Software Engineering

Dr. Binu P Chacko
Associate Professor
Department of Computer Science
Prajyoti Niketan College, Pudukad, THRISSUR

Introduction

- Student s/w Vs industrial strength s/w
- Industrial strength s/w: S/W should be produced at reasonable cost, in a reasonable time (schedule), and should be of good quality
- Cost: person-months of effort (labor intensive)
- Productivity: LOC (KLOC) per person-month
- Pursuit of higher productivity is a basic driving force behind s/w engg and a major reason for using the different tools and techniques
- Unreliability of the s/w: dos and donts are different. The reason is the presence of defects in the s/w

Attributes of s/w quality

- Functionality: The capability to provide functions which meet stated and implied needs when the s/w is used
- Reliability: The capability to provide failure-free service
- Usability: The capability to be understood, learned and used
- Efficiency: The capability to provide appropriate performance relative to the amount of resources used
- Maintainability: The capability to be modified for purposes of making corrections, improvements or adaptation
- Portability: the capability to be adapted for different specified environments without applying actions or means other than those provided for this purpose in the product

s/w maintenance

- Corrective maintenance: the defects need to be corrected
- Adaptive maintenance: to satisfy the enhanced needs of the users and the environment
- Scale: Different set of methods must be used for developing large s/w
- Change: additional requirements need to be incorporated during development stage

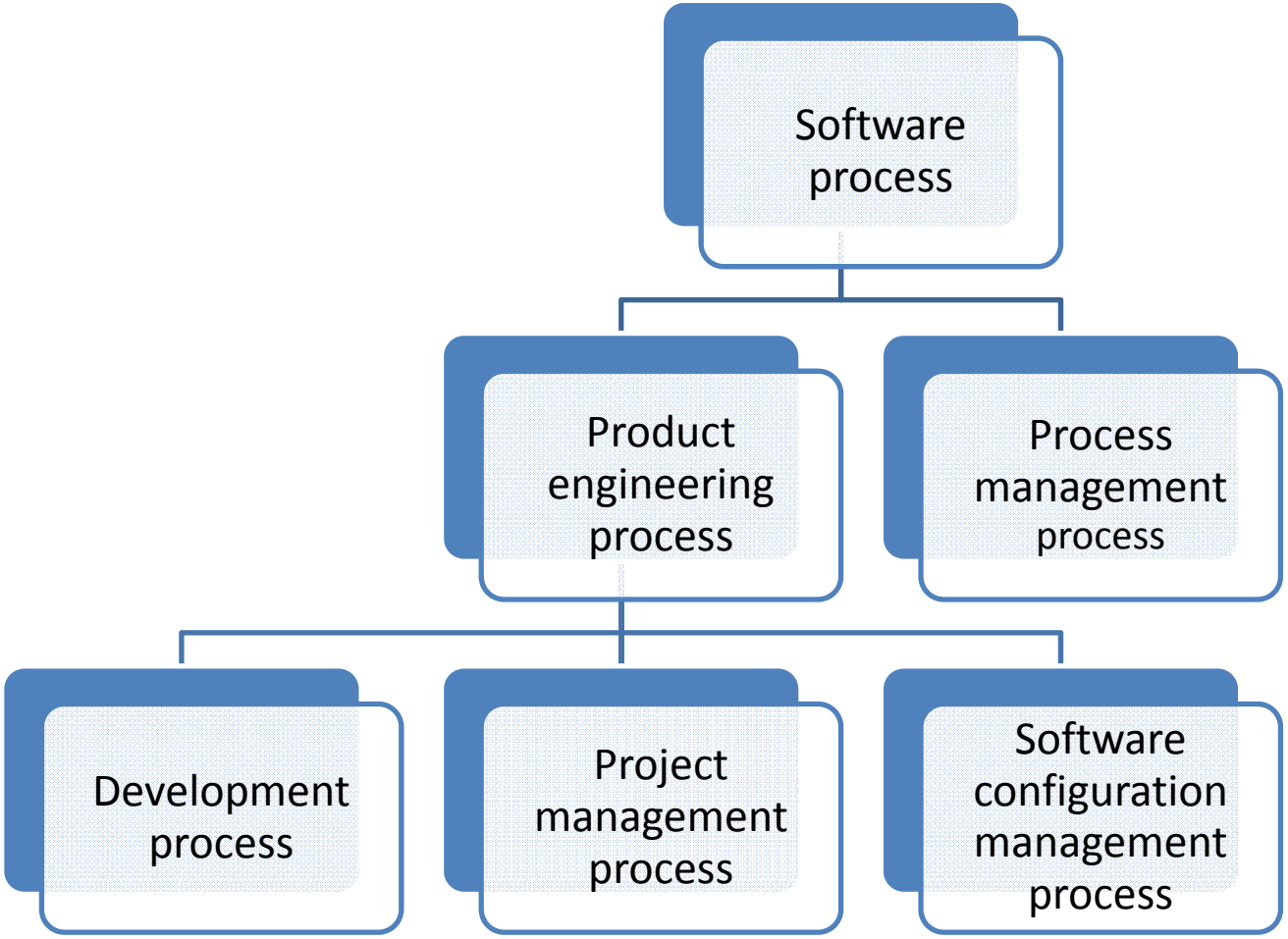
Software Process

- Software Engineering is defined as the systematic approach to the development, operation, maintenance, and retirement of software
- Final quality delivered and productivity achieved depends on the skills of the people involved in the software project, the processes people use to perform the different tasks in the project, and the tools (technology) they use
- SE focuses on the process for producing the products
- Process is a sequence of steps performed for a given purpose. A process may be divided into *subprocesses* (sequence steps for a stage)
- *Process specification is a description of the process* which presumably can be followed in some project to achieve the goal for which the process is designed
- Process model specifies a general process, which is optimum for a class of projects

