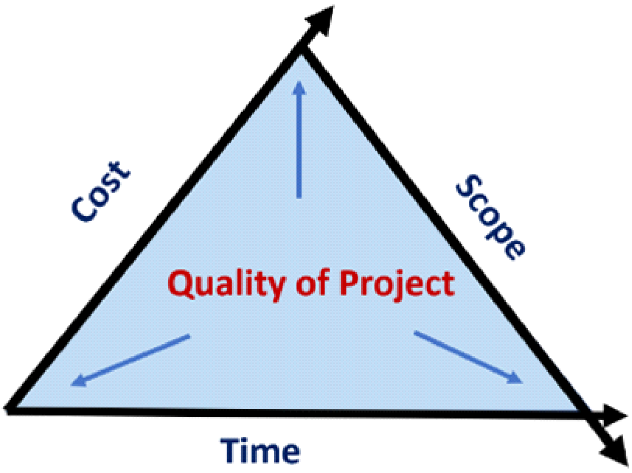
Unit 2 Speedrun

11 December 2024 17:00

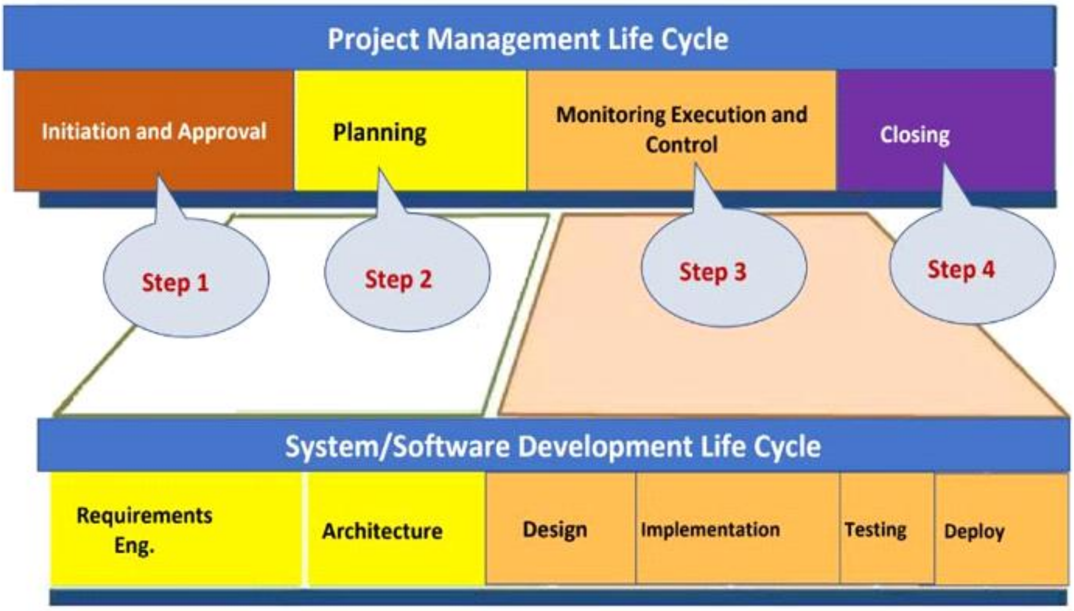
Software Project Management

- **Project** = temporary effort to create value through product, service, or result
  - All have beginning + end + budget + schedule
- **Project management =** use of knowledge, skills, tools, techniques to deliver something of value
- **Software project management =** art of planning and leading sw projects
  - Subdiscipline of project management
  - Software projects are:
    - Planned
    - Implemented
    - Monitored
    - Controlled
- **Software project manager's job:** Leader + liaison + mentor
  - Define project requirements
  - Build project team
  - Blueprint for project
  - Communicate goals of project to team
  - Budget allocation for tasks
  - Ensures expectations are met
- **Project quality:**
  - Objective = achieve project goals and targets while considering scope, time, cost



