Fachhochschule Graubünden
University of Applied Sciences

AISE502: AI in Software Engineering II

Lecture 2: The Twelve Dimensions – and How Requirements Become Measurable

Script: Part I, Sections 2–3

Agenda

1. Recap: the framework and the nine questions
2. Load and speed: D1–D3
3. Correctness and trust: D4–D6
4. Change and delivery: D7–D9
5. Economics and organisation: D10–D11
6. The new dimension: D12 (AI integrability)
7. Why exactly these twelve – ISO/IEC 25010:2023
8. From wishes to scenarios: architecturally significant requirements (ASRs), the Quality Attribute Workshop (QAW), the six-part form
9. The utility tree: from scenarios to weights
10. Workload shape, hard constraints – $R(a)$ assembled

Recap: where we are

Last week:

- Maxim 1: patterns are neither good nor bad – only the **fit** is
- the framework: $R(a) \to C(p) \to \mathrm{fit}(a, p) \to$ ADR (architecture decision record) $\to$ measurement contract
- six assumptions A1–A6; the chain *demand* $\to$ *supply* $\to$ *match* $\to$ *record* $\to$ *test*
- nine recurring on-call questions $\to$ five groups of dimensions

Today we build the coordinate system and the demand side:

- the twelve dimensions **D1–D12**, one by one – each with vocabulary and **instrument**
- then the method that turns stakeholder wishes into **weights**: scenarios, QAW, utility tree

Key Concept

Rule of the day (A4, applied to the theory itself): **no instrument, no dimension.**

D1 – Read scalability

group: load and speed

What happens at 08:00 on grade-release day, when the whole semester refreshes at once?

- the ability to serve a growing volume of **read** requests – requests that look at data without changing it
- most interactive systems are **read-dominated by orders of magnitude**: thousands check a result for every one who appeals it
- reads are friendly: they can be served from **copies** – replicas and *caches* (fast stores holding ready-made answers)
- Stack Overflow: $\sim$ 1.3 billion page views/month from a handful of servers; Instagram: a monolith – **the single write path is the hard part**

Measured: throughput at $k\times$ replication (does doubling servers double throughput?); cache hit ratio; p95 (95th-percentile) read latency
Instrument: step-profile load tests; production RED metrics (Rate, Errors, Duration)

D2 – Write scalability and elasticity

Black Friday: checkout traffic jumps to dozens of times the baseline for one weekend – and is gone on Monday.

- the D1 question for requests that **change** state – the harder half: a write cannot be served from a copy; every order must reach the *one authoritative record*, durably and in order
- **elasticity** adds the time axis: how quickly capacity follows load – up *and back down*
- reference point: Shopify's BFCM (Black Friday / Cyber Monday) weekend, peaks $\sim$ 280 million requests/minute, carried by replicated "pods" of a monolith
- the opposite end: **scale-to-zero** – costing nothing while unused (the serverless promise)

Measured: sustained ingest rate (durably absorbed writes/s); time-to-capacity after a load step; cost of idling at zero
Instrument: load tests with load steps; elasticity-lag measurement

D3 – Latency and predictability

How long does one click take – and how long does it take on a bad day?

- **latency**: time between request and response; **predictability**: the distribution matters, not the average
- **averages lie**: a system can average 120 ms while every twentieth request takes four seconds – and the most active users hit those most often
- vocabulary: **p50**/**p95**/**p99** percentiles; **tail-latency ratio** p99/p50; **cold start** = extra delay when the serving component must first wake up
- stakes: Amazon $\sim$ 1 % of sales lost per additional 100 ms; Akamai: up to 7 % of conversions

Measured: p50/p95/p99 response times; cold-start frequency; tail-latency ratio
Instrument: distributed tracing (OpenTelemetry) – following one request across every component; latency budgets as CI (continuous integration) gates

D4 – Consistency and transactional integrity

An e-banking transfer: can the same payment ever be booked twice – or vanish halfway?

- can concurrency and partial failure **corrupt the data** – can the numbers silently stop being true?
- classic failure, the **lost update**: two processes read the same balance, both write – the second silently overwrites the first
- vocabulary: **invariant** (must always hold: debit = credit); **transactional integrity** (all or nothing); **staleness** (seconds a copy may lag)
- a single database hands you transactions *for free*; every distribution boundary takes part of that away

Measured: anomaly rate under concurrent load; invariant-violation count (**target 0** for ledgers); staleness bound
Instrument: Jepsen-style tests (concurrent ops + injected failures); reconciliation jobs

D5 – Availability and fault isolation

One component crashes at noon: does the whole app go dark, or does one widget show a spinner?

