost Logistics

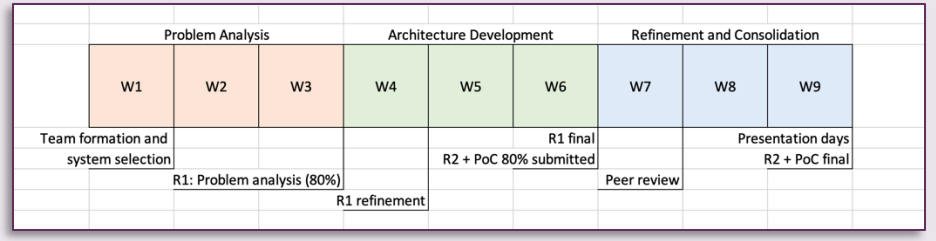
- Town Hall: Low traffic, important messages from teachers only
- Q-and-A: Open for questions
- Off-Topic: Random noise
- Per team: Own channel, open to all (including teachers)
  - Use as primary means of communication within team
  - Traceable to teachers

Open Learning; Accountability

- Course gives you lots of freedom:
  - *You need to show us how you used this freedom*
- Mattermost communication
- Gitlab git history, issues, merge requests, roadmaps, milestones, …
- Participation in bi-weekly meetings with TA coaches
- Expected effort: 5 * 28h = 140h = 15h / week

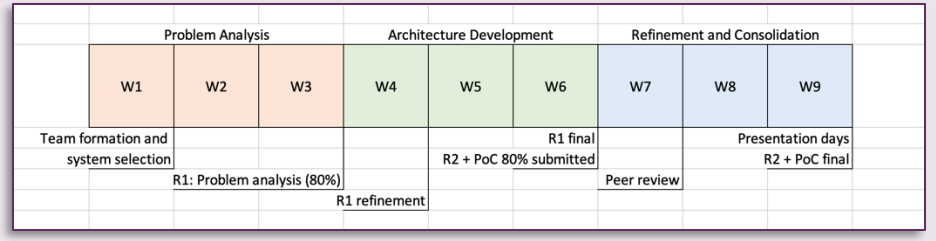


Labwork By Week (1-2)



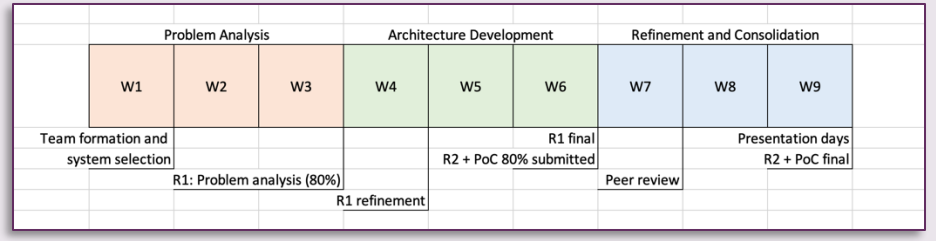
- Week 1:
  - Team formation & System selection.
  - Deadline: Fri Sept 6, 17:00
- Week 2:
  - Teams get assigned TA (Monday)
  - Teams get (and use) Mattermost channel
  - Teams make first commits to gitlab blog repo + their gitlab PoC repo
  - Teams have 1st appointment with their TA (30 minutes)
  - Each team presents their plans (on slides) to TA –
    - upload plan to BrightSpace by end of week (Monday Sept 16, 17:00))
  - Schedule appointments with TA in W4 (refinement), W6 and optionally W8

Labwork by Week (3 & 4)



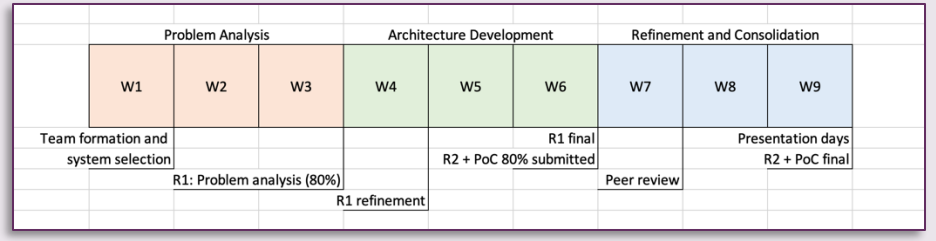
- Week 3:
  - Teams conduct problem analysis
  - Teams write report R1
  - Teams start working on R2 (architecture) and Proof-of-Concept
- Week 4:
  - Deadline R1 (80%): Monday Sept 23, 17:00
  - Tue-Fri: Refinement meeting with TA and other team
  - Move focus to R2 (architecture) and Proof-of-Concept