Cont…

- Software process: processes that deal with technical and management issues of software development
- components – development process, project management process
- Development process specifies all the engineering activities that need to be performed by programmers, designers, testers, etc.
- Management process (by project management) specifies how to plan and control these activities so that cost, schedule, quality, and other objectives are met
- s/w configuration control process deals with managing change by configuration controller
- The above three processes comprise product engineering processes (objective is to produce the desired product)
- Process management process (by SEPG) : The whole process of understanding the current process, analyzing its properties, determining how to improve, and then affecting the improvement
- Nonsoftware process: business, social and training processes

Cont…


Software process
Product engineering process
Process management process
Development process
Project management process
Software configuration management process

s/w Development Process Models

- Defines the tasks the project should perform and the order in which they should be done
- Provides generic guidelines for developing a suitable process for a project
- *Activities*: design, coding, testing
- Waterfall model (by Royce): phases are organized in a linear order
- Feasibility analysis → Requirements analysis and project planning → System design → Coding → Testing and integration → Installation → Operations and maintenance
- Each phase deals with a distinct and separate set of concerns, and thus helps the engineers and managers to handle the complexity of the problem
- To clearly identify the end of a phase and the beginning of the next, some certification mechanism (verification and validation) has to be employed at the end of each phase

Cont…

- A good plan is based on the requirements of the system
- o/p of each phase is called work product which is in the form of *documents* (requirements document, Project plan, design document, test plan and test reports, final code, software manuals)
- Adv: simplicity – divides the large task into separate phases – each phase deals with separate logical concern
- Limitations: for new systems, user does not know the requirements
- A large project might take few years to complete – h/w technology changes
- The entire s/w is delivered in one shot at the end
- All requirements must be stated at the beginning
- Requires formal documents at the end of each phase

Prototyping

- A throwaway prototype is built to help understand the requirements. This prototype is developed based on the currently known requirements
- Good idea for complicated and large systems
- Activities: design, coding, testing
- Develop requirements specification document → allow the clients to use and explore the prototype → give feedback to the developer → modify the prototype based on the feedback → repeat this cycle
- Reduce the cost for prototyping – include only valuable features, focus on quick development rather than quality, reduce testing
- Benefits: reduce the cost of actual s/w development, fewer changes in the requirements, quality of the s/w will be improved, developing a prototype mitigates many risks

Iterative Development

- s/w should be developed in increments
- A simple initial implementation is done for a subset of the overall problem
- A project control list is created that contains all tasks that must be performed to obtain the final implementation. Each task should be performed in one step of the iterative enhancement process
- Designing the implementation for the selected task → coding and testing the implementation → performing an analysis of the partial system and updating the list as a result of the analysis. The process is iterated until project control list is empty
- Another approach: do requirements and architecture design in a standard waterfall or prototyping approach, but deliver the s/w iteratively. Adv: feedback from an iteration can be incorporated in the next iteration
- Popularity due to – after each iteration, some working s/w is released. Changing requirements can be incorporated in the s/w. Helps in developing stable requirements for the next iteration

Rational Unified Process

- Iterative model by Rational (part of IBM)
- Designed for object oriented development using UML
- Development of a s/w is divided into cycles, each cycle delivers a fully working system (adds some capability to the existing system)

Each cycle is broken into:-

- Inception phase: establish goals and scope of the project. Completion: lifecycle objectives
- Specify vision and high level capability of the system, expected benefits, plan (cost & schedule), risks
- Based on the o/p of this phase, a go/no-go decision is taken
- Elaboration phase: design the architecture of the system. Completion: lifecycle architecture
- Take decision regarding technology and architecture
- *o/p*: technical evaluation of the proposed solution, cost/benefit analysis

Cont…

- Construction phase: build and test the s/w
- Deliver the s/w along with manuals. Completion: initial operational capability
- Transition phase: move the s/w from development environment to client's environment
- Additional testing, training, conversion of old data to this s/w.
- Completion: product release
- Subprocesses: requirements, analysis and design, implementation, test, deployment, project mgt, configuration mgt
- Difference: RUP separated the phases from tasks and allows multiple subprocesses to function within a phase. Provides flexible process model

Timeboxing Model

- Parallelism b/w different iterations is employed. Development of a new release happens in parallel with the development of the current release
- Reduces average delivery time for iterations by utilizing additional manpower
- Basic unit of development is a time box of fixed duration. Requirements or features should be fit into the time box
- Each time box is divided into a sequence of stages – each stage performs some clearly defined task for the iteration – duration of each stage is same – separate team for each stage
- A time box consisting of 3 stages: requirement specification, build, deployment
- If time box size is T days, first s/w delivery after T days, subsequent deliveries after every T/3 days
- Suitable for project with large number of features
- Increased complexity of the project mgt

Agile Processes

Agile approaches are based on

- Working s/w is the key measure of progress in a project
- s/w should be developed and delivered rapidly in small increments
- Late changes in the requirements should be entertained
- Face-to-face communication is preferred over documentation
- Continuous feedback and involvement of customer is necessary
- Simple design which evolves and improves with time is a better approach
- Decide delivery dates
- Deliver high quality s/w at low cost and cycle time

Agile Methodology – eXtreme Programming

- Changes are inevitable rather than treating changes as undesirable – for this, development process has to be lightweight and quick to respond – develop s/w iteratively
- XP starts with user stories (no detailed requirements)
- Using time estimate and stories release planning is done
- Bugs found during acceptance testing for an iteration can form work items for the next iteration
- An iteration starts with iteration planning. High-value and high-risk stories are considered as higher priority and implemented in early iterations
- Development is done by pairs of programmers
- Test-driven development: code should be written to pass the test before the actual code is written
- Refactoring to improve the design, and then use refactored code for further development. During refactoring, no new functionality is added
- Encourages frequent integration of different units

