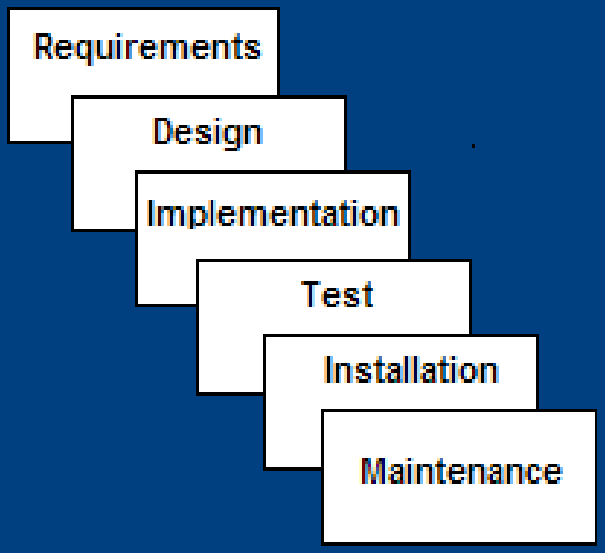
Software Development Life Cycle (SDLC)

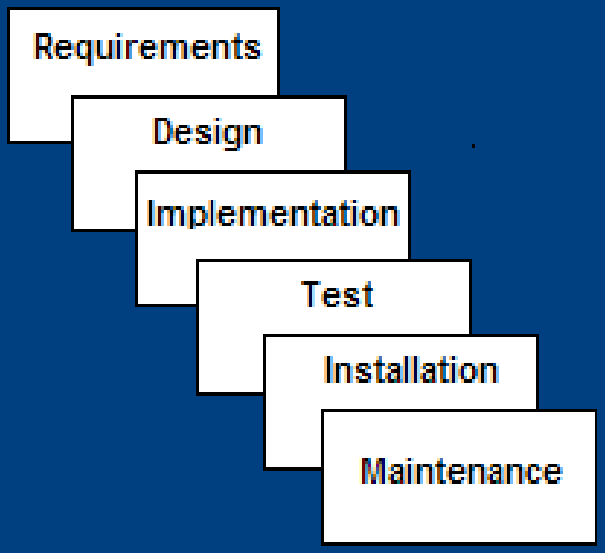
- Waterfall Model
- V-Shaped Model
- Prototyping
- Incremental Model
- Spiral Model

Waterfall Model



- **Requirements** – defines needed information, function, behavior, performance and interfaces.
- **Design** – data structures, software architecture, interface representations, algorithmic details.
- **Implementation** – source code, database, user documentation, testing.

Waterfall Model



- **Test** – check if all code modules work together and if the system as a whole behaves as per the specifications.
- **Installation** – deployment of system, user-training.
- **Maintenance** – bug fixes, added functionality (an on-going process).

Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)