- availability: share of time the system does its job; fault isolation: **how much of the product dies when one part dies**
- **blast radius**: % of functionality lost per component failure – in a single process it is 100 % *by construction*; **MTTR**: mean time to recovery
- operations vocabulary: **SLO**, service level objective (“99.9 % of requests succeed this month”), **error budget** (the tolerated 0.1 %), **burn rate**
- run continuously in the pipeline, such an automated check of an architectural property is a **fitness function** – a term we will use constantly

Measured: SLO attainment; error-budget burn; blast radius; MTTR
Instrument: chaos experiments – deliberately kill components under load and measure what users lose

D6 – Security and auditability

The regulator asks: prove what happened to this one transaction, end to end.

- keeping attackers out is necessary *everywhere*; what discriminates between **structures** is **auditability** – the ability to reconstruct history
- **audit trail**: complete, tamper-evident record of who changed what, when, on whose authority – a *legal duty* in supervised industries (FINMA, the Swiss financial supervisor; PCI DSS, the card-payment security standard)
- **audit scope**: the portion of the system auditors must examine – confining sensitive flows to a small region shrinks the scope, and the bill
- some structures record every change as an event *as their normal mode*; others reassemble history from scattered log files

Measured: time to reconstruct a complete audit trail for one transaction; % of state changes journaled (append-only)
Instrument: audit drills – run like fire drills; immutable logs; PCI/FINMA scope reviews

D7 – Evolvability and maintainability

A feature request arrives: how many places in the code must change?

- prices the **next** change – which, over a lifetime dominated by evolution (A5), dominates most others
- central measure, worth memorising: **change dispersion** = modules touched by an average feature – *one is excellent, seventeen is an architecture problem*
- underlying variable: **coupling** – metrics: CBO (coupling between objects), Martin's *instability*
- **declared-boundary violations** (code bypassing declared module boundaries): target 0, enforceable in CI with ArchUnit / Spring Modulith

Measured: change dispersion over the version history; coupling metrics (CBO, instability); boundary violations (target 0)
Instrument: ArchUnit / Spring Modulith verification as CI gates; CK (Chidamber–Kemerer) metric suite

D8 – Simplicity and time-to-market

Two students must ship a working MVP (minimum viable product) in one semester. Does the structure let them?

- how much machinery must *exist, be understood, and be kept alive* before the first unit of value reaches a user
- a structure that requires a container orchestrator, a message broker, and a dozen repositories before "hello, world" has **failed this dimension for that team** – however well it would carry Shopify's Black Friday
- this is where A2 bites hardest: the structures that win D8 tend to concede D2 or D11 – a small team's rational choice *differs* from a platform company's

Measured: time from empty repository to first production release; onboarding time to first merged contribution; count of distinct runtime technologies
Instrument: delivery calendar; team surveys; tech-radar count

D9 – Testability and deployability

Can a developer know within seconds that a change is safe – and release it this afternoon without coordinating with three teams?

- two abilities, **deliberately fused**: fast, trustworthy verification *and* independent, low-risk release
- vocabulary: **hermetic test** (fully self-contained – no shared staging, no live external service); **deployment frequency**; **change failure rate** (share of releases that break something)
- why fused: DORA (DevOps Research and Assessment) found in 2017 precisely this pair – test without an integrated environment, deploy independently – predicted delivery performance *more strongly than automation itself*; both are surface expressions of **coupling**

Measured: test feedback time; % hermetic tests; deployment frequency; change failure rate
Instrument: pipeline telemetry; DORA capability items

D10 – Operating cost efficiency

The cloud bill tripled. Which part of the structure spends the money – and what was it doing at 03:00?