Project Management Process

- How long each phase should last, how many resources should be assigned to a phase, how a phase should be monitored
- Basic task is to plan the detailed implementation of the process for the particular project and then ensure that the plan is properly executed

Activities

- Planning: develop a plan for s/w development
- Cost estimation, schedule and milestone determination, project staffing, quality control plans, controlling and monitoring plans
- Project monitoring and control includes all activities the project management has to perform while the development is going on
- Monitor potential risks for the project, interpretation of the information
- Termination analysis is performed when the development process is over
- Provide information about the development process and learn from the project to improve the process

s/w Requirements Specification

- Major parties interested in a new system: need of a client, created by developer, used by end user
- Purpose: to bridge communication gap b/w client and developer
- Advantages: SRS establishes the basis for agreement b/w client and supplier
- SRS provides a reference for validation of the final product. An error in SRS will reflect in the final product also
- High quality SRS is a prerequisite to high quality s/w.
- High quality SRS reduces the development cost

Requirement Process

- Sequence of activities that need to be performed in the requirements phase

Tasks

- Problem analysis: starts with problem statement
- Problem domain and environments are modeled, constraints on the system, i/p, o/p
- Meeting of analyst, client and end users
- Documents describing the work or organization and o/p from existing methods may be given to SA
- Role of SA: listener, seek clarification, verification, document information, build some models, brainstorming, explain to client what he understands
- Requirements specification
- Requirements validation: to verify and ensure quality
- Requirements process terminate with the production of validated SRS

Requirements Specification

- Modeling is a tool to get complete knowledge about the proposed system
- Modeling focuses on problem structure, not its external behavior, performance constraints, design constraints, standards compliance or recovery
- SRS is written based on knowledge acquired during analysis

Characteristics

- Correct
- Complete
- Unambiguous: if every requirement specified has one and only one interpretation
- Verifiable
- Consistent
- Ranked for importance and/or stability

Components

- Functional requirements specify the expected behavior of the system – which o/p should be produced from the given i/p
- Relationship b/w i/p and o/p, system behavior in abnormal situations
- Description of i/p and their source, units of measure, range of valid i/p, operation to be performed on i/p
- Performance requirements specifies performance constraints on the s/w system
- Types: static, dynamic
- Static (capacity) requirements don't impose constraint on the execution characteristics of the system. E.g. number of terminals to be supported
- Dynamic requirements specify constraints on execution behavior (response time and throughput) of the system.
- Acceptable range of different performance parameters, acceptable performance for both normal and peak workload conditions

Cont…

Design constraints

- Standards compliance: report format and accounting procedures
- h/w limitations: type of machines used, OS, languages supported, limits on primary and secondary storage
- Reliability and fault tolerance: What the system should do if some failure occurs
- Security: restrictions on the use of certain commands, control access to data, provide different kinds of access requirements for different people, use of passwords and cryptography techniques, maintain log of activities in the system

External interface: interactions of s/w with people, h/w and other s/w

- Create preliminary user manual

General Structure of SRS

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definition, Acronyms, Abbreviations
   - 1.4 References
   - 1.5 Overview
2. Overall description
   - 2.1 Product perspective
   - 2.2 Production functions
   - 2.3 User characteristics
   - 2.4 General constraints
   - 2.5 Assumptions and dependencies
3. Specific requirements

- No one method is suitable for all projects

Cont…

- Section 1: clarify motivation, business objectives and scope of the project
- Section 2: how the system fits into larger system, an overview of all the requirements
- Product perspective: relationship of the product to other products
- Schematic diagrams showing different functions and their relationships with each other can be given

Organization of Specific Requirements

3. Detailed requirements
   - 3.1 External interface requirements
     - 3.1.1 User interface
     - 3.1.2 H/W interface
     - 3.1.3 s/w interface
     - 3.1.4 Communication interface
   - 3.2 Functional requirements
     - 3.2.1 Mode 1
       - 3.2.1.1 Functional requirement 1.1
       - :
       - 3.2.1.n Functional requirement 1.n
     - :
     - 3.2.m Mode m
       - :
   - 3.3 Performance requirements
   - 3.4 Design constraints
   - 3.5 Attributes
   - 3.6 Other requirements

Cont…

- Communication interface specify the communication with other systems
- Section 3.2: functional capabilities of all modes of operation
- Required i/p (source of i/p, units of measure, valid ranges, accuracies), desired o/p, and processing requirements have to be specified
- Section 3.3: specify both static and dynamic performance requirements

Functional Specification with Use Cases

- Use cases specify functionality of a system by specifying the behavior of the system
- Used to describe the business processes of larger business or organization
- Actor: person or system which uses the system for achieving some goal
- Primary actor: main actor who initiates a UC for achieving a goal
- Some agent may actually executes UC for primary actor
- E.g. time-driven trigger
- Scenario: describes a set of actions performed to achieve a goal under some specified conditions
- A step in a scenario is a logically complete action by the actor (enter information), some logical step that the system performs (deliver information), or an internal state change by the system (update the record)

Cont…

- Main success scenario: describes the interaction in which all steps in the scenario succeed
- Extension (exception) scenarios: describe the system behavior if some of the steps in the main scenario don’t complete successfully
- Use case is a collection of all success and extension scenarios related to the goal
- Use case also specifies a scope
- Scope of system use case is the system being built; scope of a component use is a subsystem
- Adv: use cases focus on external behavior, thereby avoiding doing internal design during requirements

Developing Use Cases

- Actors and goals: specifies actors for each goal. Also scope of the system and overall view
- Main success scenarios: system behavior for each use case is specified
- Failure conditions: identify all possible failure conditions
- Failure handling: what should be the behavior under different failure conditions

