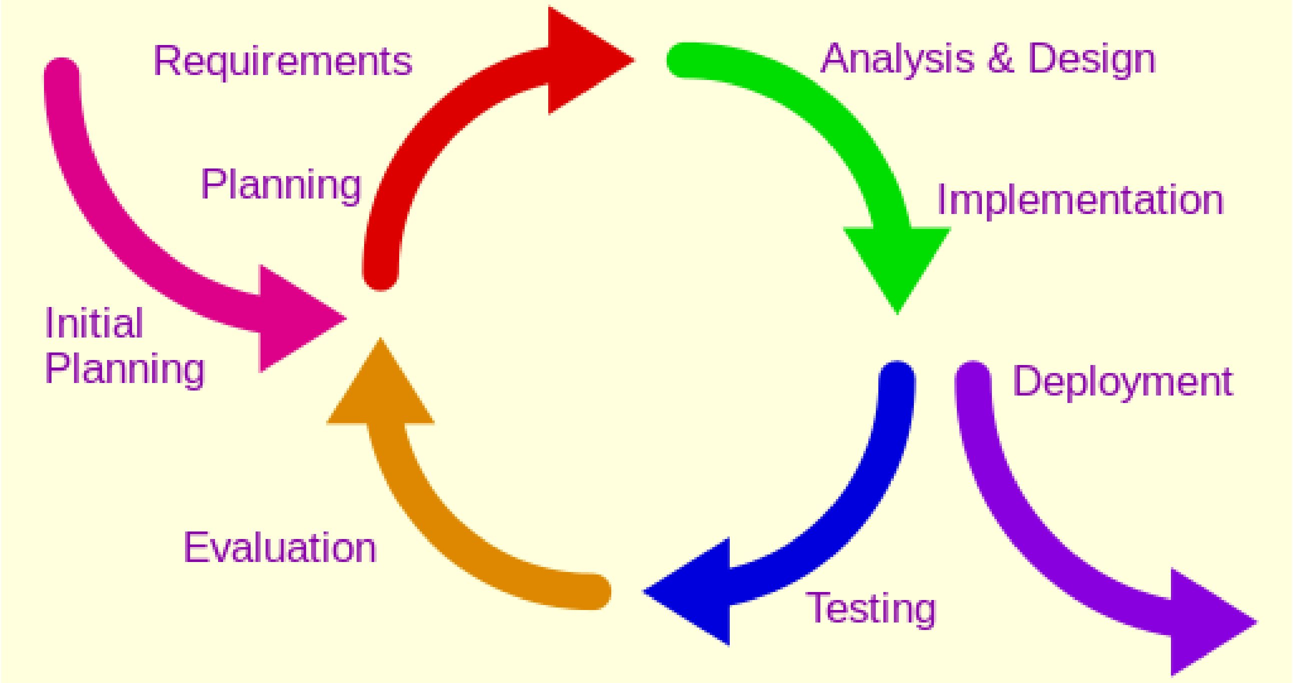