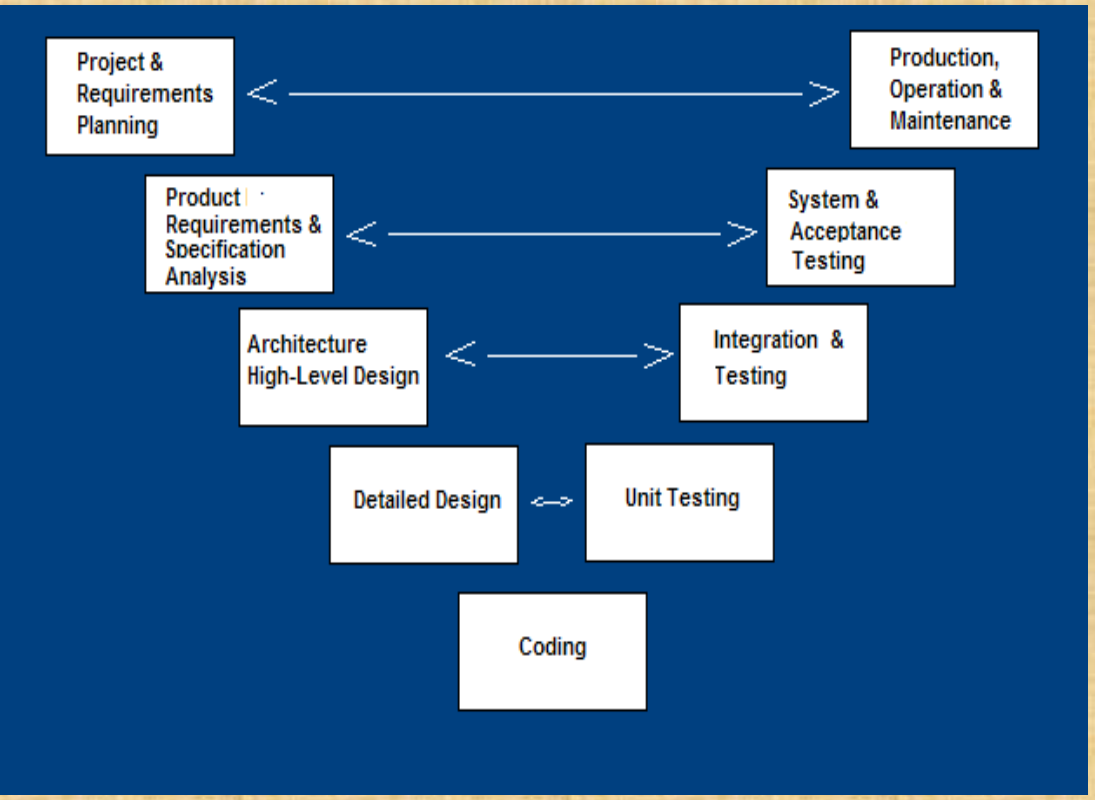
Waterfall Deficiencies

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

When to use the Waterfall Model

- Requirements are very well known
- When it is possible to produce a stable design
- E.g. a new version of an existing product
- E.g. porting an existing product to a new platform.

V-Shaped SDLC Model



- A variant of the Waterfall that emphasizes the verification and validation of the product.
- Testing of the product is planned in parallel with a corresponding phase of development

V-Shaped Steps

- Project and Requirements Planning – allocate resources
- Product Requirements and Specification Analysis – complete specification of the software system
- Architecture or High-Level Design – defines how software functions fulfill the design
- Detailed Design – develop algorithms for each architectural component
- Coding – transform algorithms into software
- Unit testing – check that each module acts as expected
- Integration and Testing – check that modules interconnect correctly
- System and acceptance testing – check the entire software system in its environment
- Production, operation and maintenance – provide for enhancement and corrections

V-Shaped Strengths

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

V-Shaped Weaknesses

- Does not easily handle concurrent events
- Does not handle iterations or phases
- Does not easily handle dynamic changes in requirements

When to use the V-Shaped Model

- Excellent choice for systems requiring high reliability – hospital patient control applications
- All requirements are known up-front
- When design is stable
- Solution and technology are known

Prototyping Model

- Developers build a prototype during the requirements phase
- Prototype is evaluated by end users
- Users give corrective feedback
- Developers further refine the prototype
- When the user is satisfied, the prototype code is brought up to the standards needed for a final product.

Prototyping Steps

- A preliminary project plan is developed
- An partial high-level paper model is created
- The model is source for a partial requirements specification
- A prototype is built with basic and critical functions
- The designer builds
  - the database
  - user interface
  - algorithmic functions
- The designer demonstrates the prototype, the user evaluates for problems and suggests improvements.
- This loop continues until the user is satisfied

Prototyping Strengths

- Customers can "see" the system requirements as they are being gathered
- Developers learn from customers
- A more accurate end product
- Unexpected requirements accommodated
- Allows for flexible design and development
- Steady, visible signs of progress produced
- Interaction with the prototype stimulates awareness of additional needed functionality

