ANALYSIS

Inherent

(Complex) Problem domain

Stakeholders

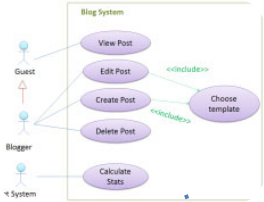
Obvious
Users, Customers
Developers, Dev Company
Government, Law Enforcement
Not so obvious

We use OO to 'align the view'.
We use UML as the standard notation.

Why can't we only ask the user?
- They are not the only stakeholder
- They can tell the problem but rarely know the solution

Establish requirements...
- brainstorming
- Focus groups
- User surveys
- Product surveys
- Observations
- Interviews
- Prototyping

user needs & justifications



Leave out certain details

Combat complexity...
Use abstraction, build models

We can divide & conquer

UML: General purpose, widely known

user-system interactions for usage scenarios

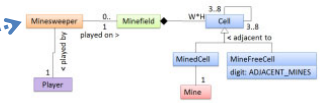
System: Square game Use case: U02 - play a game
Actors: Player (multiple players)

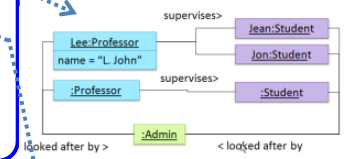
1. A Player starts the game.
2. SquareGame asks for player names.
3. Each Players enters own name.
4. SquareGame determines and shows the order of
5. SquareGame prompts for the current Player to t
6. Current Player adjusts the throw speed.
7. Square Game throws the die.

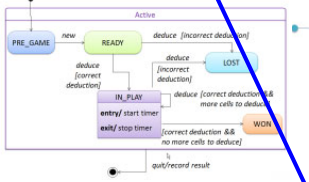
Specify requirements...
- Textual descriptions
- Feature list
- User stories
- Use cases

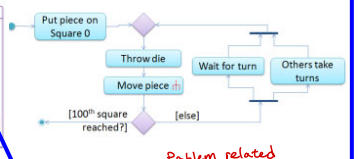
- OO domain models
- Object diagrams
- State machine diagrams
- Activity diagrams

- UI prototypes
- Glossary
- Supplementary requirements
- ...









Problem related

Possible Improvements:
- Apply OO paradigm
- Apply analysis patterns
- Categorize and prioritize requirements

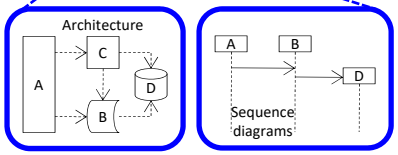
Design Patterns

Requirements Specification

What the users need

DESIGN

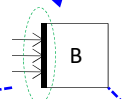
System Specification

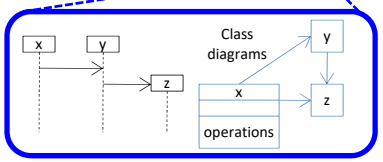


Architecture styles such as n-tier, client-server, peer-to-peer, broker, pipes-and-filters, service-oriented, transaction-processing, and MVC

Top-down design

API (Application Programming interface)





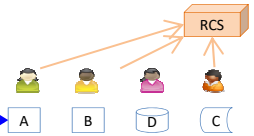
Bottom-up design

More agile -> less upfront detailed design

Detailed design

Possible Improvements:
- ↑Cohesion, ↓Coupling
- Apply design patterns (Singleton, Command, Observer, Façade, MVC)
- Apply design principles (SOLID, Law of Demeter, Separation of concerns, ...)
- Use polymorphism (inheritance, interfaces, dynamic binding)
- Good product design guidelines (e.g., usability, ...)

IMPLEMENTATION

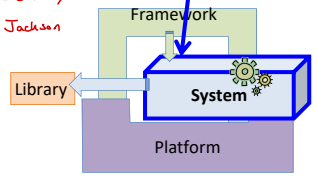


Integration

Can be top-down, bottom-up, sandwich, big-bang

Can be late-one-time, or early-and-continuous

3rd Party
Jackson
J Unit
JavaFX



Java
↳ comes with run time

Possible Improvements:
- Refactor code
- Coding standards and good coding practices
- Protect code using assertions, exceptions, logging, and defensive coding.
- Build automation

QUALITY ASSURANCE

= Validation & Verification

Unit testing

Integration testing

Developer testing

QA

System testing

User

Acceptance testing
Alpha/beta testing

Other V&V techniques:
- formal verification
- static analyzers
- code reviews
- ...

Testing

After modifications... Regression testing

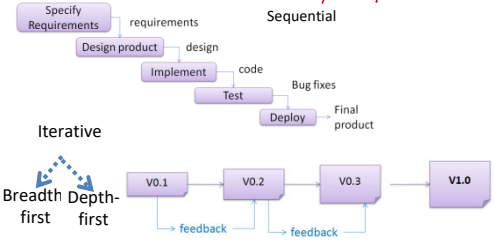
Test cases

- Made up as you go
  → Exploratory testing
- Predetermined
  → Scripted testing
- Can be created in these ways:
  - Black-box, Glass-box, Grey-box

Possible Improvements:
- Automate testing
- Use Test-Driven Development
- Increase efficiency and effectiveness of testing (Equivalence partitioning, Boundary Value analysis)
- Increase test coverage (function/entry/exit/statement/branch/condition/path coverage)
- isolate SUT using drivers, stubs and dependency injection

PROJECT MANAGEMENT

System Specification: More about the solution.



Unified process

CMMI

Agile processes

XP

Scrum

Work-breakdown structures

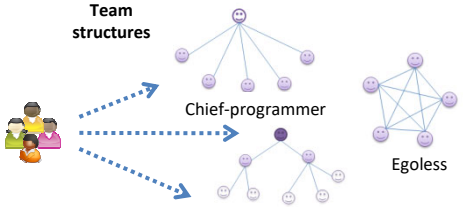
WBS1: Requirements
WBS1.1: Domain model
WBS1.2: Use cases
WBS1.2.1: Decide the list of use cases
Detail each use case
Prototypes and final

Includes buffers and milestones

Project plans

Issue trackers

Team structures



Made with Goodnotes