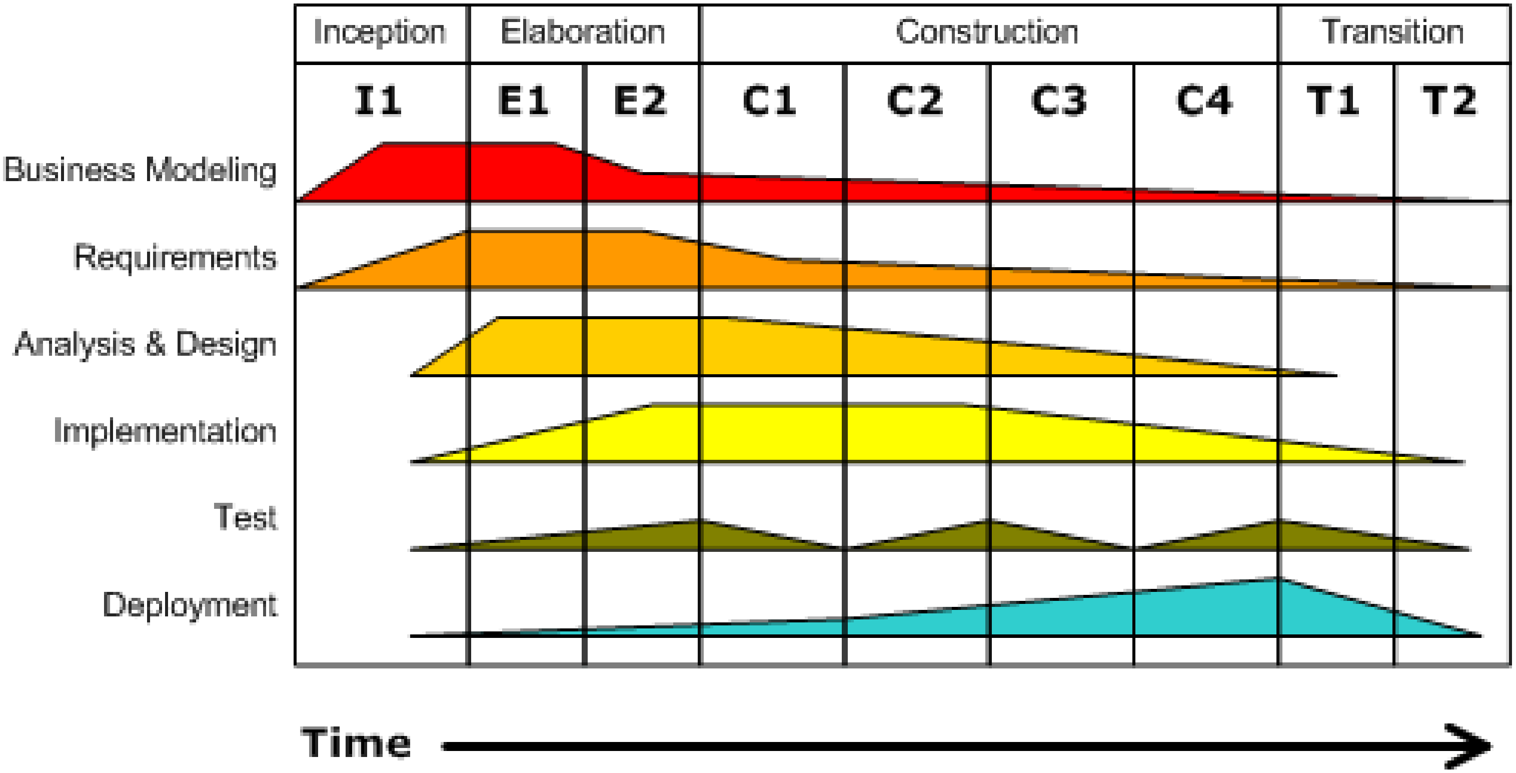
Software Design