Labwork by Week (5-7)



- Week 5 & 6:
  - Finalize R1
  - Work on R2 and PoC
  - TA / Team progress meeting in W6
- Week 7:
  - Deadline Monday October 14, 17:00:
    - **Final** version R1; **80% version** of R2 and PoC.
  - Tue-Fri: Conduct reviews of other teams
  - Continue finalizing R2 (architecture) and Proof-of-Concept

Labwork by Week (8, 9)



- Week 8:
  - Deadline: Submit review, Monday October 21, 17:00
  - Revise R2 & PoC to process feedback, draw lessons from R1 feedback
  - Finalize R2 & Proof-of-Concept
  - Prepare presentation
- Week 9:
  - Tue Oct 29 + Wed Oct 30: attend **full morning** for presentations
- Week 10:
  - Monday Nov 4, 17:00. Submit **final** versions of R2 & PoC

Rubrics for Grading & Peer Review

- Derived from seven learning objectives
- Will be elaborated in lectures
- Involve technical content as well as communication skills (writing, presentation)
- Will accommodate for freedom to shape this course towards your own learning objectives

Report 1 Rubrics

Main Criteria

- Problem description rigor
- Use of domain analysis techniques
- Model quality
- Writing clarity

Levels per Criterion

- 3 points: Outstanding
- 2 points: Good
- 1 point: Weak
- 0 points: Missing

Rubric Report 1, Problem Analysis

	Level 1 -- Outstanding (3 points)	**Level 2 -- Good** (2 points)	**Level 3 -- Weak** (1 point)	**Level 4 -- Missing** No points
Problem description rigor	Compelling, coherent analysis of a convincing and challenging problem	Adequate analysis of a problem relevant to software architecture	Little value added by the problem description to what is already known	Overall report weak and unconvincing
Use of domain analysis techniques	Exemplary and in depth use of range of analysis techniques covered in course. Clear which techniques were chosen and why, and how they contribute to overall story	Adequate use of relevant analysis techniques	Superficial use of domain analysis techniques that adds little value	The report shows very little evidence of engagement with material as discussed in course
Model quality	Relevant models (e.g., context, domain class / interaction diagrams) and diagrams, well integrated in the text, used to illustrate key points.	Good models, but only weakly connected to the main storyline	Few or superficial models, that add little to the overall story	No models
Writing clarity	Clear and carefully worded writing with authentic tone and coherent paragraphs; adequate use of references in application domain or software architecture literature	Clear and professional writing, but not very engaging. Indicator can be overuse of series of bullet points with insufficient connections.	Adequate writing with mostly correct sentences, but with insufficient connections between paragraphs and insufficient coherence within paragraphs.	Sloppy writing with mistakes in e.g. grammar or formatting.

Final version will be on BrightSpace

Summary: Possible Problem Analysis Ingredients

1. The system vision
2. A domain model
3. Scenarios
4. Quality attributes
5. Stakeholder analysis
6. External dependencies
7. Sources of ambiguity
8. Pricing models
9. Ethical considerations
10. Roadmap considerations

Domain analysis techniques discussed in later lectures

Report 2 Rubrics

Main criteria:

- Architectural description rigor
- Use of architectural analysis, patterns, and techniques
- Model quality
- Writing clarity
- Elaboration of decision points
- Connection to proof-of-concept