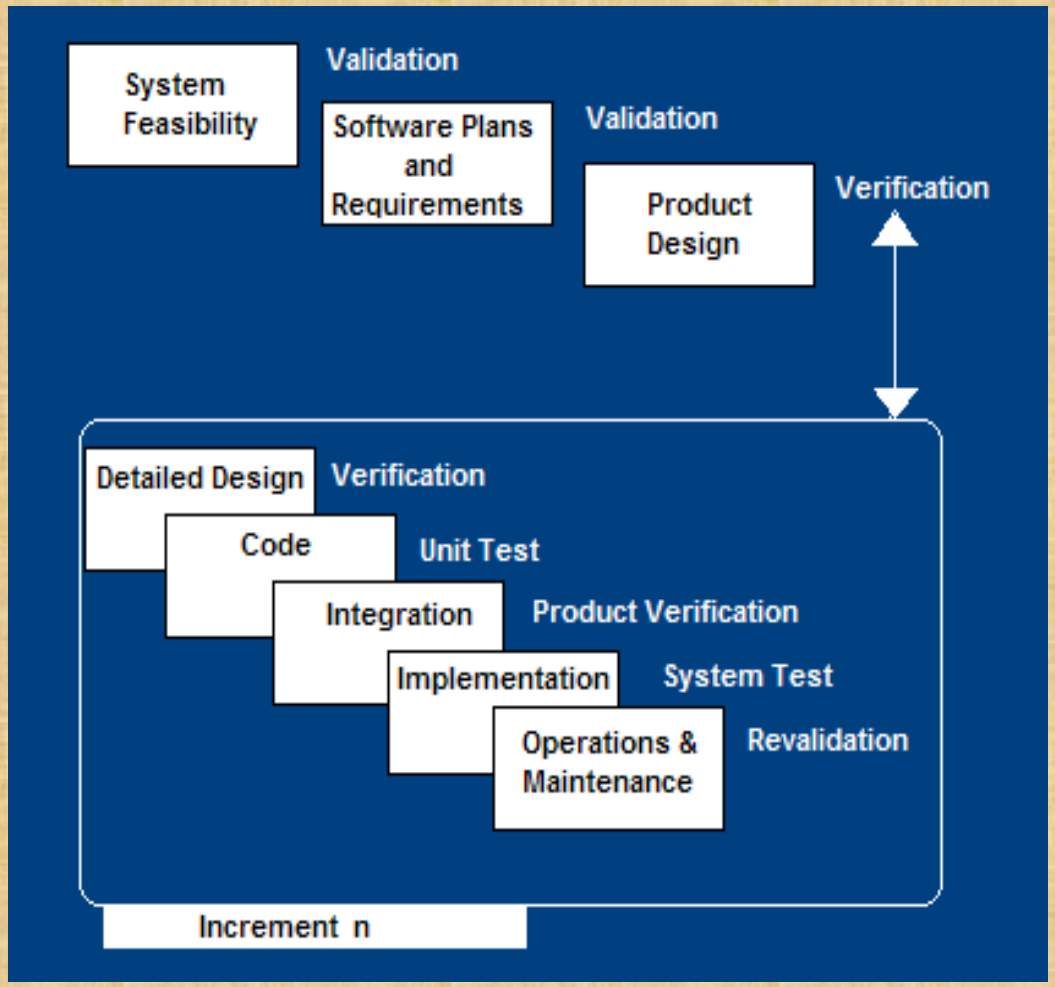
Prototyping Weaknesses

- Tendency to abandon structured program development for “code-and-fix” development
- Bad reputation for “quick-and-dirty” methods
- Overall maintainability may be overlooked
- Process may continue forever (scope creep)

When to use Prototyping

- Requirements are unstable or have to be clarified
- As the requirements clarification stage of a waterfall model
- Develop user interfaces
- New, original development

Incremental SDLC Model



- Construct a partial implementation of a total system
- Then slowly add increased functionality
- The incremental model prioritizes requirements of the system and then implements them in groups.
- Each subsequent release of the system adds functions to the previous release, until all designed functionality has been implemented.