SD

Iterative development model

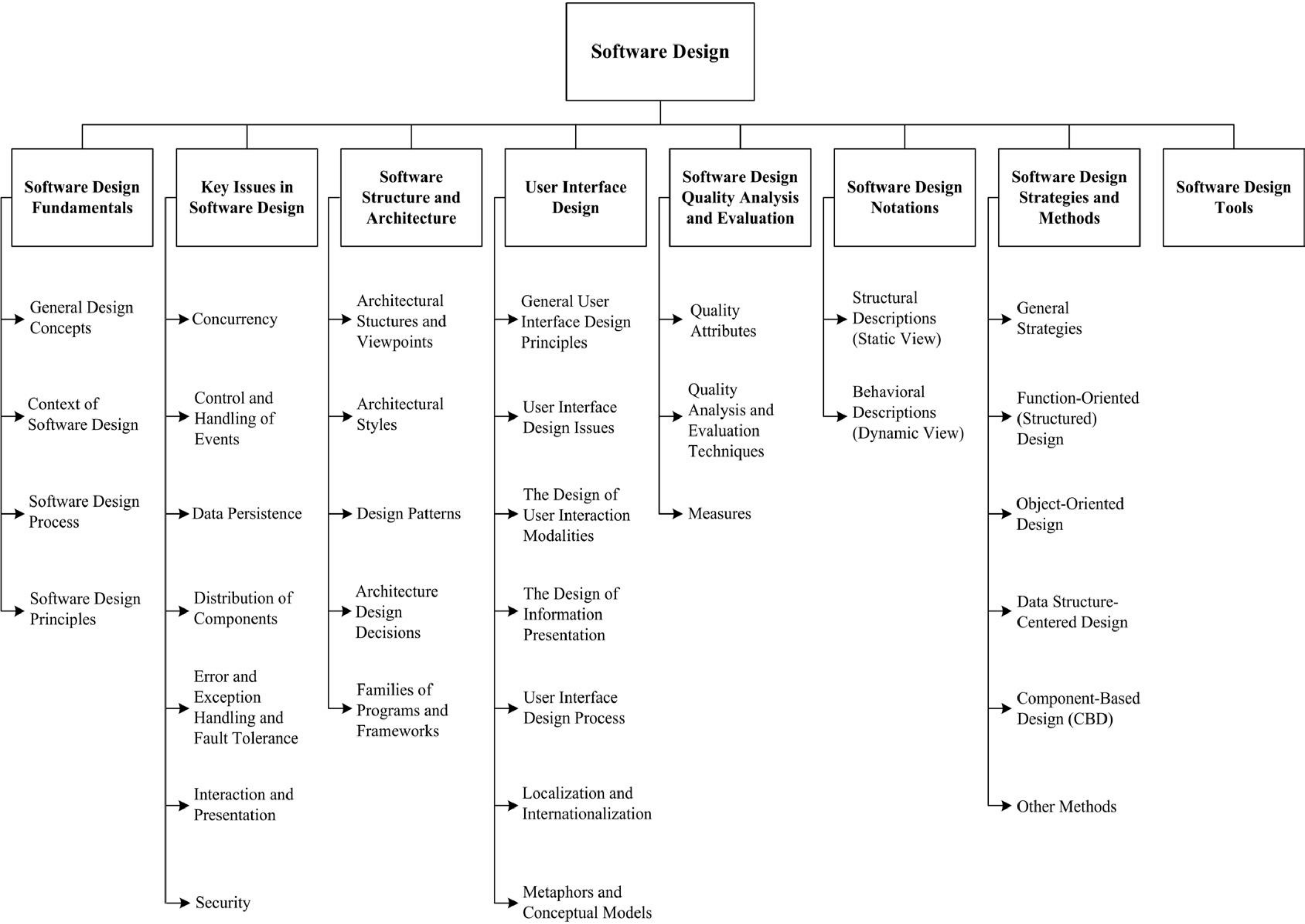


RUP model



Iterative Development

Business value is delivered incrementally in
time-boxed cross-discipline iterations.


Software Design
Software Design Fundamentals
General Design Concepts
Context of Software Design
Software Design Process
Software Design Principles
Key Issues in Software Design
Concurrency
Control and Handling of Events
Data Persistence
Distribution of Components
Error and Exception Handling and Fault Tolerance
Interaction and Presentation
Security
Software Structure and Architecture
Architectural Stuctures and Viewpoints
Architectural Styles
Design Patterns
Architecture Design Decisions
Families of Programs and Frameworks
User Interface Design
General User Interface Design Principles
User Interface Design Issues
The Design of User Interaction Modalities
The Design of Information Presentation
User Interface Design Process
Localization and Internationalization
Metaphors and Conceptual Models
Software Design Quality Analysis and Evaluation
Quality Attributes
Quality Analysis and Evaluation Techniques
Measures
Software Design Notations
Structural Descriptions (Static View)
Behavioral Descriptions (Dynamic View)
Software Design Strategies and Methods
General Strategies
Function-Oriented (Structured) Design
Object-Oriented Design
Data Structure-Centered Design
Component-Based Design (CBD)
Other Methods
Software Design Tools

SD Process

Software design consists of two **activities** that fit between software **requirements analysis** and software **construction**:

- ❑ Software **architectural** design – **high-level** design:
  - developing top-level structure and organization of software
  - identifying various components
- ❑ Software **detailed** design: specifies each component in **sufficient** detail to facilitate its construction

Fundamental Principles of SD - I

- **Abstraction** – view of an object that focuses on the information relevant to a **particular purpose** and ignores the remainder of the information:
  - Abstraction by specification:
    - procedural abstraction
    - data abstraction
    - control (iteration) abstraction
  - Abstraction by parameterization – representing the data as named parameters
- **Sufficiency and completeness** – a software component captures all the **important characteristics of an abstraction** and nothing more
- **Encapsulation and information hiding** – grouping and **packaging** the internal details of an abstraction and making those details inaccessible to external entities

Fundamental Principles of SD - II

- **Decomposition and modularization** – large software is divided into a number of smaller named components having well-defined interfaces that describe component interactions

- **Separation of interface and implementation** – defining a component by specifying a public interface (known to the clients) that is separate from the details of how the component is realized

- **Coupling and Cohesion:**
  - Coupling – a measure of the interdependence *among modules* in a computer program
  - Cohesion – a measure of the strength of association of the elements *within a module*

