Role of Documentation in (OO) Software Development

Dr. Ugur Dogrusoz

Computer Eng Dept, Bilkent Univ, Ankara, Turkey
Outline

- Documenting software
  - Motivation
  - Types and purpose of documentation
  - Documentation needed for CS 491-2

- Object-oriented software engineering
  - Why building software is *real* engineering
  - Various software methodologies
  - Importance of modeling
Software Engineering

Appreciating Software Engineering

- Build complex software systems within time and budget in the context of frequent change
- SE *is* an engineering discipline: analyze, design, and then implement

Technical vs managerial knowledge (hard vs soft skills)
SE Methodologies

Every software needs
- Analysis (*what* to do)
- Design (*how* to do)
- Implementation & Testing

Process/algorithm used for development
- Waterfall
- Prototyping
- Iterative
  - Incremental
  - XP (eXtreme Programming)
- ...

Ugur Dogrusoz
Computer Eng Dept, Bilkent Univ
Motivation Behind Documentation

- **Effective use of software**
- **Communication medium** among developers
  - Working in teams unavoidable! Need to express ideas well both verbally and written
  - Soft skills (e.g. communicational skills) just as important as technical skills
  - Written ideas are more concrete and avoid ambiguities and incompleteness
  - Flow of information
    - across teams of different responsibilities
    - as team members change

Ugur Dogrusoz
Computer Eng Dept, Bilkent Univ
Types of Documentation

- Process documentation
  - Records process of development and maintenance
  - Plans, estimates, schedules, process quality docs, project standards

- Product documentation (focus of this talk!)
  - Describes a particular product being developed
  - User/exposed documentation
    - End-user documentation, administrator documentation, etc.
  - System/internal documentation (most CS491-2 docs)
    - From viewpoint of developers and maintainers
Problem Statement: report organization cs491

- Introduction
  - Description
  - Constraints (e.g. economical)
  - Professional and ethical Issues
- Requirements
- References

Typically written by the client
Modeling

- To understand
  - real-life system to be automated (model it “as is”)
  - software system to be built (model it “as you want it to be”)

- Same idea as
  - blueprints for building bridges, houses, etc.
  - layouts for manufacturing VLSI chips
Modeling

- Textual (e.g. long textual descriptions or X3D) or visual/graphical (e.g. sketches or flowchart)

- UML (Unified Modeling Language) is a visual modeling language to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development
  - Helps you model both application and software domains!
Summary of use case model: relationships between actors and use cases
Problem Statement

Requirements Elicitation

Non-functional Req.  Functional Model

Analysis

Analysis Object Model  Dynamic Model

Class Diagrams

System Design

Use Case Diagrams

Sequence Diagrams

State Diagrams

Activity Diagrams

OO Analysis w/ UML
Analysis: object model
Analysis: UML sequence diagrams
Analysis: UML activity diagrams
Analysis: user interface

User Interface

- Menus and screens/forms/dialogs/windows
- Navigational paths between menus and screens
- Mock-ups

Ugur Dogrusoz
Computer Eng Dept, Bilkent Univ
Analysis: report organization CS491

- Introduction
- Current system (if any)
- Proposed system
  - Overview
  - Functional requirements
  - Nonfunctional requirements
  - Constraints ("Pseudo requirements")
- System models
  - ...
- Glossary & References

Share w/ customer; forms a basis for a contract w/ customer!