Validation

- To ensure that the SRS reflects the actual requirements accurately and clearly
- To check that the SRS document is itself of good quality

Types of errors

- Omission: some user requirement isn't included in SRS
- Inconsistency: due to contradictions within the requirements themselves or due to incompatibility of the stated requirements with the actual requirements of the client or with the environment in which the system will operate
- Incorrect fact: some fact recorded in SRS isn't correct
- Ambiguity: some requirements have multiple meanings
- Validation should focus on uncovering the above types of errors
- Inspection of SRS (or requirements review) is the most common method for validation
- Review group: author of requirements document, someone who understands the needs of the client, a person from design team, person responsible for maintaining the requirements document, s/w quality engineer

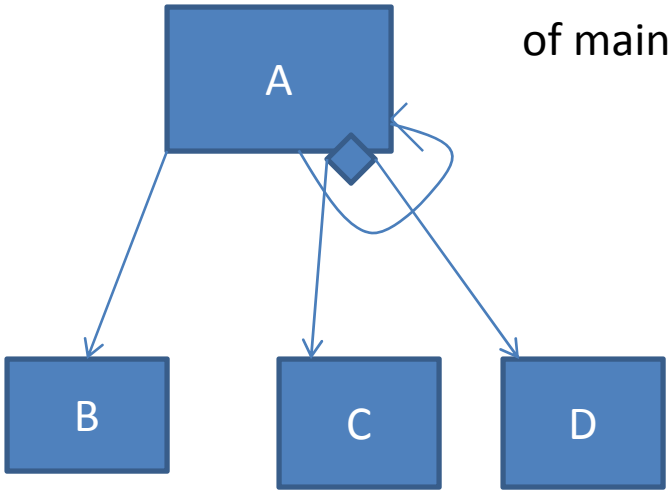
Function Oriented Design

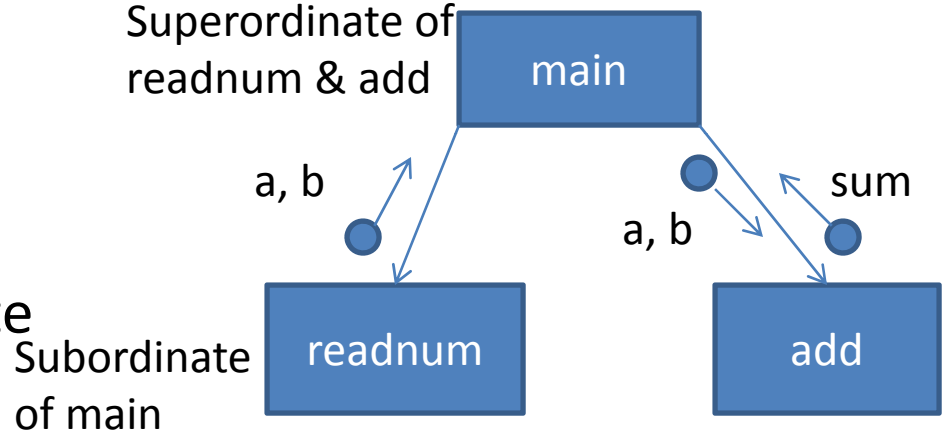
- Structure chart: used to represent the design graphically
- Major loops and decisions

can also be represented

in a structure chart

- i/p module, o/p module, transform module, coordinate module





Structured Design Methodology

- Principle: problem partitioning at top level into subsystems – one for i/p, one for o/p, one for transformation
- i/p modules have to deal with issues of screens, reading data, formats, errors, exceptions, completeness of info, structure of info
- o/p modules have to prepare o/p in presentation formats, make charts, produce reports

Restate the problem as a DFD

- Construct DFD – develop a model of the system – shows major transforms or functions in the s/w

Identify the i/p and o/p data elements

- In most cases, i/p is first converted into a form on which transformation can be applied. Transformation module often produce o/p that have to be converted into desired o/p
- Goal: separate the transforms in DFD that convert i/p or o/p to the desired format from the ones that perform the actual transformations

Cont…

- Identify highest abstract level of i/p and o/p
- Most abstract i/p data elements: data elements in the DFD that are farthest removed from physical i/p but are still i/p to the system. Data elements obtained after error checking, data validation, proper formatting, and conversion
- Most abstract o/p data elements: data elements that are most removed from the actual o/p but still considered as outgoing. These data elements are considered logical o/p data items. Transforms convert logical o/p into the required o/p form
- Central transforms: take the most abstract i/p and transforms it into most abstract o/p

First level factoring

- Results in very high level structure – i/p, o/p and transform modules
- Specify main (coordinate) module whose purpose is to invoke the subordinates

Cont…

Factoring of i/p, o/p and transform branches

- To simply these branches and distribute their work
- i/p module is considered as main module, and a subordinate i/p module is created for each i/p stream
- o/p module is created for each data stream
- Determine subtransforms that will together compose the overall transform and then repeat the process for newly found transforms until we reach the atomic modules

Object Oriented Design

- Permit changes more easily
- New applications can use existing modules more effectively, thereby reducing development cost and cycle time
- Classes and Objects are the basic building blocks of OO design. Object is the basic runtime entity.
- Abstraction, encapsulation, interface
- In an object, the state is preserved and it persists through the life of the object
- The state and services of an object together define its behavior

Relationship among objects

- A link exists b/w objects if an object uses some services of the other object
- Links b/w objects capture client-server type of relationship
- Aggregation generally implies containment. If an object A is an aggregation of object B, then object B will generally be within object A. A contained object can't survive without its containing object