Rubric Report 2, Architectural Description

	Level 1 -- Outstanding (3 points)	**Level 2 -- Good** (2 points)	**Level 3 -- Weak** (1 point)	**Level 4 -- Missing** No points
Overall architectural description rigor, coherence, and storyline	Compelling, coherent, and rigorous; description of an architecture that realizes the system proposed in R1. Includes a careful consideration of stakeholder needs and (possibly conflicting) quality attributes	Adequate presentation of an architecture for the required system. Improvements possible in terms of, e.g., coherence, depth, or choice of topics covered.	Superficial presentation of an architecture. Appears to be in the right direction, but lacks in rigor; Not sufficiently clear how it will meet needs expressed in problem analysis.	Overall architectural description weak and unconvincing
Use of architectural analysis patterns, methods, and techniques	Exemplary and in depth use of range of architectural analysis and design techniques covered in course and in literature; Clear which techniques were chosen (with references) and why, and how they contribute to overall story	Adequate use of relevant architectural analysis and design techniques. Improvements possible in terms of cohesion or depth of the analysis.	Superficial use of architectural analysis and design techniques adding little value.	The report shows very little evidence of engagement with material as discussed in course
Model quality	Relevant models (e.g., container, component, class, interaction diagrams) and architectural views, well integrated in the text, used to illustrate key points.	Good models, but only weakly connected to the main storyline	Few or superficial models, that add little to the overall story	No models
Writing clarity	Clear and carefully worded writing with authentic tone and coherent paragraphs; adequate use of references in application domain or software architecture literature	Clear and professional writing, but not very engaging. Indicator can be overuse of series of bullet points with insufficient connections.	Adequate writing with mostly correct sentences, but with insufficient connections between paragraphs and insufficient coherence within paragraphs.	Sloppy writing with mistakes in e.g. grammar or formatting.
Elaboration of key architectural decision point	Important and challenging decision point selected, with in depth analysis of four alternatives, and a well-motivated choice among them.	Adequate description of an architectural decision point and four alternatives. Improvements possible in terms of rigor or relevance to a decision in the overall architecture.	Superficial description of architectural decision point, with weak analysis of some of the alternatives.	Missing description of architectural decision point and possible choices
Relevance and quality of the proof of concept as described in the report	Clear and convincing proof of concept explained at the right level of abstraction and supported by insightful diagrams, well connected to a key architectural decision.	Adequate description of an interesting proof of concept. Improvement possible, e.g., in terms of rigor, or relevance to a decision in the overall architecture.	Superficial description of a proof of concept, with a weak connection to the overall architecture	Missing description of the proof of concept

Final version will be on BrightSpace

Report 2 Elements (II)

- See if C4 / C5 (connectors) matches your system
- Explore arc42 and see what fits
- Main architectural style(s) adopted
- Selected (relevant) architectural views
- Selected scenarios and how they affect architectural elements
- Way in which key quality attributes are achieved
  - Including tradeoffs between them
- (Remote) API design, including principles and patterns used