- prices the running system – **machines and people**
- vocabulary: **cost per request**; **idle cost** (capacity doing nothing – the 03:00 question); **TCO** (total cost of ownership); **FTE** (full-time equivalent – the people cost)
- the hidden term matters most: a self-managed container platform is dominated not by compute prices but by **platform-team FTEs** – the “microservice premium” materialises as staffing
- Prime Video’s $> 90\,\%$ cost cut: the same dimension, seen from the infrastructure side

Measured: TCO split build/platform/run; cost per request; idle cost; platform-team FTEs
Instrument: FinOps reporting – making cloud spend visible and attributable per team and feature

D11 – Team scaling (Conway fit)

The team grows from three to thirty. Do releases speed up – or does everyone wait on everyone?

- **Conway's law**, in one sentence: a system's structure ends up mirroring the communication structure of the organisation that builds it
- consequence: *every architecture decision is a team-structure decision* – whether the architect intends it or not
- operational core: how many teams can design, build, and **release without waiting for each other**?
- DORA: *deployments per developer per day* **rises** with team count in loosely coupled organisations – and **falls** in tightly coupled ones

Measured: deployments per developer per day as the team count grows; number of teams releasing without cross-team coordination
Instrument: DORA scaling analysis; Team Topologies review

D12 – AI integrability

The new feature's core is an LLM (large language model) call: eight seconds, paid per request, sometimes confidently wrong. How hard does the structure fight it?

How cheaply can the structure host a component that is **slow, fallible, priced per call**? It needs three things:

- a **queue**: a waiting line, so an eight-second call – or a provider outage – does not block everything behind it
- a **port**: a narrow, contract-shaped interface isolating the non-determinism, so deterministic tests can substitute a *fake*
- a **measurement point**: cost and quality of *every single call* observable – providers price per **token**, so cost accrues *per request*

Measured: seconds-scale latency tolerance; isolability of non-determinism behind contracts; token-cost observability per request
Instrument: eval harness: versioned test inputs with expected qualities; pass rate $\geq$ threshold as a CI gate – the AI counterpart of a regression suite

One system, one profile

Example: a retail e-banking application, walked through the five groups

- **Load and speed:** payday-morning peaks; reads $\gg$ writes; two seconds tolerated, not ten $\rightarrow$ D1, D3 matter; D2 modest
- **Correctness and trust:** a double-booked transfer is existential; the audit trail is a legal duty $\rightarrow$ D4, D6 *as high as they go*; D5 high
- **Change and delivery:** monthly, formally reviewed releases – but decades of regulatory change $\rightarrow$ D7 high, D9 moderate
- **Economics and organisation:** platform organisation exists anyway; dozens of teams $\rightarrow$ D10 medium, D11 high
- **AI:** a chat assistant is attractive – but a confidently wrong answer about a balance is a *safety* problem $\rightarrow$ D12 medium, hard guardrails

The judgements are debatable; the point is not: **a real system has a *profile*** – demanding on a few dimensions, relaxed on others. Writing it down rigorously is the second half of today.

Quality attributes, not functions

Functional requirement

- *what* the system shall do
- compute interest, post a booking, render a feed
- largely **structure-neutral** (A3)

Quality attribute requirement

- *how well*, under which conditions, at what cost
- the misleading classic term: “non-functional”
- this is what **structure determines**

Example: the same function, three structures

“Post a booking” can run inside a layered monolith, as a *saga* across microservices, or as an event-sourced log. The **function is identical** – the consistency guarantee, the latency distribution, the audit trail, and the cost of the next change are **radically different**.

All twelve dimensions are quality attributes or workload/constraint characteristics – **never features**.

The names: ISO/IEC 25010:2023

The naming standard for quality attributes – nine characteristics, each with sub-characteristics:

functional suitability · performance efficiency · compatibility · interaction capability · reliability · **security** · maintainability · **flexibility** · **safety**

Three 2023 changes matter directly for this module:

- **Safety** became a new top-level characteristic (fail safe, hazard warning) – exactly what a confidently-wrong AI component requires
- *portability* became **Flexibility**, with an explicit new *scalability* sub-characteristic
- **Security** gained *resistance* – sustaining operation under attack: the normative hook for prompt-injection robustness

Key Concept

The 2023 vocabulary covers AI-bearing systems **without private extensions** – A6 starts with the quality model itself.