Inheritance

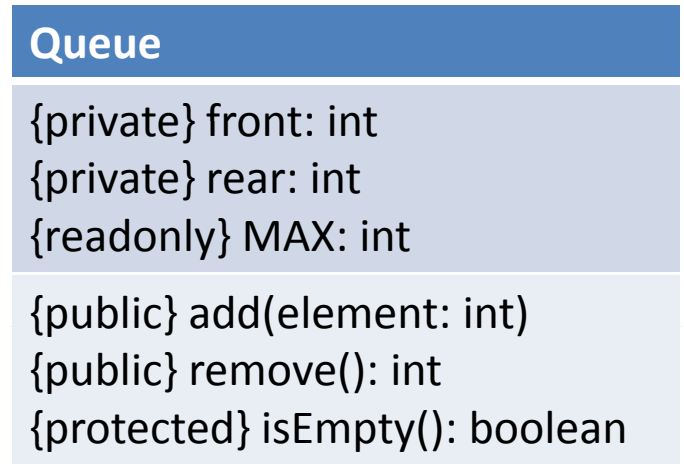
- Strict inheritance: a subclass takes all features of parent class + additional features
- Nonstrict inheritance: subclass doesn't have all the features of parent class or some features are redefined
- Polymorphism: an entity has a static type and a dynamic type. Static type of an object is the type of which the object is declared in the program. Dynamic type can change from time to time and is known only at reference time.
- Polymorphism requires dynamic binding (code associated with a given procedure call isn't known until the moment of call)
- Dynamic binding reduces the size of code

UML

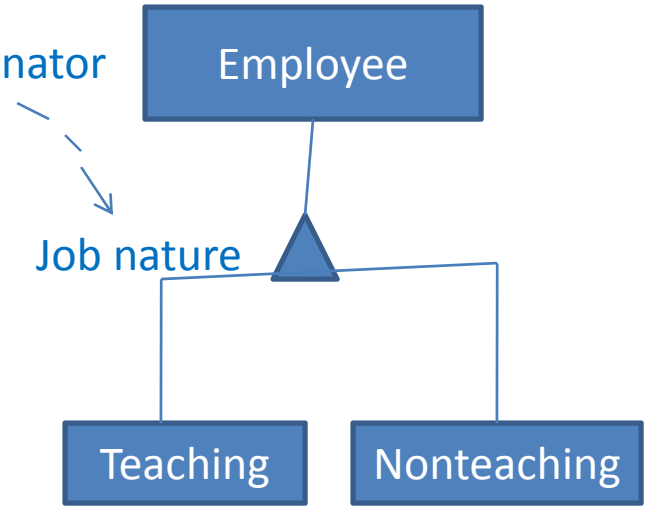
- A graphical notation for expressing OOD

Class diagram

- Defines classes, association b/w classes, and subtype, supertype relationship

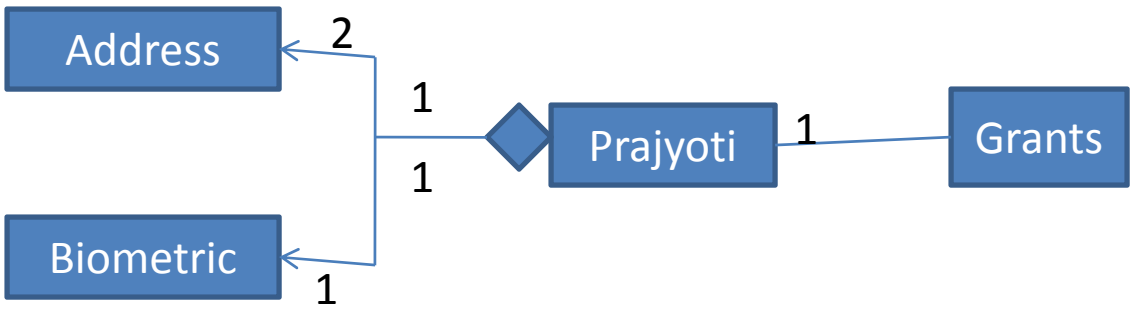


- Generalization-specialization relationship: Properties of general significance are assigned to superclass, while properties which specialize an object are put in subclass. A subclass contains its own properties as well as those of superclass

