**Software Engineering 2022**

**Lecture Design Patterns**

Framework 🡪 a partially completed design that can be extended to solve a problem in a domain

Design Pattern 🡪 a description of a problem that reoccurs and an outline of an approach to solving that problem

* name and description of a proven solution to a design problem
* Documentation of a design decision
* Language independent
* Object oriented (usually not always)
* A good way to create understandable solutions for other devs to maintain

“Patterns are half baked – you always have to finish them yourself and adapt them to your own environment” – Martin Fowler

3 main categories of design patterns:

* Creational (Singleton,Factory…etc)
* Structural (Composite,Decorator ,Adapter …etc)
* Behavioral (State,Observer,Visitor …. etc)

Entity 🡪 Globally Unique Number

Component 🡪 Provides an aspect of state to an Entity. It’s an abstraction.

Entity is a primary key that holds the components together.

System -> A cohesive set of functionality that does all of one thing. (holds all behaviors)

**Lecture Requirements Engineering**

**Rules of Software:**

* When working on project , development is the easy part, maintenance is the hard part. Planning is needed.
* Make sure you are solving the right problem

**Software Requirement**

* Is a condition or capability needed by a user to solve a problem or achieve an objective
* Must be met or possessed by a system or component to satisfy a contract, standard , specification or other formal constraint.

Requirements are often buried “deep beneath layers of assumptions, misconceptions and politics”

*(Never assume you know what your customer wans. NEVER!!!!!!)*

**Requirements engineering:**

* The systematic way of developing requirements through an iterative process of analyzing a problem, documenting the resulting observations and checking the accuracy of the understanding gained.
* RE is Made of : 🡪 Analysis 🡪 Modelling

**Requirements Elicitation**

* Requirements elicitation is the process of gathering requirements from the stakeholders.
* A stakeholder is an entity that has vested interest in the system

**Types**:

* Functional 🡪 To specify what function system tends to perform
* Non-Functional 🡪 not necessarily associated with a single piece of functionality rather describe a state/condition of the system
* Constraints 🡪 Restrict the implementation in some way

**Functional requirements:**

* Specific and unambiguous
* Measurable and observable
* Testable in some way to determine if the requirement has been met

**Non-Functional requirements**:

* Security
* Privacy
* Usability
* Accessibility
* Reliability
* Availability
* Performance

Non-Functional requirements tend to be cross-cutting throughout the system. Often brought up as feelings or properties about the system. If (NFR contains problem) then all system might be needed to be redone.

**Elicitation Techniques**:

1. Interview
2. Observation
3. Examining Docs/Artifacts
4. Joint Application Design
5. Groupware
6. Questionnaires
7. Prototypes
8. Focus Groups
9. On-site customer

**Problems with requirements elicitation** :

1. The boundary of the system is ill-defined
2. Unnecessary design information may be given
3. Stakeholders have incomplete understanding of their needs
4. Stakeholders have poor understanding of computer capabilities and limitatations
5. Software Engineers have poor knowledge of the problem design
6. **Stakeholders and Software Engineers speak different languages**
7. “Obvious” information is omitted
8. Different stakeholders have conflicting views
9. Requirements are vague and untestable (user friendly , robust 🡪 Subjective)
10. Requirements are volatile and change over time

**Requirements Specification**

* Requirements Analysis – studying user needs to get a definition of the system
* Requirements modelling – turning use Requirements into actionable statements that all Software Engineers can unambiguously understand

**Requirements Specification** -> Restating of Requirements in a technical format of some kind that is actionable by Developers

**Use Cases**

* More visual way of modelling requirments
* Consists of use case diagram (UML)
* Based around interaction with the system

SRS (Software Requirements Specification)

* A template followed by a team to document functional Requirements. Non-functional Requirements and constraints
* Use Cases -> Used as part of a larger SRS
* Follows IEEE Specifications

**Risk Management**

Approaches

1. Ignore it
2. Be reactive of it (When a problem arises, everyone drops everything to address the issue)
3. Be proactive of it (If you identify and plan for some of the most common scenarios ahead of time, they won’t have as much impact when they occur)

Risk Management Cycle

* Identify
* Analyze
* Prioritize
* Plan
* Mitigate (lower costs)
* Monitor

Aspects of a project that can introduce risk:

* People
* Project Size
* Software Processes
* Tech and tools
* Organizational and managerial
* Customer
* Estimation
* Support

**SE Code of Ethics**

8 principles of SE Code of Ethics

1. PUBLIC – Software Engineers shall act consistently with the public interest
2. CLIENT AND EMPLOYER -> SE Shall ac in a manner that is in the best interest of their client and employer consistent with the public interest
3. PRODUCT – SE shall ensure that their products and related modifications meet the highest professional standards possible
4. JUDGMENT – SE shall maintain integrity and independence in their professional judgment .
5. MANAGEMENT – SE managers and leaders shall subscribe to and promote an ethical approach to the management of Software Development and maintenance
6. PROFESSION – SE Shall advance the integrity and reputation of the profession consistent with public interest
7. COLLEAGUES- SE Shall be fair to and supportive to their colleagues
8. SELF – SE Shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the profession.