  - Balance all three for high-quality final product = project manager's job

Software project management lifecycle

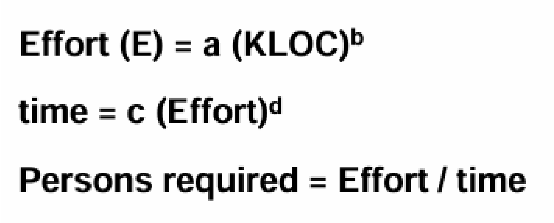


1. **Initiation and approval**
    - ○ Opportunity for project is identified and project is kicked off
    - ○ Initiation is at the approval of the feasibility study
    - ○ Actions in this phase:
        - ▪ Project charter (vision, objectives, scope, deliverables) created
        - ▪ Detail out the responsibilities
        - ▪ Project owner/manager identified
        - ▪ Initial budget identified
        - ▪ Identification of resources
2. Project planning
    - ○ **Outcome:**
        - ▪ Project plan
        - ▪ WBS
        - ▪ Schedule
        - ▪ Resource management
        - ▪ Plan for communication
        - ▪ Risk management
        - ▪ Roles
    - ○ **Actions in this phase:**
        - ▪ **Understanding project**
            - ○ Perspectives:
                - • Execution stakeholder-
                    - □ Lifecycle to be followed
                    - □ How to prioritize requirements
                    - □ Project organization
                    - □ Standards, guidelines, and procedures
                - • Sponsor
                - • Customer
        - ▪ **Estimate effort:** Work Breakdown Structure: CoCoMo (Constructive Cost Model)
            - ▪ Regression model based off lines of code
            - ▪ Procedural cost estimate model
            - ▪ Outcome of cocomo = parameters that define quality of product
            - ▪ Key parameters:
                - • Effort = amt of labor to complete a task (person-months)
                - • Schedule = amount of time required for completion of job
            - ▪ Cocomo has different models too that can be applied to a variety of projects

Organic	Team is small, problem is understood and prev solved, members have prev experience
Semi-detached	Lies between organic and embedded

Embedded	Highest level of complexity, creativity, and experience. Large team with experienced people

        - So there's three types of cocomo:
            - Basic
            - Intermediate
            - Advanced
        - Estimation formula:



$$\text{Effort (E)} = a\,(\text{KLOC})^{b}$$

$$\text{time} = c\,(\text{Effort})^{d}$$

$$\text{Persons required} = \text{Effort} / \text{time}$$

        - Value of a, b, c, d varies with type of project
        - You can use other estimation approaches too- wideband delphi, function/feature point analysis
    - **Scheduling**
        - Activity that distributes estimated effort across planned project duration
        - Evolves over time
        - Early: Macroscopic- major framework activities
        - When project starts, it's refined
        - Basic principles:
            - Compartmentalization
            - Interdependency
            - Time allocation
            - Effort validation
            - Defined responsibilities
            - Defined outcomes
            - Defined milestones
    - **Risk Management**
        - Helps manage and understand uncertainty
        - Done by everyone in the software process
        - Recognizing whatever can go wrong = risk identification
        - After identifying, rank by probability and impact
        - Then develop plan
        - <u>Risk mitigation, monitoring and management plan made</u>
    - **Quality management**
        - Done by everyone
        - Reduces rework and costs and time to market
        - RACI MATRIXXXXXXX (responsible, accountable, consulted, informed)

Responsible	Designates task to person or group
Accountabl e	Delegates and reviews the work involved in a project (only 1)
Consulted	Provide input and feedback
Informed	Looped into progress when req

- **Steps:**
    1. Develop quality management process
        - Plan for progress tracking
        - Communication plans
        - QA plans
        - Test completion criteria
        - Early verification
    2. Plans for tracking project and delivery plan
        - Plan for management
        - Procedure for product release
        - Staff management

        - Performance management
        - Compensation

3. Project monitoring and control
    - Ensure that the project is on track
        - Project monitoring = using continuously collected quantitative data
        - Project control = making decisions or adjustments for the project
    - Decisions and adjustments in dimensions like:

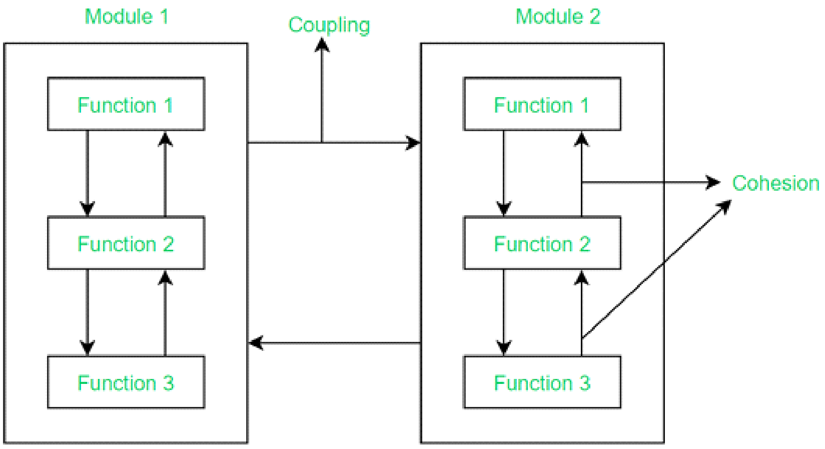
Cost and Infra	• Personnel, capital, expenses
Quality	• Designed in, not afterthought • Requirements may conflict • Leading indicators: point towards future • Lagging indicators: pattern in progress confirmation
Organization	• Structure, roles, responsibilities from team
Time	• Number of person months • Brooks' law- add people to late project, make it later
Information	• Availability, propagation, communication, documentation • Agile = less focus on explicit docu