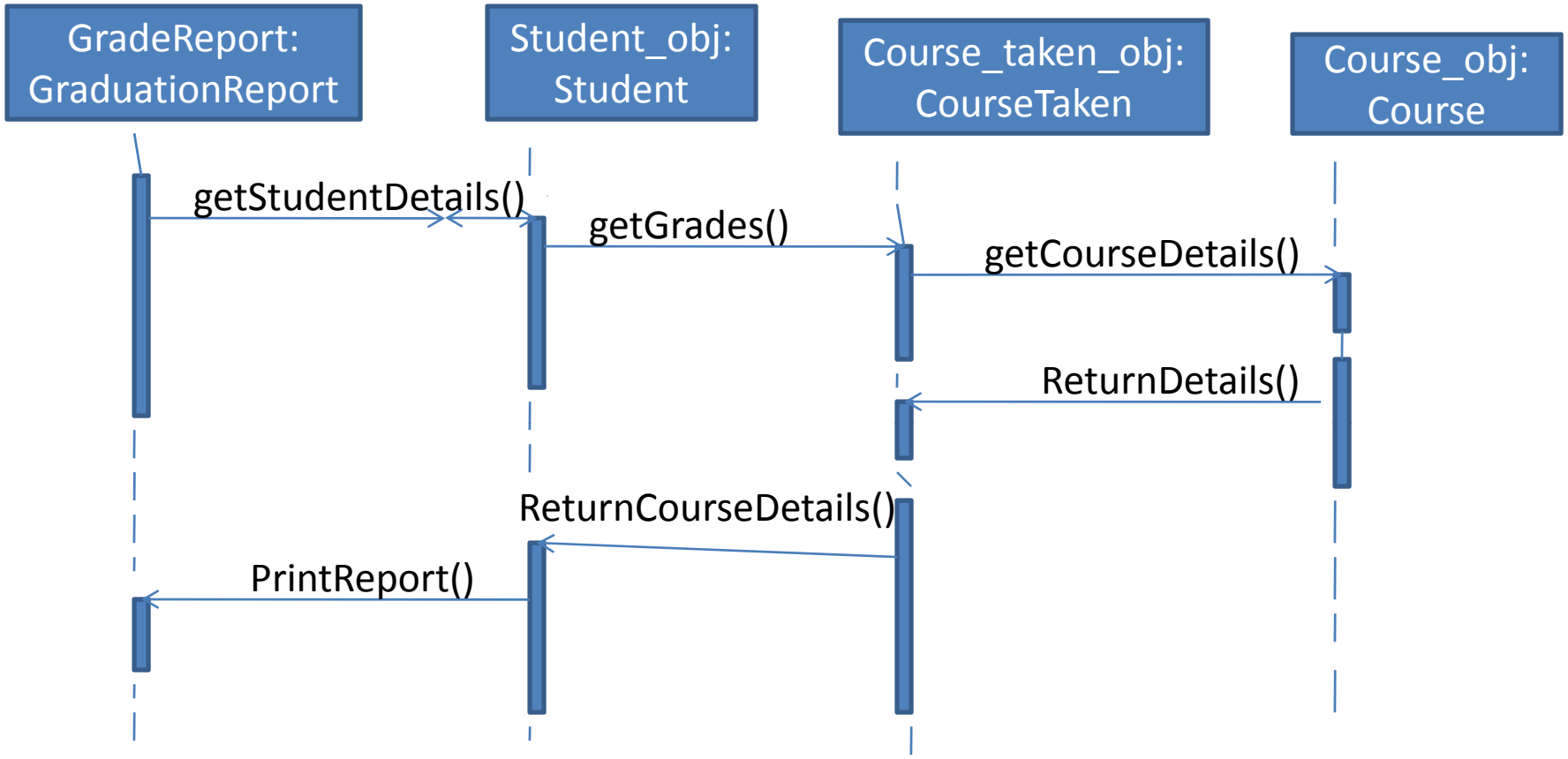


Cont…

- Association (one-to-one, one-to-many, etc.): object of one class needs services of objects of other class
- Part-whole relationship: an object is composed of many parts, each part itself is an object – containment or aggregation

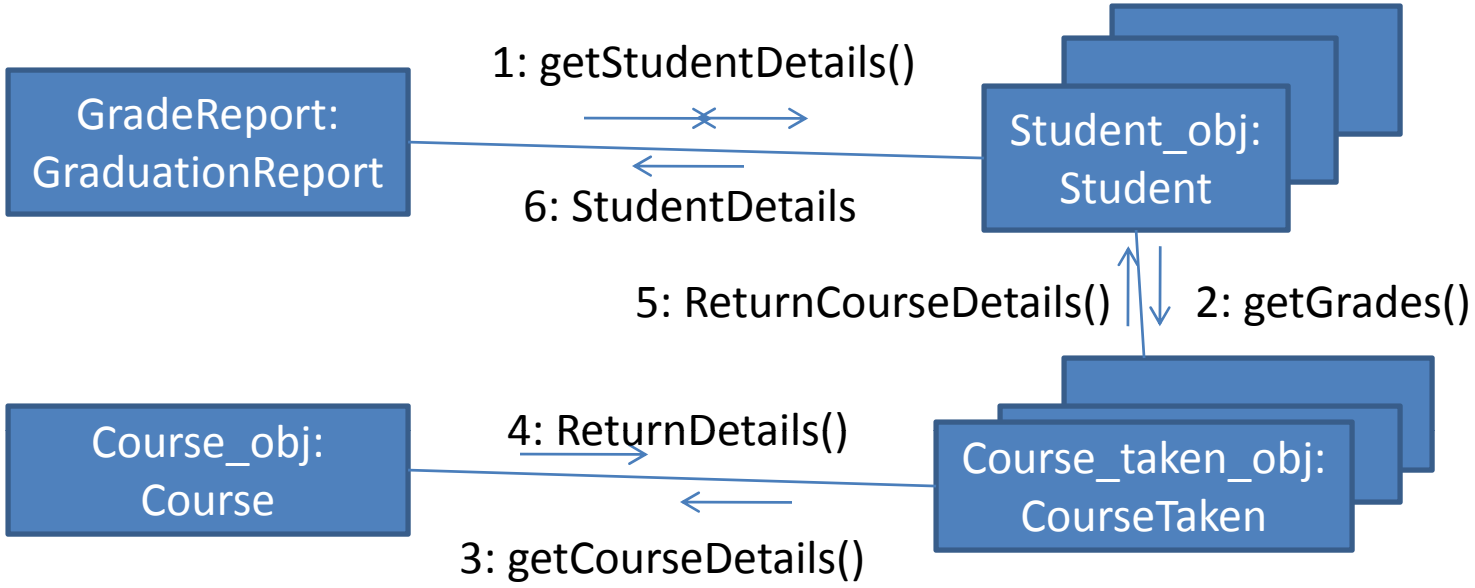


Sequence Diagram



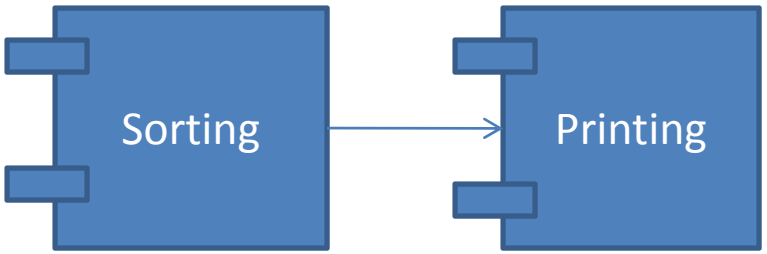
- Shows the series of messages exchanged b/w objects and their temporal ordering
- To model the interaction b/w objects for a particular use case
- Vertical bar: lifeline of an object, arrow: message from one object to another