Architectural analysis techniques discussed in later lectures

Rubric Proof of Concept

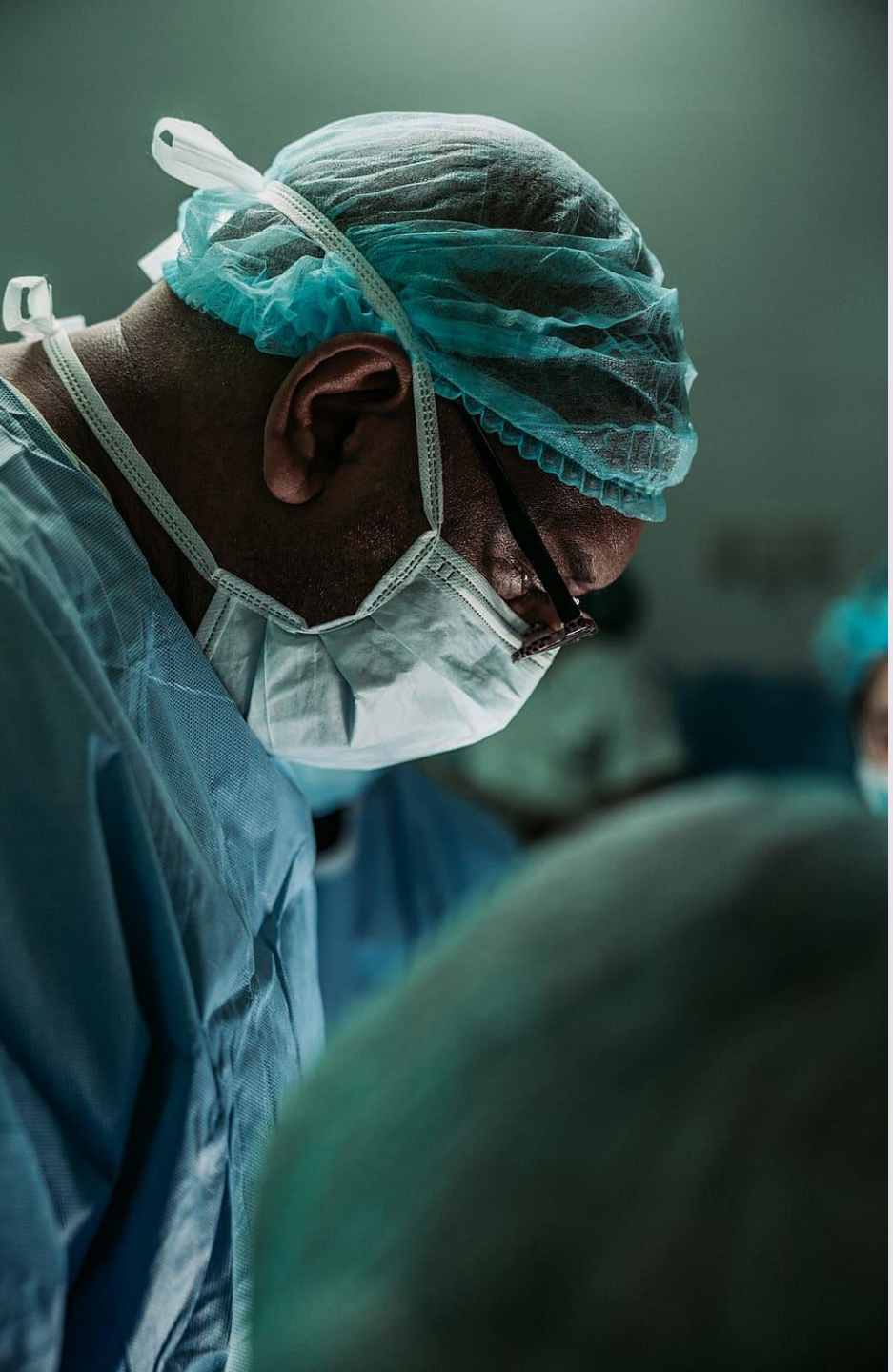
	Level 1 -- Outstanding (3 points)	**Level 2 -- Good** (2 points)	**Level 3 -- Weak** (1 point)	**Level 4 -- Missing** No points
PoC objectives and relevance to architectural decisions	The PoC comes with a clearly articulated goal, which when achieved will help resolve a challenging and relevant architectural decision.	There is an explicitly stated goal, but a relatively simple PoC and analysis would be sufficient to meet the stated goal.	The stated goal is trivial or contributes little to the architectural decision making	There is no stated objective for the PoC, or it is unclear what the PoC is supposed to bring.
Depth and rigor of the constructed PoC	The PoC is exemplary, thoughtfully designed, and convincingly supports the architectural decision making process related to the stated goal of the PoC.	The PoC is a working system or interface design that adds relevant insight to the architectural decision making process.	The PoC yields a working system or interface design that could contribute to the given (or apparent) goal of the PoC.	The PoC is trivial to construct and unsuitable for any architectural analysis.
Depth and rigor of the experiments conducted using the PoC and the lessons learned from it	The PoC is used to draw compelling lessons, by means of carefully conducted experiments or analysis, critically reflecting on the results, and well presented using relevant visualizations	The PoC is used to conduct measurements and draw lessons. Overall good, bug improvements possible in terms of, e.g., critical reflection of the results, depth of the analysis, or rigor of the measurements.	The PoC was used to conduct light weight measurements or analysis, and some reflection on e.g. feasibility	A PoC has been constructed, but it was not used to conduct experiments or draw lessons.
Use of sound engineering practices	The proof-of-concept follows sound engineering practices, including automated testing, CI/CD, containerization, code review, and clearly documented design choices, making the PoC understandable, modifiable, and the results credible reproducible	Adequate use of engineering best practices. Room for improvement of existing practices or adoption of more practices	Superficial or limited use of engineering best practices in terms of amount and depth/quality	The PoC fails to meet minimum engineering standards, such as the use of CI/CD, four eyes principle in reviewing, documentation, testing, or reproducibility.