    - Monitor and control project work
        - Collect, measure, disseminate performance info
        - Corrective or preventative actions
        - Do Critical Path analysis
    - Ensure all change controls are followed
    - Ensure scope and deliverables are updates
    - Control the quality triangle
    - Gantt chart:
        - Show activities displayed against time
        - Critical path = phenomenon = individual task causes delay in related sequence of tasks
        - Critical path = longest sequence of floating tasks that must be completed to get the project done on time
        - Activities w/ total slack time = 0 = 0 on critical path
    - Jira is the tool usually used
4. Project closure
    - Formally closes the project and reports level of success to sponsor
    - Involves:
        - Handover of deliverables + user acceptance test sign off
        - Complete documentation
        - Post mortem
        - Release staff and equipment

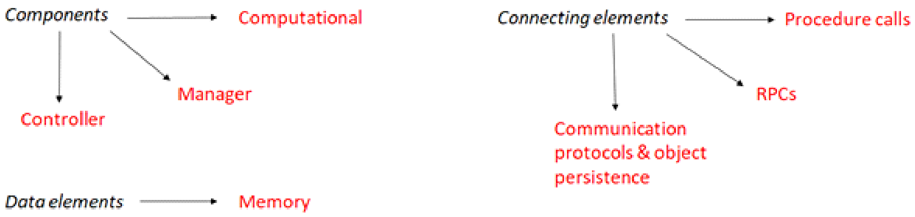
Software architecture

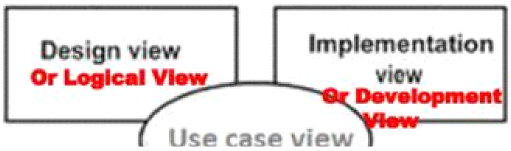
- Top level decomposition of system being developed
- Decomposition into major components + how these components interact
- Importance:
    - Architecture manifests the earliest set of design decisions
    - Supports reuse at architectural system level
    - Helps in work breakdown
    - Structures development
    - Changes to this later are super expensive
- Characteristics:
    - Addresses variety of stakeholder perspectives
    - Realizes all use cases
    - Supports separation of concerns
    - Quality driven
    - Conceptual integrity (anywhere you look, design is part of same overall design)
- Influenced by:

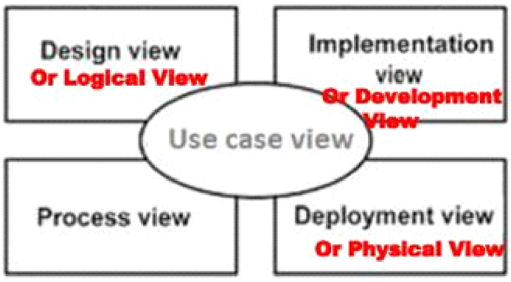
    - Functional requirements
    - Data profile
    - Audience
    - Usage characteristics
    - Business priority
    - Regulatory and legal obligations
    - Architectural standards
    - Dependencies
- Architect = high level design choices, technical standard setter
    - Distinct tole in project
    - Broad training + experience
    - Deep domain understanding
- Architectural view = ways of describing software arch = represents system as composed of some types of elements and relationships between them
    - Diff views highlight diff properties and attributes
        - Module view point
            - Code based
            - Don't represent run-time struct
            - Eg: class, package, procedure



        - Component-and-connector view point
            - System = collection of runtime entities called components
            - Component = unit with identity in executing system
            - Eg: objects, process
            - Components need to interact. They do so by connectors (pipes and sockets)