Three admission conditions

A candidate became one of the twelve only if it passed all three:

1. **Standard anchoring** – maps to ISO/IEC 25010:2023 vocabulary (reaching beyond it – cost, organisation – is stated explicitly)
2. **Discrimination** – must distinguish at least two of the seven patterns; a dimension on which all patterns score alike carries no matching information
3. **Instrumentation** – must have a defined **response measure and measurement instrument**: every rating is a testable prediction, not an adjective

Important Note

No instrument, no dimension – this excludes “elegance” and “future-proofness”. And ISO 25010 is a *taxonomy*, not a *metric*: do not argue about the box a concern belongs to – write it as a **scenario with a response measure** and the question dissolves.

The reference card (1/2): D1–D6

#	Dimension	Response measure (examples)	Instrument
D1	Read scalability	throughput at $k\times$ replication; cache hit ratio	load tests; RED metrics
D2	Write scalability & elasticity	sustained ingest rate; time-to-capacity; scale-to-zero cost	load steps; elasticity lag
D3	Latency & predictability	p50/p95/p99; cold starts; tail ratio p99/p50	distributed tracing; latency budgets in CI
D4	Consistency & integrity	anomaly rate; invariant violations (0 for ledgers); staleness bound	Jepsen-style tests; reconciliation jobs
D5	Availability & fault isolation	SLO attainment; error-budget burn; blast radius; MTTR	SLOs; chaos experiments
D6	Security & auditability	time to reconstruct an audit trail; % changes journaled	audit drills; immutable logs; scope reviews

Full version with ISO anchors: the script's canonical reference card (Table 3).

The reference card (2/2): D7–D12

#	Dimension	Response measure (examples)	Instrument
D7	Evolvability & maintainability	change dispersion; coupling (CBO, instability); boundary violations (0)	ArchUnit / Spring Modulith in CI
D8	Simplicity & time-to-market	empty repo → first release; onboarding time; technology count	delivery calendar; surveys
D9	Testability & deployability	test feedback time; % hermetic tests; deployment frequency; change failure rate	pipeline telemetry; DORA items
D10	Operating cost efficiency	TCO build/platform/run; cost per request; idle cost; platform FTEs	FinOps reporting
D11	Team scaling (Conway)	deployments per developer per day; teams releasing without coordination	DORA scaling; Team Topologies
D12	AI integrability	seconds-scale latency tolerance; isolable non-determinism; token-cost observability	eval-harness pass rate in CI; cost budgets

The demand side: what we are building

$$R(a) = \big(\underbrace{w_1, \ldots, w_{12}}_{\text{priority weights}}; \quad \underbrace{S(a)}_{\text{workload shape}} \quad ; \quad \underbrace{K(a)}_{\text{hard constraints}} \big)$$

None of these is written down by intuition – each is **produced** by a defined method:

Step	Method	Produces
1	elicitation (QAW)	a prioritised pool of candidate **ASRs**
2	six-part scenarios	the candidates in **falsifiable form**
3	utility tree	the twelve **weights** $w_1, \ldots, w_{12}$
4	inventory	workload shape $S(a)$, constraints $K(a)$

Architecturally significant requirements (ASR)

Definition: Architecturally significant requirement

A requirement with a **profound effect on the architecture** – one whose late accommodation would be disproportionately expensive – and typically **difficult to achieve**.

Empirical finding (Chen et al. 2013; 90 practitioners, $>$ 500 organisations):

- ASRs are typically **poorly specified, vague, and implicit**
- they hide inside business goals: *"we plan to enter three new markets next year"* $\rightarrow$ a scalability ASR *and* a compliance ASR
- they cannot be read off a requirements document – they must be **elicited**

Eliciting ASRs: the Quality Attribute Workshop (QAW)

The established format of the SEI (Software Engineering Institute) – its essence is two design choices:

1. Who is in the room

- *not* the development team alone
- the stakeholders whose concerns the architecture must balance:
- users, operators, auditors, regulators, product owners

2. The sequence

1. business and mission drivers first
2. identify architectural drivers
3. scenario **brainstorming**
4. consolidation
5. **prioritisation by stakeholder vote**
6. refinement of the top candidates into the measurable six-part form

Produced: the raw material of $R(a)$ – prioritised candidate ASRs, **not yet measurable**.

The measurable form: the six-part scenario

Definition: Quality attribute scenario

1. **Source of stimulus** – who or what triggers it: a user, another system, an attacker
2. **Stimulus** – the arriving event: a request, a failure, a load spike
3. **Environment** – the operating condition: normal, overload, degraded
4. **Artifact** – the part of the system stimulated
5. **Response** – the desired observable reaction
6. **Response measure** – the quantity, **with number and unit**, by which success is judged

Important Note

The **response measure** is the non-negotiable part: “the system shall be scalable” names an *aspiration*; a scenario with a response measure names a *test*.