Final version will be on BrightSpace

Grading: Team + Individual

- Reports + PoC + Presentation: Team grade
  - 60%: Reports R1 + R2
  - 20%: PoC
  - 20%: Presentation
- Individual pass / fail (!!):
  - Handing in reviews in time
  - Presence at Presentation Day and at TA meetings (W2, W4, W6, W8)
- Individual adjustments of ±1pt possible based on:
  - Team's self assessment of distribution of work
  - Brightspace Buddycheck outcomes
  - git history (blog + PoC) / accountability analysis
  - TA impressions of participation activity

All partial grades must be pass or > 5.8



Repair?

- Health-related (or other) special cases:
  - *Always via academic counselor*
    (whose advice the teachers will follow)
- Team insufficient?
  - Repair of reports or PoC via
    extra assignment.
- Individual insufficient?
  - *Avoid through active participation*
  - Repair via *extra assignment*

Team Dynamics

- Safe space for students and teachers
- In case of internal team problems:
  - Discuss with team, with TA, renew agreements
  - Last resort: Re-grouping
- In case of health problems: Academic counselor
- Reduced availability / drop out:
  - Inform your TA as soon as possible
  - Don’t let your team mates suffer

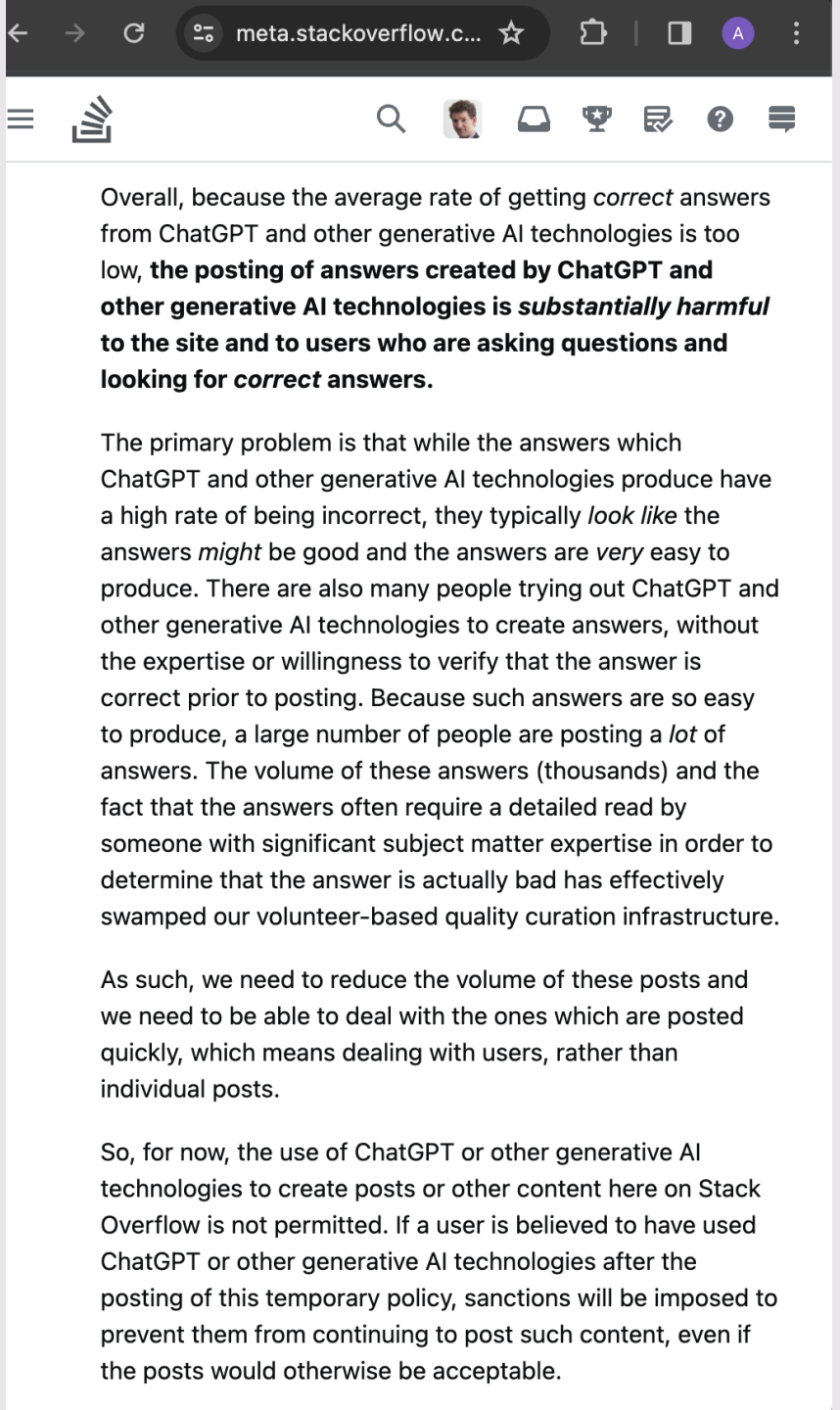
AI-Augmented Architecting?