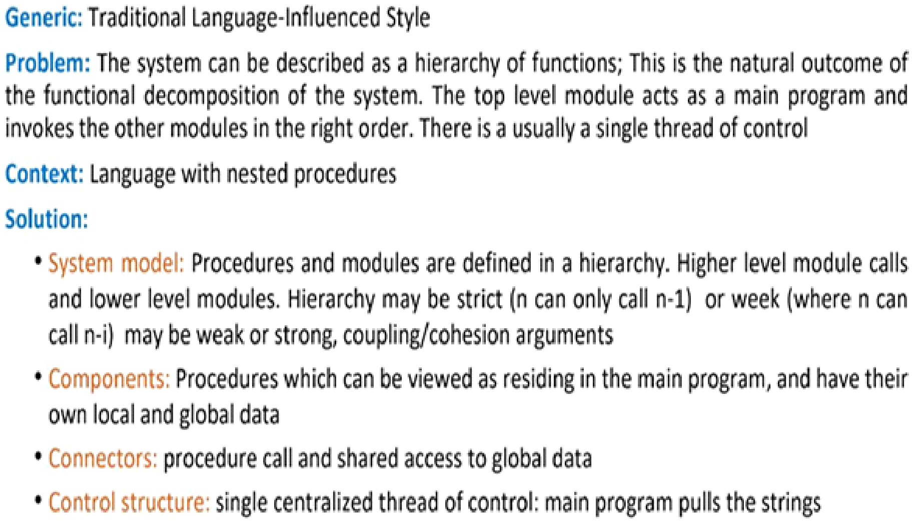


        - Allocation view point
            - How different software units are allocated to resources
            - Deployment structure: how sw is assigned to hw elements and communication paths
            - Implementation struct: how sw is mapped onto files in system
            - Work assignment: who is doing what
        - Krutchens 4 + 1 view
            - describing the architecture of software-intensive systems, based on the use of multiple, concurrent views
            - 

- 

- Use case- requirements of system
- Design view- vocab of problem and solution space
- Process view- dynamic aspects of runtime behavior
- Implementation view- realization of system
- Deployment view- system engineering issues

- Architectural style = how subsystems are organized = characterized by features that make it noticeable
    - Way of organizing modules
    - Provides:
        - Vocab- set of design elements
        - Design rules- desgin constraints
        - Semantic interpretation- well defined meaning of connected components
        - Analysis- for systems built in that style
    - Main program with subroutines:



Generic: Traditional Language-Influenced Style

Problem: The system can be described as a hierarchy of functions; This is the natural outcome of the functional decomposition of the system. The top level module acts as a main program and invokes the other modules in the right order. There is a usually a single thread of control

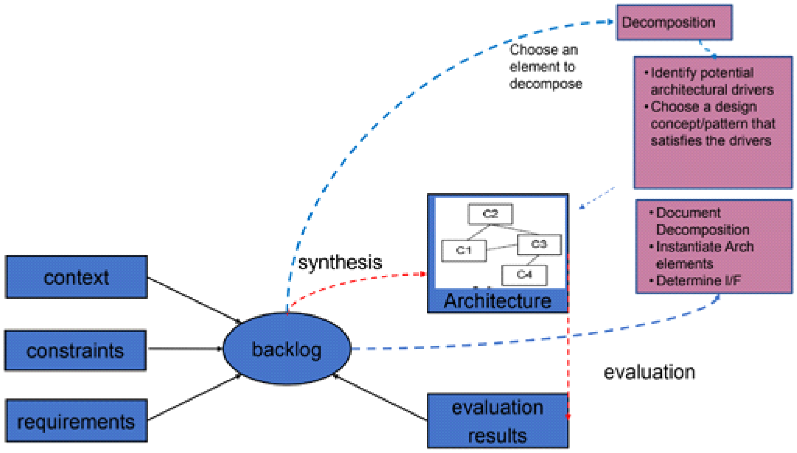
Context: Language with nested procedures

Solution:

- System model: Procedures and modules are defined in a hierarchy. Higher level module calls and lower level modules. Hierarchy may be strict (n can only call n-1) or week (where n can call n-i) may be weak or strong, coupling/cohesion arguments
- Components: Procedures which can be viewed as residing in the main program, and have their own local and global data
- Connectors: procedure call and shared access to global data
- Control structure: single centralized thread of control: main program pulls the strings

- Architectural pattern = proven solution to architectural problem structuring = features that make it notable
    - Named collection of architectural design decisions:
        - Applicable to given dev context
        - Constrain design decisions
        - Beneficial qualities in resulting system
    - The main difference is that a pattern can be seen as a solution to a problem, while a style is more general and does not require a problem to solve for its appearance
    - Popular patterns:
        - Layered
        - Client server
        - Master slave
        - Pipe filter
        - Broker
        - Peer-to-peer
        - Event-bus
        - Model-view-controller
        - Blackboard
        - Interpreter
    - Importance:
        - Reuse of design and code

        - Ease of unserstanding
        - Increased interoperability
        - Earliest design decisions - most critical to get right
        - First desgin artefact
- Types:
    - Centralized
    - Peer-to-peer
- Quality-attribute tradeoff:
    - Decision that positively affects one attr but negatively affects another
- Generalized model:


Decomposition
Choose an element to decompose
•Identify potential architectural drivers
•Choose a design concept/pattern that satisfies the drivers
C2
C1
C3
C4
Architecture
•Document Decomposition
•Instantiate Arch elements
•Determine I/F
synthesis
context
constraints
backlog
requirements
evaluation results
evaluation

    - Backlog = issues to be tackled + open problems + ideas that have to be investigated
        - Context = upfront ideas an architect may have
        - Requirements = you know
        - Constraints = given
- **Theme of architecture: decomposition**
    - Step 1 = Decompose problem into individual modules based on:
        - Layering
        - Distribution of computational resources
        - Exposure
        - Functionality
        - Generality
        - Volatility
        - Configuration
    - There are other approaches too:
        - Divide and conquer
        - Stepwise refinement
        - Top down approach
        - Bottom up
        - Info hiding

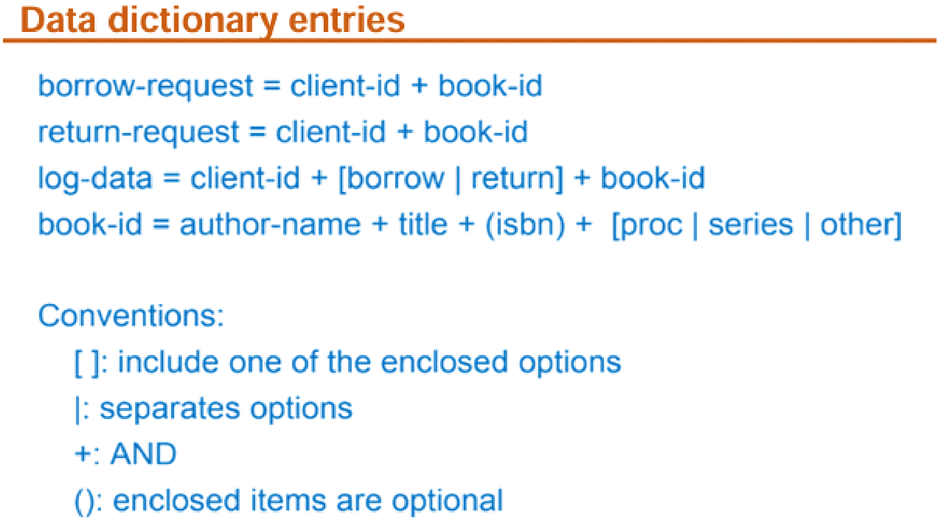
Software design

- **Principles:**
    - Further decomposition of components being developed if necessary
    - Description of sub-systems as part of architectural design
    - Description of how interfaces will be realized using DSA
    - Use of appropriate structural and behavioral design patterns
    - Description of how system will facilitate interaction with user through UI
- **Techniques:**
    - Abstraction (essential properties)
    - Modularity, cohesion, coupling
        - Modularity = extent to which large module is decomposed
        - Coupling = how strongly modules are connected to other modules
            - Content= one directly affects other
            - Common =shared data

      - External = communicate through external medium
      - Control = one module directs execution of another via necessary control info
      - Stamp = complete data structures are passed
      - Data = only data is passed
    - Cohesion = extent to which modules are related to each other
      - Coincidental= grouped into modules in haphazard way
      - Logical = elements realize tasks that are logically related
      - Temporal = independent but activated at same point of time
      - Procedural = number of elements executed in some order
      - Communicational = operate on same external data
      - Sequential = output of one is input to other
      - Functional = elements contribute to single function
  - Information hiding (series of decision, who should know and who shouldn't for each of them)
    - Encapsulation = hide data and access through specific functions
    - Separation of interface and implementation = specify public interface, separate from how component is realized
  - Limiting complexity (amount of effort to build solution)
    - Intra modular (within module)
    - Inter module
    - Higher value => Higher complexity => Higher effort required (= worse design)
  - Hierarchical struct
- **Issues:**
  - Concurrency
  - Non functional req
  - Data persistence
  - Event handling
  - Error, exception handling, fault tolerance
  - Distribution of components
  - Interaction and presentation
- **Arch vs design**

Software architecture is the structure (or structures) of the system, which comprise of software components, the externally visible properties of those components, and the relationships between them	Software design is problem-solving and planning for a software solution internal to the system
Architectural decisions are harder to change compared to design decisions which are simpler with lesser impact	Software architecture has more influence on the non-functional requirements while design has on functional requirements