**Lecture: Software Processes and Methodologies**

Following a process is key to making it work. Code is easy, Engineering good software is hard.

Software Process: 🡪 Set of activities that take place in sequence in the pursuit of creation of a software system.

All proceses have some form of:

* Requirements Elicitation
* Requirements Specification
* Design
* Development
* Validation
* Evolution (Maintenance)

Software Process Models:

* Waterfall
* Spiral
* Various Agile models
* Various Plan-driven models
* Etc

A specific software development methodology is an instance of a process model. (For ex Scrum is an Agile Methodology)

Factors that impact the choice of a company for the process they are following:

* Organizational Factors
* Technology Factors
* Domain/Sector Factors
* Regulatory Factors
* Human Factors

Agile Methodologies:

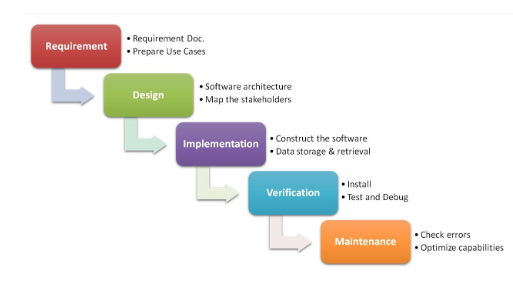
* Scrum
* Extreme Programming (XP)
* Kanban

Plan Driven Methodologies:

* Relational Unified Processes (RUP)
* Personal Software Processes (PSP)
* Team Software Processes (TSP)

Common Ancestor:

* Waterfall model



**Drawbacks**:

* Need iteration and feedback
* Very grid (not realistic to freeze results after each phase)
* Model does not emphasize important issues such as risk management, prototyping , quality

**Better Waterfall 🡺 Spiral = Waterfall with the ability to cycle back on phases.**

**Each cycle has 4 stages:**

* Determine objectives , constraints
* Identify and manage risks
* Develop and verify next stage or level of the product
* Review Results and plan for next stage

**Plan-Driven methodologies highlights:**

* Focus on repeatability and predictability
* Defined , standardized and incrementally improving process
* Thorough documentation
* A defined software system architecture defined up-front
* Details plans , workflow , roles , responsibilities and work product descriptions
* Ongoing risk management
* Focus on verification and validation

**RUP – Rational Unified Process**

Process is configurable. Built around 6 best practices:

* Develop software iteratively
* Manage requirements
* Use component based architectures
* Visually model software
* Verify software quality
* Control changes to software

Based in UML.

**RUP – Phases:**

* Inception Phase
* Elaboration Phase
* Construction Phase
* Transition Phase

Each phase has a distinct transition point into the next phase. 4 phases can be iterated on. Transition phase can be an internal build.

**Workflows**:

Within each phase there are workflows with particular tasks that increase and decrease in activity.

**Agile Processes:**

**Extreme Programming (12 Practices) :**

1. The planning game
2. Whole Team (on site)
3. Sustainable Pace (40 hour work week , avoid burn out of team members)
4. Small Releases (get customer feedback often to avoid wrong solution)
5. Coding Standards
6. Pair Programming
7. Test Driven Development
8. Refactor Mercilessly
9. Collective Code Ownership
10. Continuous Integration (constantly build the code, run tests, rebuild , always have a working build)
11. Simple Design (Simplest thing that works (build it) )
12. Metaphor over Architecture
13. Stand Up Meeting

**SCRUM**

* 2-4 weeks
* Product backlog
* Spring Planning (get from backlog and choose to implement)
* Development
* Spring Review or Demo (at the end of each spring, possibly releases)
* Repeat , what is left continue to the next sprint

**Lecture 10 : Cost Estimation**

Function Points: Function points are the most important measure for the size of software and for cost estimation. It is not based on program code, but in requirements specifications

Why should costs be estimated in the beginning of a project ?

* Project planning : Deadline, staff
* Contracts with fixed prices

Cost measuring unit: **MM (MY) , Line of Code , Function Points**

**What do costs depend on ?**

* Size of problem to solve
* Quality Requirements
* Quality Developers
* Company’s experiences with similar tasks

**When should the costs be estimated ?**

* As early as possible
* At the beginning of analysis and definition phase
* After analysis and definition phase based on the final functional specification

Basic Approaches to cost estimation:

* Analogy method
* Multiplicator method
* Weighing method
* Percentage method

**Analogy method**

Comparison of development to be estimated and already finished product developments based on similarity criteria. (Criteria may include: same/similar application area , same/similar product size , same/similar degree of complexity , same/similar programming language etc)

**Multiplication Method**

* Partitioning the system into subsystems -> Cost of susbystems

**Weighing Method**

Determine influencing factors for estimation. Compute with calculation formula

**Percentage Method**

* Analysis of earlier company projects

**Function Point method**

Size of requirements

Characteristics:

* Precise (Relatively)
* Subjective
* Experiences needed for method application
* Industry standard cost estimation methods
* Precondition: commercial application

|  |  |
| --- | --- |
| Advantages (+) | Disadvantages(-) |
| Based on product requirements  Iterative method  Estimation possible at early points  Easy to learn  Precise  Tool support available | Limited only to commercial applications  Tend to underestimate |