Ugur Dogrusoz
Computer Eng Dept, Bilkent Univ
Analysis: report organization CS491

- System models
  - Scenarios
  - Use case model
  - Object model
  - Dynamic models
  - User interface

- Glossary

- References
Analysis: summary

- Application domain is modeled to fully understand the real-life system
  - “as is” (vs. “as you want it to be”)

- Resulting model
  - specifies exactly **what** the system is going to do
  - is a **contract** between developer and customer
  - is **input** to design phase
Analysis: focuses on the application domain

Design: focuses on the solution domain

- The solution domain is changing very rapidly
- Design knowledge is a moving target

what vs. how
Analysis Object Model → System Design → Subsystem Decomposition → Object Design Model → Implementation & Testing → Software

Dynamic Model

Deployment Diagrams
Component Diagrams

Design Goals

Object Design Model

Class Diagrams
Design goals: typical tradeoffs

- Functionality v. Usability
- Cost v. Robustness
- Efficiency v. Portability
- Rapid development v. Functionality
- Cost v. Reusability
- Backward Compatibility v. Readability
- Space v. Speed

A low cost system does not do much error checking (e.g., 5.00 or 5.00 Euros).

It’d be very difficult to build a real-time game that is portable.
Subsystem

- Collection of classes, associations, operations, events and constraints that are closely interrelated with each other
- Great way to handle complexity
Design: subsystem decomposition

- From objects to subsystems, taking into account the non-functional requirements
- No single/fixed algorithm
  - High coherence & low coupling
- Initial decomposition usually derived from functional requirements; constantly revised as new issues addressed
- The objects and classes from the object model could be the “seeds” for the subsystems
Component Diagram:
- Illustrates dependencies between components at design time, compilation time and runtime.
Design: UML component diagrams
Design: UML deployment diagrams

- Deployment Diagram:
  - Once we have done
    - Subsystem decomposition
    - Concurrency
    - Hardware/Software Mapping
  - Illustrates the distribution of components at run-time.
  - Deployment diagrams use nodes and connections to depict the physical resources in the system.
Design: UML deployment diagrams
Introduction
- Purpose of the system
- Design goals
- Definitions, acronyms, and abbreviations
- Overview

Current software architecture (if any)

...
High-level design: report organization CS491

…

Proposed software architecture
- Overview
- Subsystem decomposition
- Hardware/software mapping
- Persistent data management
- Access control and security
- Global software control
- Boundary conditions

Subsystem services

Glossary & References
Design: UML class diagrams

Contains all public operations of the system as a Facade class and delegates functionality as needed.

DatabaseManager

UserManager

User

- name
- ID
- password

RegistrationSystem

FormManager

Form

+ onCancel()
+ onOK()
+ logout()

DatabaseManager

UserManager

User

- name
- ID
- password

RegistrationSystem

FormManager

Form

+ onCancel()
+ onOK()
+ logout()

Courses and Sections form an example of the Abstraction-Occurrence pattern.

Course

- prereqs : Course[]
- add(Student, Section)
- drop(Student, Section)
- addPrereq(Course)
- dropPrereq(Course)
- cancel(Section)

CourseCatalog

+ courses() : List
+ instructors() : List

CourseCatalog

Professor

- student : Student
- semester : int
- course : Course
- section : Section

Professor

- student : Student
- semester : int
- course : Course
- section : Section

User

- name
- ID
- password

Student

- primaryCourses : Course[]
- alternateCourses : Course[]
- add(Course, Section)
- drop(Course)
- createBillFor()

Student

Course

Professor

Enrollment

add/drop methods of Course are examples of Delegation pattern as they are delegated to the corresponding Section method.

Course and its Sections form an example of the Abstraction-Occurrence pattern.

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog

Professor

CourseCatalog
Low-level design: report organization CS492

- Introduction
  - Object design trade-offs
  - Interface documentation guidelines
    - Collections have an iterator() method returning an Iterator
  - Engineering standards (e.g., UML and IEEE)
  - Definitions, acronyms, and abbreviations

- Packages

- Class Interfaces

- Glossary & References
Final report organization CS492

- Final software architecture
- Status
- User’s manual
- Misc.
  - Impact
  - New tools and technologies
  - Library and Internet resources used
References

- Software Documentation, Ian Sommerville, 2011
- OO Software Engineering, Using UML, Patterns, and Java, Bernd Bruegge and Allen H. Dutoit, 2010
Questions?

- Thanks for your attention!
Name and briefly explain two distinct types of documentation for software.