- **Design methods**
  - Data flow diagrams:
    - Two steps:
      - Structured analysis -> logical design (data flow diagram)
      - Structured design -> logical design into program structure (structure chart)
    - Illustrate data flow in system
    - Four levels:
      - **Level 0**
        - Highest level
        - Major processes, data flow, data store
        - No details about internal working
      - **Level 1**
        - Break down major processes into sub processes
        - Each sub process = separate process
      - **Level 2**
        - Each sub process depicted as a different process
      - **Level 3**

            - Each process is depicted with detailed description of input, processing, output
        - Components:
            - External entity (square) (source/dest of transaction)
            - Process (circle) (transform data)
            - Data store (parallel line) (lie between processes)
            - Data flow (arrow))(data struct travels between processes)
        - Data dictionary entries:
            - Precise desc of structure of data
            - Centralized meta data repo
            - 

Data dictionary entries

borrow-request = client-id + book-id

return-request = client-id + book-id

log-data = client-id + [borrow | return] + book-id

book-id = author-name + title + (isbn) + [proc | series | other]

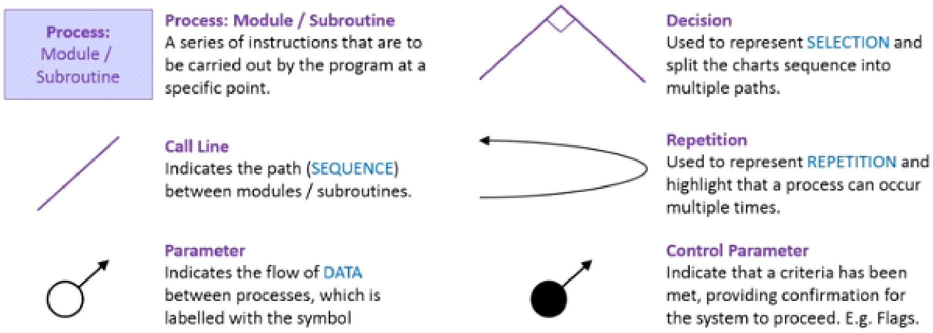
Conventions:

[ ]: include one of the enclosed options

|: separates options

+: AND

(): enclosed items are optional

        - Structure chart = chart derived from dfd (transform centered design)
            - Represents system in more detail
            - Breaks entire system down into lowest functional modules
            - 

Process: Module / Subroutine A series of instructions that are to be carried out by the program at a specific point.

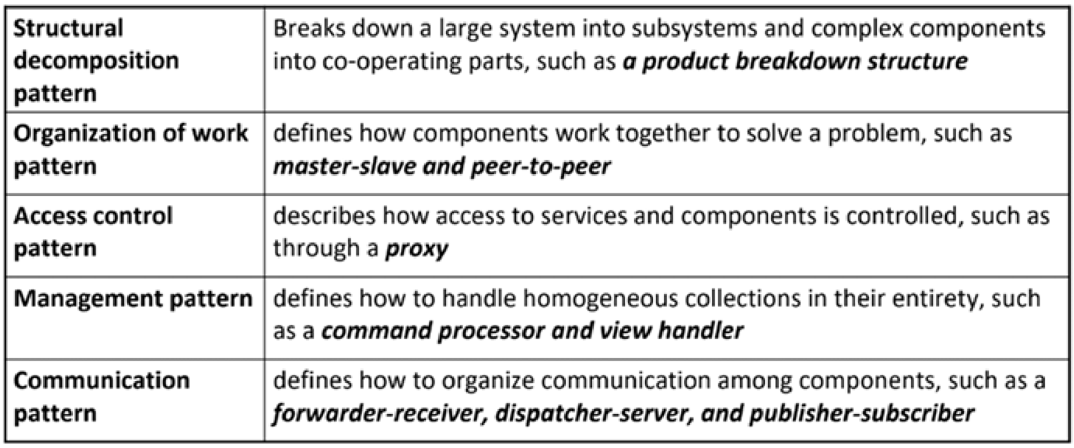
Decision Used to represent SELECTION and split the charts sequence into multiple paths.

Call Line Indicates the path (SEQUENCE) between modules / subroutines.

Repetition Used to represent REPETITION and highlight that a process can occur multiple times.