Incremental Model Strengths

- Develop high-risk or major functions first
- Each release delivers an operational product
- Customer can respond to each build
- Uses "divide and conquer" breakdown of tasks
- Lowers initial delivery cost
- Initial product delivery is faster
- Customers get important functionality early

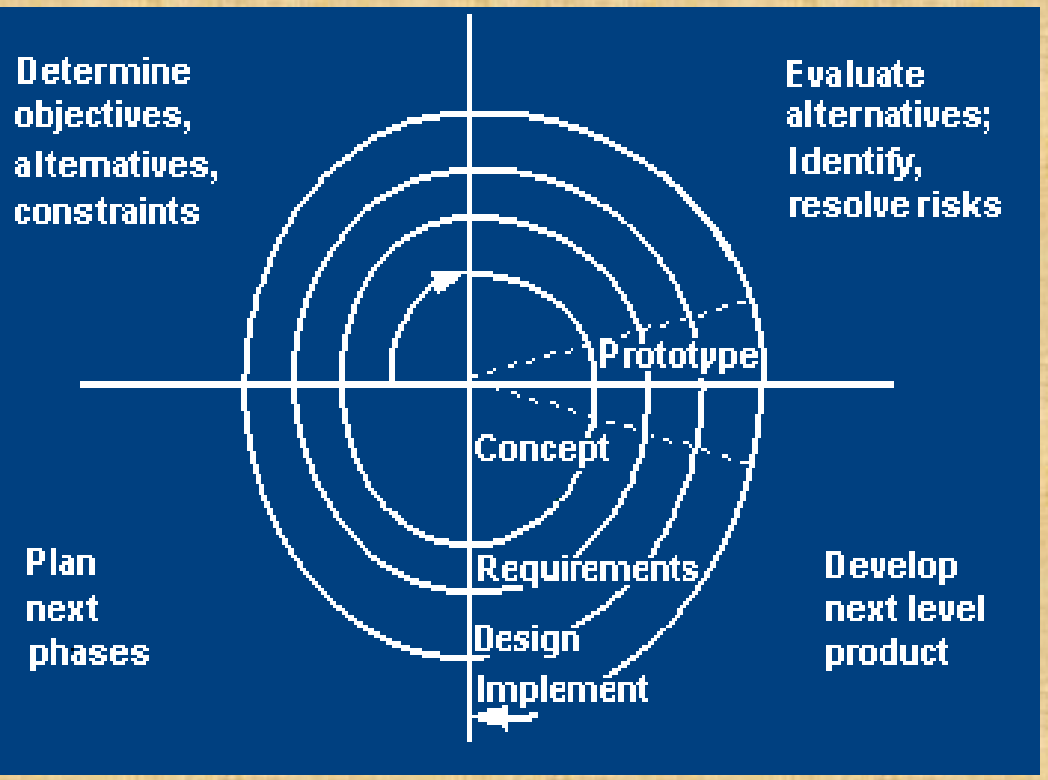
Incremental Model Weaknesses

- Requires good planning and design
- Requires early definition of a complete and fully functional system to allow for the definition of increments
- Well-defined module interfaces are required (some will be developed long before others)
- Total cost of the complete system is not lower

When to use the Incremental Model

- Most of the requirements are known up-front but are expected to evolve over time
- A need to get basic functionality to the market early
- On projects which have lengthy development schedules

Spiral SDLC Model



- Adds risk analysis, and 4gl RAD prototyping to the waterfall model
- Each cycle involves the same sequence of steps as the waterfall process model

- Objectives: functionality, performance, hardware/software interface, critical success factors, etc.
- Alternatives: build, reuse, buy, sub-contract, etc.
- Constraints: cost, schedule, man-power, experience etc.

Spiral Model Strengths

- Provides early indication of insurmountable risks, without much cost
- Users see the system early because of rapid prototyping tools
- Critical high-risk functions are developed first
- Users can be closely tied to all lifecycle steps
- Early and frequent feedback from users

Spiral Model Weaknesses

- Time spent for evaluating risks too large for small or low-risk projects
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- The model is complex
- Risk assessment expertise is required
- Spiral may continue indefinitely
- Developers must be reassigned during non-development phase activities

When to use Spiral Model

- When creation of a prototype is appropriate
- When costs and risk evaluation is important
- For medium to high-risk projects
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected

Quality Assurance Plan

- Defect tracing – keeps track of each defect found, its source, when it was detected, when it was resolved, how it was resolved, etc
- Unit testing – each individual module is tested
- Source code tracing – step through source code line by line
- Integration testing - test new code in combination with code that already has been integrated
- System testing – execution of the software for the purpose of finding defects.