- **Primitiveness** – the design should be based on patterns that are easy to implement

Key Issues in SD

- **Concurrency** – decomposing software into **processes**, **tasks** and **threads** and dealing with related issues of efficiency, atomicity, synchronization and scheduling

- **Control and handling of events** – organizing data and control **flow** as well as handling reactive and temporal events through various mechanisms such as implicit invocation and call-backs

- **Data persistence** – handling long-lived data

- **Distribution of components** – distributing the software across the hardware, organizing communication of components, using middleware to deal with heterogeneous software

- **Interaction and presentation** – structuring and organizing interactions with users as well as the presentation of information

- **Error and exception handling and fault tolerance**

- **Security**
  - preventing unauthorized disclosure, creation, change, deletion or denial of access to information and other resources
  - tolerating security-related attacks or violations by limiting damage, continuing service, speeding repair and recovery, and failing and recovering securely
  - using of cryptology

Software Architecture

- **Architecture**:
  - strict sense -- **set of structures** needed to reason about system, which comprise software elements, **relations among them**, and properties of both
  - general sense -- set of views -- different high-level facets -- about software design at different levels of abstraction
- **Representation of a partial aspect** of a software architecture, that shows specific properties of a software system, by the views:
  - **logical view** -- satisfying functional requirements
  - **process view** -- concurrency issues
  - **physical view** -- distribution issues
  - **development view** -- how the design is broken down into **implementation units** with explicit representation of the dependencies among the units

Architectural Styles

- **Architectural styles** can be viewed as patterns describing the **high-level** organization of software
- Architectural style
  - specialization of element and relation types, together with a set of constraints on how they can be used
  - providing the software’s high-level organization
- Major architectural styles
  - General structures (layers, pipes and filters, blackboard)
  - Distributed systems (client-server, three-tiers, broker)
  - Interactive systems (**Model-View-Controller**, Presentation-Abstraction-Control)
  - Adaptable systems (microkernel, reflection)
  - Others (batch, interpreters, process control, rule-based).

Design Patterns

- **Design patterns** (GoF) can be used to describe details at a **lower level**:
  - Creational patterns
  - Structural patterns
  - Behavioral patterns

SD Notations

- **Structural Descriptions (Static View)**
  - **Class and object diagrams**
  - **Component diagrams**
  - Class responsibility collaborator cards
  - **Deployment diagrams**
  - Entity-relationship diagrams
  - Architecture description languages
  - Interface description languages
  - Structure charts
- **Behavioral Descriptions (Dynamic View)**
  - **Sequence diagrams**
  - **Activity diagrams**
  - Communication diagrams
  - Data flow diagrams
  - Decision tables and diagrams
  - Flowcharts
  - **State transition and state chart diagrams**
  - Formal specification languages
  - Pseudo code and program design languages

SD Quality Attributes

- Quality Attributes contribute to the quality of a software design:

  “-ilities” : maintainability, portability, testability, usability...

  “-nesses”: correctness, robustness ...

- Attributes **discernible** at runtime:

  performance, security, availability, functionality, usability

- Attributes **not discernible** at runtime:

  modifiability, portability, reusability, testability

- Attributes related to the architecture’s intrinsic qualities:

  conceptual integrity, correctness, completeness

Quality analysis and evaluation techniques

- **Software design reviews** – informal and formalized techniques to determine the **quality of design artifacts**:
  - architecture reviews
  - design reviews and inspections
  - scenario-based techniques
  - **requirements tracing**
- Security evaluation by design reviews
- Review of aids for installation, operation, and usage
- **Static analysis:** formal or semiformal static (**nonexecutable**) analysis that can be used to evaluate a design (fault tree analysis or automated cross-checking)
  - Design vulnerability analysis – static analysis for security weaknesses
  - Formal design analysis – using mathematical models that allow to predicate the behavior and validate the performance of the software instead of having to rely entirely on testing; can be used to detect residual specification and design errors
- **Simulation and prototyping:** dynamic techniques to evaluate a design for performance simulation or feasibility prototypes

Measures

- Measures are classified in two broad categories:
  - **function-based** (structured) design measures:
    - obtained by analysis of functional decomposition
    - generally represented using a structure chart (a hierarchical diagram), on which various measures can be computed
  - **object-oriented design measures**:
    - design structure is typically represented as a class diagram, on which various measures can be computed
    - measures on the properties of the internal content of each class
- Most measures depend on the approach used for producing design