Parameter Indicates the flow of DATA between processes, which is labelled with the symbol

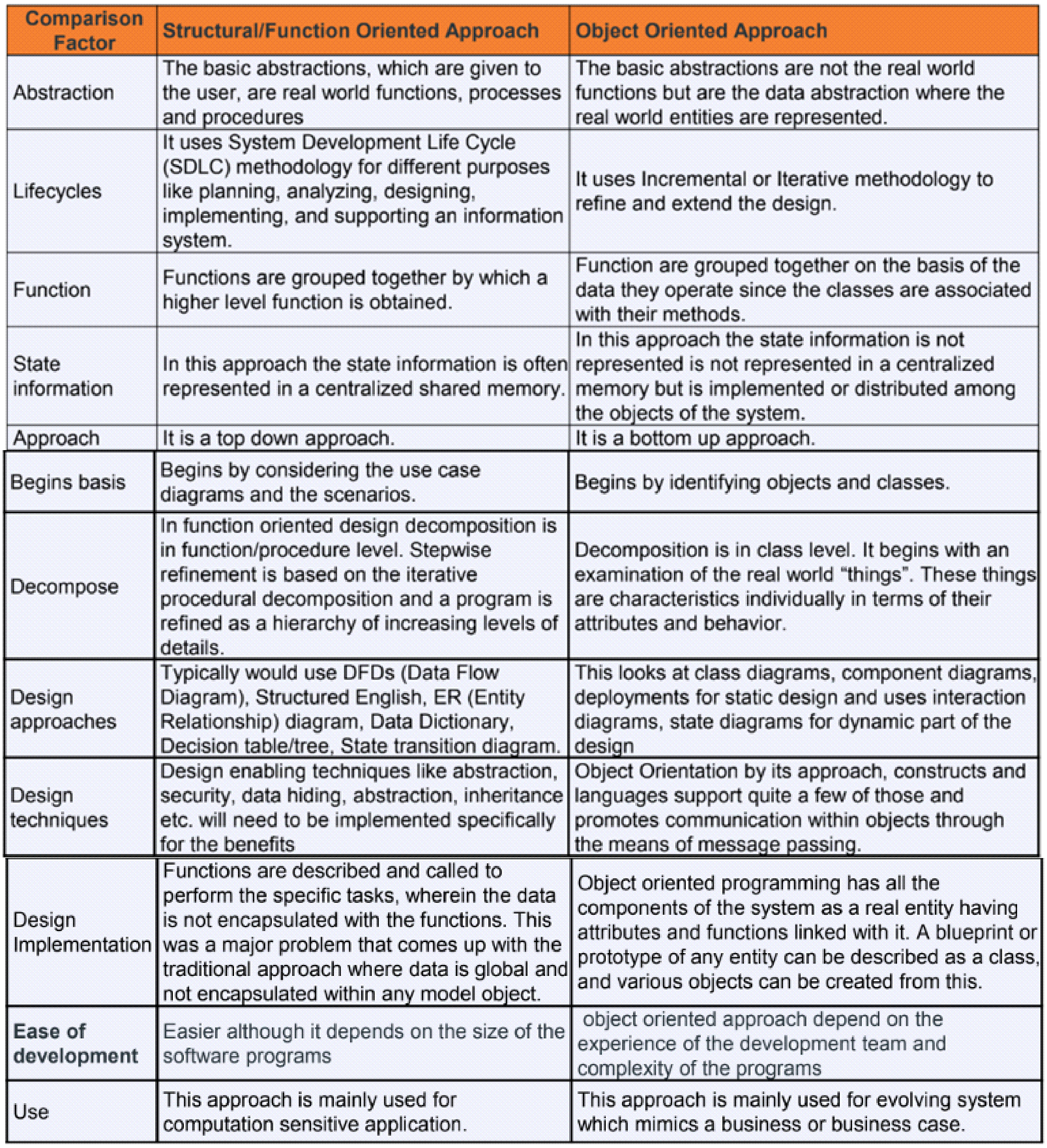
Control Parameter Indicate that a criteria has been met, providing confirmation for the system to proceed. E.g. Flags.

- **Design pattern**
    - Solution to recurring problems
    - Abstraction above level of single component
    - Common vocab for design principles
    - Means of documentation (descriptive and prescriptive)
    - Types:
        - Procedural
            - 

Structural decomposition pattern	Breaks down a large system into subsystems and complex components into co-operating parts, such as ***a product breakdown structure***
Organization of work pattern	defines how components work together to solve a problem, such as ***master-slave and peer-to-peer***
Access control pattern	describes how access to services and components is controlled, such as through a ***proxy***
Management pattern	defines how to handle homogeneous collections in their entirety, such as a ***command processor and view handler***
Communication pattern	defines how to organize communication among components, such as a ***forwarder-receiver, dispatcher-server, and publisher-subscriber***