- Idea generation & creativity
- Writing assistant
- Coding assistant
- Access to common knowledge
  - Analysis of pros / cons, tradeoffs
  - General domain knowledge (verify!!)
- Design space exploration?

StackOverflow Generative AI Policy

All use of generative AI
(e.g., ChatGPT and other LLMs)
is banned when posting content
on Stack Overflow.

This includes "asking" the question
to an AI generator
then copy-pasting its output
as well as using an AI generator
to "reword" your answers.



Overall, because the average rate of getting *correct* answers from ChatGPT and other generative AI technologies is too low, **the posting of answers created by ChatGPT and other generative AI technologies is *substantially harmful* to the site and to users who are asking questions and looking for *correct* answers.**

The primary problem is that while the answers which ChatGPT and other generative AI technologies produce have a high rate of being incorrect, they typically *look like* the answers *might* be good and the answers are *very* easy to produce. There are also many people trying out ChatGPT and other generative AI technologies to create answers, without the expertise or willingness to verify that the answer is correct prior to posting. Because such answers are so easy to produce, a large number of people are posting a *lot* of answers. The volume of these answers (thousands) and the fact that the answers often require a detailed read by someone with significant subject matter expertise in order to determine that the answer is actually bad has effectively swamped our volunteer-based quality curation infrastructure.

As such, we need to reduce the volume of these posts and we need to be able to deal with the ones which are posted quickly, which means dealing with users, rather than individual posts.

So, for now, the use of ChatGPT or other generative AI technologies to create posts or other content here on Stack Overflow is not permitted. If a user is believed to have used ChatGPT or other generative AI technologies after the posting of this temporary policy, sanctions will be imposed to prevent them from continuing to post such content, even if the posts would otherwise be acceptable.

Beware of ChatGPT's Limitations

It will *hallucinate*: It will invent likely answers, with no guarantee any of it is correct.

Answers as useful as the prompt you engineer: This takes effort too.

Answers may bias you, and limit your incentive to look beyond them

ChatGPT is human too: it is a master in politely not actually answering the question.

Biggest challenge: turn vague, wordy, and abstract writing into crisp, to the point, and compelling line of reasoning

ChatGPT / Generative AI CS4505 Usage Constraints

- Usage permitted
  - Idea generation, writing/coding assistant, common knowledge
- All usage must be **verified**
  - Never trust ChatGPT
  - Always check answers
  - Always identify and reference reliable knowledge sources
- All usage must be **documented**
  - Clearly explain usage (in appendix) and verification procedure

Responsible Architecting

- The architect is a decision maker
- Making a decision means *being accountable*
- This accountability can never be replaced by AI

Labwork Learning Objectives

Demonstrated ability to:

- **Understand and address upstream ambiguity**
  - *From domain knowledge and stakeholder needs, derive solution neutral system objectives*
- **Design architecture that meets stated objectives**
  - *Manage (down stream) complexity, adopt creativity, explore alternatives, demonstrate effectiveness of chosen solution*
- **Communicate design decisions and trade-offs**
  - *Through models, written text, presentations, and a proof-of-concept*