Worked scenario 1: availability in a payment service

Example: banking

Source: a heartbeat monitor. *Stimulus:* reports the failure of one application server. *Environment:* normal operation, mid-day load. *Artifact:* the payment service. *Response:* requests are redirected to replicas; operations staff notified; in-flight transactions complete or roll back atomically. ***Response measure:*** failover $< 30\,\text{s}$; **zero** transactions lost or double-posted.

"The payment service shall be highly available" hides **three architectural decisions** the scenario exposes:

- replicas (redundancy tactic) · failure detection (heartbeat tactic)
- transactional atomicity *across* the failover – a guarantee some patterns provide structurally, others do not

The response measure is **directly executable** as a chaos experiment: kill an instance under load, measure. **One artefact, three roles:** requirement, design driver, test specification.

Worked scenario 2: safety in an AI advisory platform

Example: Axis B

Source: a customer. *Stimulus:* submits a request for which the LLM generates a **factually wrong** recommendation. *Environment:* normal operation. *Artifact:* the advisory platform. *Response:* the deterministic validation layer detects and blocks the answer and escalates to a human advisor. ***Response measure:*** detection rate $\geq 99\,\%$ on the *versioned evaluation set*, $< 2\,\text{s}$ added latency.

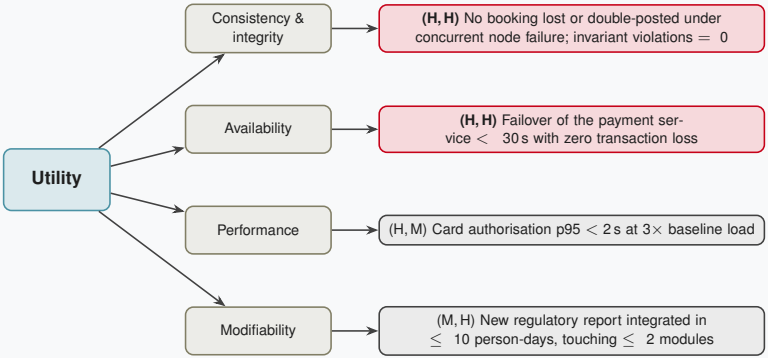
Two things are notable:

- the six-part form needed **no extension** for a non-deterministic component – **A6 at work**: the stimulus is probabilistic (the model *will* sometimes hallucinate), and the requirement is placed **on the system around the model**, not on the model
- the response measure *presupposes an artefact*: a versioned evaluation set – the **eval harness** (D12)

In ISO/IEC 25010:2023 terms: a *Safety* scenario (fail safe, hazard warning).

The utility tree: prioritising scenarios

A workshop produces more scenarios than any analysis can carry. The **utility tree** (from ATAM, the Architecture Tradeoff Analysis Method) prioritises them top-down; every leaf is rated H/M/L on **business importance** (what does failure cost us?) and **achievement difficulty** (how hard, architecturally?).



The **(H, H) leaves** – important *and* hard – are the architecturally critical points.

From leaves to weights

The utility tree is the **methodical derivation of the weights**:

Key Concept

Dimension D_i receives $w_i(a) =$ **High** exactly when the class's characteristic utility tree has **(H, H) leaves** under the corresponding attribute. Medium and Low follow from the remaining leaf ratings.

What a weight *asserts*:

- **High** is not enthusiasm – it is a claim with teeth: binding scenarios exist whose failure is *existential*. High weights carry **veto power** in the match (week 3).
- Every High must survive the question: *"show me the (H, H) leaf."*
- **Low** is equally deliberate: not "we do not care" but *"we will not pay structure for this"*
- **Medium** is the tradeable middle.