Collaboration Diagram



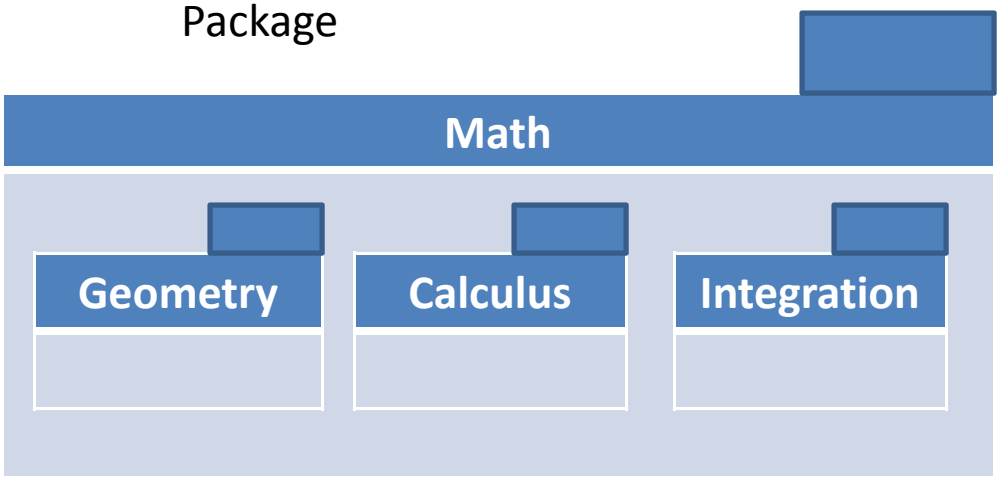
- Shows how objects communicate
- Chronological ordering of messages is given by message numbering
- Class diagram captures the structure of system's code, whereas many diagrams are needed to model the dynamic behavior of the system

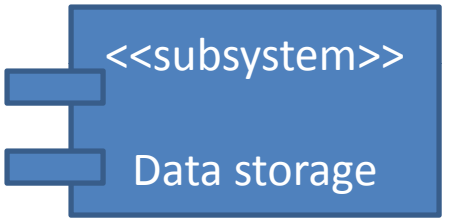
Other Diagrams



Component - Connector

Package





Subsystem

- Components often encapsulates subsystems and provide clearly defined interfaces through which components can be used by other components in the system
- Packages are formed by combining many classes
- State diagram: used to model the behavior of objects. How an object evolves as operations are performed on it
- State represents different states of the object; transition captures the performing of different operations on that object
- Activity diagram: for modeling dynamic behavior. Model a system by the activities

Design Methodology

- During architecture design, system is divided into high level subsystems or components

OOD consists of:-

- Identifying classes and relationships: requires identification of object types, structures b/w classes, attributes of classes, association b/w classes, and the services provided
- While identifying attributes, new classes may be defined or old classes may disappear
- Structure must reflect the hierarchy in the problem domain
- For associations, we need to identify relationship b/w objects. Object of Company may be related to object of Person by “employee” relationship

Cont…

- Dynamic modeling: aims to specify how the state of various objects changes when events occur
- Events are interactions with the outside world and object-to-object interactions
- Each event has an initiator and a responder. Internal events – both initiator and responder are within the system. External event – initiator is outside the system. E.g. user or sensor
- From scenarios different events performed on objects can be identified, which are then used to identify services on objects. Different scenarios together characterize the behavior of the system
- Modeling scenario triggered by external events. First model the main scenario, then the exceptional ones

Cont…

- Functional modeling: how to compute o/p from i/p
- It is useful in cases where i/p – o/p mapping involves many steps
- It is represented by DFD. Most of the processing is done by operations on classes
- Defining internal classes and operations: each class is critically evaluated to see if it is needed in its present form in the final implementation
- Implementation of operations on classes: rough algorithms for implementation might be considered. A complex operation may get defined in terms of lower level operations on simpler classes
- Optimize and package: issue of efficiency
- Final structure shouldn't deviate too much from logical structure produced

Verification

- o/p of the design activity should be verified before proceeding to the next phase
- Check for internal consistency. E.g interface of a module is consistent, data usage is consistent with declaration
- Most common approach for verification is design review. Purpose is to detect errors and to ensure that design satisfies the requirements and is of good quality
- Review group: member of system design team and detailed design team, author of requirements document and author responsible for maintaining design document, s/w quality engineer
- This team should reveal design errors or undesirable properties
- Design error: design doesn't fully support the requirements
- Design quality: modularity, efficiency

Coding

- Goal: implement the design. Try to reduce cost of the later phases
- Coding (less cost) affects both testing and maintenance (costly)
- Write code quickly that are easy to read, modify and understand
- Programming is a skill acquired by practice. It is independent of programming language, although well structured programming languages make the programmer's job simpler
- Structured programming: "goto-less" programming
- A program has a static structure (text of the program) and a dynamic structure (execution sequence of statements)
- Sequence in static structure of a program is fixed, while it will differ from execution to execution in dynamic structure
- Closer the corr. b/w execution and text structure, program is easy to understand. Goal is to ensure that the static structure and dynamic structure are same
- State the i/p conditions in which program is to be invoked – precondition of the program
- Expected final state of the program is called post condition of the program
- Program verification: determine precondition for which post condition will be satisfied

