Foundations & Laws of Software Dev

SWEBOK & Plan-Driven Methods

- SWEBOK: core SE knowledge.
- Plan-driven — Pros: predictable, clear milestones, heavy docs, defined roles.
- Cons: rigid, late value, costly late changes.
- Use: mission-critical, regulated, stable requirements.
- Avoid: innovative, rapidly changing projects.

The Laws of Software Development

- **Conway**: systems mirror org communication.
- **Brooks**: adding staff to late projects delays them.
- **Parkinson**: work expands to available time.
- **Pareto fallacy**: final 20% often costly and slow.
- **Sturgeon**: most tools/code are low-quality.
- **Iceberg**: dev cost << lifecycle/ maintenance cost.
- **Peter**: people rise to incompetence level.
- **Eagleson**: code unread >6 months feels foreign.
- **Greenspun**: complex C/Fortran often reimplements Lisp features.

Agile Basics & Extreme Programming

The Agile Manifesto

- **Values**: individuals & interactions; working software; customer collaboration; responding to change.
- **Principles** (summary): frequent delivery, welcome change, close biz-dev collaboration, sustainable pace, technical excellence, simplicity, inspect & adapt.
- **Iron triangle**: Time & Cost fixed; Scope variable.
- **Goal**: flatten cost-of-change via continuous integration/testing.

Extreme Programming (XP)

- **Values**: communication, simplicity, feedback, courage, respect.
- **Planning Game**: biz+dev prioritize and estimate work.
- **Pair programming**: driver/navigator collaboration.
- **TDD**: write failing test, then code to pass.
- **Refactoring**: improve design without behavior change.
- **Collective ownership**: anyone may change any code.

Software Craft & “Tidy First?”

Software Craftsmanship

- **Focus**: restore technical excellence amid process-heavy Agile.
- **Practices**: coding dojos, katas, continuous skill practice.

Kent Beck’s “Tidy First?”

- Rule: separate refactor (structure) from feature (behavior) changes; tidy before adding features.

User Stories, Estimation & Planning

User Stories & Estimation

- Format: As a [role], I want [feature] so that [benefit].
- INVEST: Independent, Negotiable, Valuable, Estimable, Small, Testable.
- Hierarchy: Theme → Epic → Story.
- Story points: relative effort/complexity/risk.
- Velocity: story points completed per sprint.
- Planning Poker: consensus-based relative estimates.

Planning Concepts

- Kano: Basic, Performance, Excitement features.
- Planning levels: Strategy → Portfolio → Product → Release → Sprint → Daily.

Scrum & Kanban

Scrum

- Roles: Product Owner, Scrum Master, Developers.
- Events: Sprint, Planning, Daily, Review, Retrospective.
- Artifacts: Product Backlog, Sprint Backlog, Increment.
- DoD: checklist for releasable work.
- DoR: criteria for ready-to-start items.

Kanban & Lean

- Practices: visualize flow, limit WIP, manage flow, explicit policies.
- Lead time vs cycle time: request→done vs start→done.
- Kaizen: continuous small improvements.
- Waste: partial work, extra features, handoffs, delays, context switches, defects.

DevOps, Automation & CI/CD

DevOps Fundamentals

- DevOps: culture+practices+tools to deliver faster.
- DORA: deploy freq, lead time for changes, time to restore, change-failure rate.
- IaC: declare infra in code.

Pipelines

- CI: frequent merges trigger builds/tests.
- CD: changes auto-prepared; manual deploy trigger.
- Continuous deployment: passing changes auto-deploy.

Software Architecture

Concepts & Definitions

- Architect: sets high-level design, standards, business alignment.
- Layer vs Tier: logical code separation vs physical deployment.
- Brooks: complexity, conformity, changeability, invisibility.
- C4: Context → Containers → Components → Code.

Architecture Patterns

- Monolith vs Microservices: single deployable vs many small services.
- CQRS: separate read/write models.
- Event sourcing: persist events, rebuild state from events.
- Strangler: incrementally replace legacy functionality.
- Circuit breaker: stop repeated failing calls to protect systems.

Cynefin Framework

The 5 Domains of Cynefin

- Clear: obvious cause→effect. Use best practices.
- Complicated: needs expertise. Analyze then respond.
- Complex: outcomes visible in retrospect. Probe, sense, respond.
- Chaotic: act to stabilize, then sense/ respond.
- Disorder: unknown—gather info to classify.

Source Code

https://github.com/josias-r/zhaw-sem5-summaries