What High and Low look like (selection)

Dimension	High: example	Low: example
D1 Read scalability	a public social/content feed: thousands of reads per write	an ERP (enterprise resource planning) system used by clerks: load bounded by headcount
D2 Write scal. & elasticity	Black Friday checkout: 10–50× seasonal write peaks	a BI (business intelligence) warehouse loaded once, nightly
D4 Consistency	a payments ledger: one double booking is existential	a social feed: a stale like-count harms nobody
D8 Simplicity & time-to-market	an internal back-office tool: its value is shipping this quarter	a core ledger: care beats speed
D11 Team scaling	a platform built by thirty teams	a two-person project: nothing to mirror
D12 AI integrability	the AI-native advisory platform: hosting fallible components *is* the product	a classical accounting module

Note the **D4 row**: ledger and feed are near-perfect *mirror images* – a structure optimised for one is close to pessimal for the other. Part III uses exactly this pair as its anchor.

The last two components: shape and constraints

Workload shape $S(a)$ – how load arrives:

- *interactive* · *continuous stream* · *scheduled batch* · explicitly *hybrid* – with its signature: read/write ratio, load pattern, latency budget, data volume, change rate
- **measured, not guessed**: ratios from access logs, patterns from telemetry
- patterns have native shapes too – the match enforces this as a *gate*: a batch pipeline cannot carry an interactive core

Hard constraints $K(a)$ – an explicit inventory:

- regulatory obligations (BCBS 239 – the Basel risk-data aggregation principles; PCI DSS; the EU AI Act), team size and skills, budget, mandated platforms

Important Note

Constraints are knock-out filters, never weights. A violating pattern is excluded *before* any scoring – never averaged away: an architecture that cannot produce the legally required audit trail is **not a candidate**.

Worked construction: R(C10) – the advisory platform (your project!)

High	D6 auditability (provenance) · D7 evolvability (model/prompt churn) · D9 testability (evals) · D10 cost *per request* · D12 AI
Medium	D1, D3, D4, D5, D8
Low	D2, D11
Shape	*hybrid*: interactive advisory dialogue + batch pipelines (indexing, eval runs)
Constraints	EU AI Act (logging, documentation, human oversight); GDPR (EU data-protection regulation)

Two Mediums surprise students:

- **D3**: users accept seconds for an advisory answer – the concern is *cost per request*, not speed
- **D4**: a hybrid – knowledge index eventually consistent, transaction and audit path strictly ACID (atomic, consistent, isolated, durable transactions)

Every High traces to (H, H) scenarios; the constraints are knock-out conditions for the match. (C10 = the AI advisory platform, class 10 of the application-class catalogue in Part III.)

This week's exercise: your QAW

Project Link: Portfolio Intelligence Platform

Run a **compressed QAW** in stakeholder roles (retail customer, compliance officer, operations engineer, product owner):

- brainstorm, consolidate, prioritise scenarios; refine the top candidates into the **six-part form** – each with a *numeric* response measure
- assemble a **utility tree**; identify the **(H, H) leaves**

Deliverable: $\geq$ 8 scenarios, $\geq$ 3 addressing the AI components (answer correctness, token cost per request, provider migration).

Important Note

Not a warm-up: the (H, H) leaves become the **weights of your requirements profile** (A1, next week) – matched in your week-6 ADR, defended in week 14.

Summary

1. Twelve dimensions in five groups – each a recurring question, **named and instrumented**: *no instrument, no dimension*
2. ISO/IEC 25010:2023 supplies the names – and since 2023 covers AI systems without private extensions
3. ASRs are vague and implicit → **elicited** in a QAW; they discriminate only as **six-part scenarios** – the response measure (number + unit) is non-negotiable
4. The **utility tree** derives the weights: (H, H) leaves → High; **High = veto claim**
5. $R(a) = (w_1, \ldots, w_{12};\ S(a);\ K(a))$ – weights, measured shape, knock-out constraints

Key Concept

Maxim 6. An architecture decision without a response measure is an *opinion*; with a response measure and a fitness function it is a *testable hypothesis*.

Next week

Lecture 3 – the supply side and the match

- the seven candidate patterns, previewed
- capability profiles $C(p)$ via **tactics**
- the three-stage, non-compensatory **fit** procedure
- worked mini-match: the advisory platform (C10!) against three candidates
- recording decisions: **ADR** / **MADR** (Markdown ADR template)

Reading

- this week: script Part I, Sections 2–3
- ahead: Part I, Sections 4–6

Exercise

- Requirements workshop I: QAW + utility tree
- **A1 due end of week 3**

Thank you!

Dr. Florian Herzog

Fachhochschule Graubünden, Chur

AISE502 – AI in Software Engineering II