- Object oriented

Singleton	Only one instance of class created Global point of access to object Useful when one object to coordinate actions across system
Anti pattern	Pattern of mistakes: * god object- concentration of all sorts of functions * poltergeist- pointless class used to call methods of other class



Comparison Factor	Structural/Function Oriented Approach	Object Oriented Approach
Abstraction	The basic abstractions, which are given to the user, are real world functions, processes and procedures	The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.
Lifecycles	It uses System Development Life Cycle (SDLC) methodology for different purposes like planning, analyzing, designing, implementing, and supporting an information system.	It uses Incremental or Iterative methodology to refine and extend the design.
Function	Functions are grouped together by which a higher level function is obtained.	Function are grouped together on the basis of the data they operate since the classes are associated with their methods.
State information	In this approach the state information is often represented in a centralized shared memory.	In this approach the state information is not represented is not represented in a centralized memory but is implemented or distributed among the objects of the system.
Approach	It is a top down approach.	It is a bottom up approach.
Begins basis	Begins by considering the use case diagrams and the scenarios.	Begins by identifying objects and classes.
Decompose	In function oriented design decomposition is in function/procedure level. Stepwise refinement is based on the iterative procedural decomposition and a program is refined as a hierarchy of increasing levels of details.	Decomposition is in class level. It begins with an examination of the real world “things”. These things are characteristics individually in terms of their attributes and behavior.
Design approaches	Typically would use DFDs (Data Flow Diagram), Structured English, ER (Entity Relationship) diagram, Data Dictionary, Decision table/tree, State transition diagram.	This looks at class diagrams, component diagrams, deployments for static design and uses interaction diagrams, state diagrams for dynamic part of the design
Design techniques	Design enabling techniques like abstraction, security, data hiding, abstraction, inheritance etc. will need to be implemented specifically for the benefits	Object Orientation by its approach, constructs and languages support quite a few of those and promotes communication within objects through the means of message passing.
Design Implementation	Functions are described and called to perform the specific tasks, wherein the data is not encapsulated with the functions. This was a major problem that comes up with the traditional approach where data is global and not encapsulated within any model object.	Object oriented programming has all the components of the system as a real entity having attributes and functions linked with it. A blueprint or prototype of any entity can be described as a class, and various objects can be created from this.
Ease of development	Easier although it depends on the size of the software programs	object oriented approach depend on the experience of the development team and complexity of the programs
Use	This approach is mainly used for computation sensitive application.	This approach is mainly used for evolving system which mimics a business or business case.

Service oriented architecture

- Approach for building software that incorporates complete enterprise development
- Reuse components from pre existing applications across enterprise
- Communication between various platforms and languages
- Each service is a complete business function by itself
- Services are published to make it easier for developers to assemble their apps
- <u>Characteristics:</u>
  - Interoperability between services
  - Methods for service encapsulation, discovery, composition, integration
  - Facilitates quality of services through service contract based on service level agreement

    - Loosely coupled services
    - Location transparency and better scalability and availability
    - Ease of maintenance with reduced cost of application development and deployment
- Advantages:
    - Service reusability
    - Easy maintenance
    - Platform independent
    - Availability
    - Reliability
    - Scalability
- Disadvantages:
    - High overhead
    - High investment
    - Complex service management

Services

- Service = logical representation of repeatable business activity with specified outcome
- Can be written in any language
- Implemented as callable entities
- Characteristics:
    - Adhere to service contract- service desc doc
    - Loosely coupled- self contained components
    - Autonomous- services have control over logic
    - Abstraction- logic is hidden
    - Reusable - designed as components
    - Can be discovered- can be discovered through meta data
    - Composed from larger services
    - Service orchestration- aggregate info from one service
    - Service choreography- coordinated interaction of services without single point of control
- Service oriented arch roles:
    - Service provider
    - Service broker
    - Service requester
- Two layers of services communicate through service bus
- Microservices:
    - set of services that act together to make a whole application operate
    - This architecture utilizes APIs to pass information, such as user queries or a data stream service to another
- **SOA vs micoservice arch**

	Microservice	SOA
Granularity	Fine-grained, small services	Coarse-grained, larger services
Independence	Highly independent, can be deployed separately	Less independent, often rely on an Enterprise Service Bus (ESB)
Communication	Lightweight protocols	Heavier protocols
Technology Stack	Diverse, allows different technologies for different services	More uniform technology stack
Deployment	Containerized, highly scalable and resilient	Traditional servers or virtual machines
Flexibility	High, easy to modify or replace services	Moderate, more tightly coupled services
Scalability	High, services can be scaled independently	Moderate, scaling often involves the entire service