Cont…

- Linearizing the control flow by using control structures. Selection, iteration, sequencing
- Information hiding: information captures in the data structures should be hidden from the rest of the system, and is visible only to the access functions in the data structure. Rest of the modules in the s/w use these functions to access and manipulate data structures
- Make the system more maintainable. Manage the complexity of developing s/w

Programming Practices

- Control constructs
- Gotos
- Information hiding
- User defined types
- Nesting
- Module size
- Module interface
- Side effects
- Robustness: a program is robust if it does something planned even for exceptional conditions

Cont…

- Switch case with default
- Empty catch block: an exception is caught, but if there is no action, it may represent a scenario where some of the operations to be done aren’t performed
- Empty if, while statement
- Read return to be checked
- Return from finally block: should be avoided
- Correlated parameters
- Trusted data sources: check i/p data if it is from a user or network
- Give importance to exceptions

Coding Standards

- Provides rules and guidelines for programming in order to make the code easier to read

Naming conventions

- Package names should be in lowercase
- Type names should be nouns starting with uppercase
- Variable names should be nouns starting with lowercase
- Constant names should be in uppercase
- Method names should be verbs starting with lowercase
- Private class variables should have _ suffix
- Variables with large scope should have long names; small scope – short names; loop variables - i, j, k, etc.
- Prefix *is* with boolean methods and variables. Avoid negative boolean variable names (isNot)
- The term *compute/find* can be prefixed with methods related to that operation
- Exception classes should be suffixed with *Exception*

Cont…

- Files: File extension .java
- Each file should contain one outer class and the class name should be same as the file name
- Line length should be limited to 80 columns, avoid special characters
- Comments: should explain what the code is doing or why the code is there
- Prologue (comments for a module) describes functionality and purpose of the module, its public interface and how the module is to be used, parameters of the interface, assumptions it makes about the parameters, and any side effects it has
- Single line comments for a block of code should be aligned with that block
- There should be comments for all major variables
- Block of comments within /* and */
- Trailing comments after statements should be short

Cont…

- Statements: declare variables with smallest possible scope, initialize with declaration
- Declare related variables in a common statement
- Class variables shouldn’t be declared as public
- Avoid complex conditional expressions
- Layout: how a program should be indented, how it should use blank lines, white space, etc. to make it more readable

Coding Process

- Incremental coding process: write code for implementing part of the functionality of the module. Compile and test the code. If it is success, add further functionality to it
- Adv: detect error as and when a new functionality is added
- Test driven development: a new approach used in extreme programming (XP)
- A programmer first writes the test scripts and then writes the code to pass the tests. Tests being written based on specifications. The whole process is done incrementally
- First few test cases are likely to focus on main functionality. Code for higher priority features will be developed earlier
- It is the task of test cases to check that the code contained all the required functionality
- Reduce interface error
- It is tedious to write test cases for all scenarios or special conditions
- Pair programming: written by a pair of programmers – one person will type the program while other will review it
- DisAdv: loss of productivity (2 people assigned to a task), issue of accountability and code ownership when pairs aren't fixed

Unit Testing

- For code (of the module) verification
- A unit may be a function / a small collection of functions for procedural languages or a class / a small collection of classes for OO languages
- Programs are executed with some test cases. If the behavior isn't as expected, programmer finds the defect in the program and fixes it (debugging)

Approaches

- Testing procedural units: behavior of the module depends on the value of its parameters as well as the overall state of the system
- If a module call other modules (i.e. a module has other modules below it in structure chart):-
  - Bottom-up approach: first test the modules at the bottom of the structure chart and then move up
  - Write stubs: stubs are throwaway code written for the called functions to facilitate testing of the caller function

Cont…

- Testing of a module involves:- set system state as needed by the test case
- Set value of parameters
- Call procedure with parameters
- Compare results of the procedure with expected results
- Declare whether the test case has succeeded or failed
- The test suit succeeds if all the test cases succeed. If a test case fails, then the test framework will decide whether to stop or continue execution
- *Testing frameworks*: CuTest, Cunit, Cutest, Check
- Unit testing of classes: create object of the class, take object to a particular state, invoke a method on it, and check whether the state of the object is as expected. Do it for all methods
- *Framework*: Junit

Code Inspection

- Applied at unit level
- Goal: To improve quality (efficiency, compliance to coding standards, etc.) and productivity
- Static testing – detect defects not by executing the code but through a manual process

Characteristics

- Conducted by programmers for programmers
- A structured process with defined roles for the participants
- Focus is on identifying defects, not fixing them
- Inspection data is recorded and used for monitoring the effectiveness of the inspection process
- Moderator has to ensure that the review is done in a proper manner and all steps in the review process are followed

Stages

- Planning: prepare for inspection – form the inspection team with a moderator
- *Entry criteria* – code compiles correctly and the available static analysis tools have been applied
- A package consisting of the code, specifications for which code was developed, and a checklist is prepared and distributed to the inspection team
- Author may give a brief overview of the product and the special areas to be noticed
- Self review: by reviewers in a continuous time span of < 2 hours
- Self preparation log: go through the code and logs all defects and mark them on the work product itself. Also prepare a summary of self review and the time spent

Cont…

- Group review meeting: *purpose* – come up with final defect list based on the initial list of defects reported by the reviewers and the new ones found during discussion in the meeting
- *Entry criterion* – moderator is satisfied that all the reviewers are ready for the meeting
- *o/p* – defect log and defect summary report
- Team (*reader*) goes through the code line by line. If any reviewer raises an issue, there will a discussion on it. The author accepts the issue as a defect or give clarification to it.
- Team (*scribe*) records the defects in the log. At the end of the meeting, scribe reads out the defects for final review
- Review team identify the defects; author should have a solution for them
- Summary report: describes the code, total effort spent and its breakup in different review process activities, type, size and number of defects found for each category