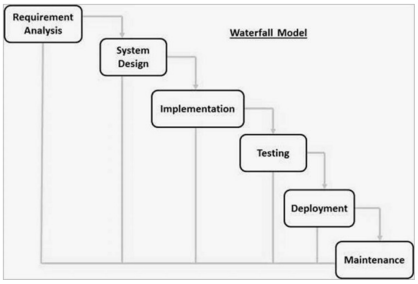
Waterfall Model - Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.



Requirement Gathering and analysis – All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

System Design – The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

Implementation – With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

Integration and Testing – All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Deployment of system – Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

Maintenance – There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

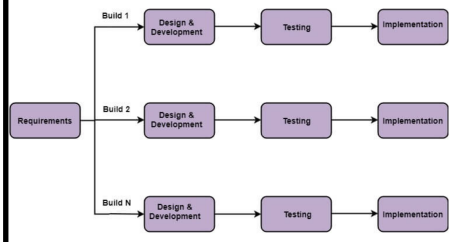
Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows –

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Incremental Model

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.



The various phases of incremental model are as follows:

1. **Requirement analysis:** In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

2. **Design & Development:** In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

3. **Testing:** In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

4. **Implementation:** Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantage of Incremental Model

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

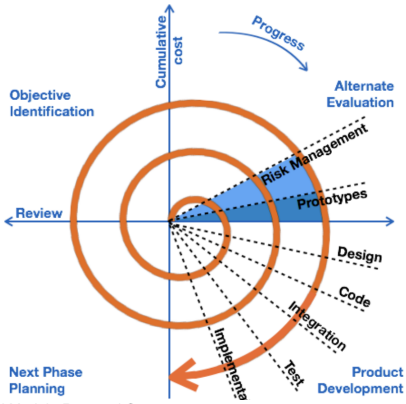
Spiral Model - Design

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification

This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.



Spiral Model - Pros and Cons

The advantage of spiral lifecycle model is that it allows elements of the product to be added in, when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple software builds and releases which allows making an orderly transition to a maintenance activity. Another positive aspect of this method is that the spiral model forces an early user involvement in the system development effort.

On the other side, it takes a very strict management to complete such products and there is a risk of running the spiral in an indefinite loop. So, the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

The advantages of the Spiral SDLC Model are as follows –

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.
- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.

Allocating Requirements

Hardware: User hardware (What user needs to access product) & Operational Hardware (Stuff like servers and databases)
System Software (Operating system)
Application Software (Like what web browsers can run it)
User Process (User interface, flow, organization)
Development Process (Which lifecycle model to use, and what methodology [agile or traditional])

Testable Requirements

Testable requirements: (1) Application portability (2) Application performance (3) Input Validation
(4) Application scalability
Non-testable requirements:
(1) Server reliability since the creation and maintenance of physical servers is not typically handled by a software development team. As such, the most we could do is try to find any public logs of the provider's history of reliability and trust that.
(2) User hardware requirements given that even if the application is designed to be highly portable and, as such, capable of running across various web browsers, we cannot control whether a user's operating system or machine is capable of running a web browser.

Case ID	Name	Description
1	Software-intensive application or system	Software application/system that executes on one or more commercial hardware platforms. It can be a stand-alone software system or a constituent within one or more systems of systems.
2	Software-intensive device	An object, machine, or piece of equipment that is developed for a special purpose and has significant features provided by software.
3	Hardware platform	Vehicle (land, sea, air, or space).
4	Family of systems/product lines	Set of systems that can interoperate with each other or are related to each other (e.g., have common components) as part of a product line.
5	System of systems/ enterprise-wide systems	Set of independent (constituent) systems that can be integrated together in a manner that allows them to interoperate and perform cross-cutting mission-specific capabilities.
6	Brownfield modernization	Refactoring or re-implementation of an older legacy system or set of systems.

Table 13.3 Function Point Weights

Software Components	Simple	Average	Complex
External inputs	3	4	6
External outputs	4	5	7
External inquiries	3	4	6
Internal logical files	7	10	15
External interface files	5	7	10

SUPPORTING SOFTWARE DEVELOPMENT METHODOLOGIES: SELECTING THE RIGHT LEVEL OF RIGOR

Methodology	Description
Architected agile	Initial iteration focuses on foundations/architecture issues, then transitions to pure agile process for development of software capabilities
Agile	Software developed using pure agile methods with short-duration iterations and with little formal documentation
Plan-Driven	Moderate level of rigor that employs up-front planning and moderate level of formal documentation and tracking throughout the development of software increments
Formal methods	Critical software system or subsystem, often containing security- or safety-relevant software or critical/high-precision algorithms that must be rigorously developed, tested, and often certified
COTS/Services	Software functionality provided through the integration of one or more commercial off-the-shelf software components or software services

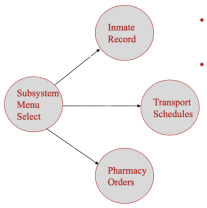
EQUIVALENCE PARTITIONING

- Input domain divided into classes of data from which test cases can be derived
  - Ranges of values
  - Sets of values
  - Boolean values
- Looking for *classes* of errors, thereby reducing the total number of test cases that must be developed
- An equivalence class represents a set of valid or invalid states for input conditions

BOUNDARY VALUE ANALYSIS (BVA)

- Similar to boundary condition testing at the unit level
- BVA defines test cases that exercise bounding values as identified by requirements
- Complements Equivalence Partitioning by selecting values at the "edge" of an equivalence class
- Should be applied to both inputs and outputs
- How to select BVA test data values:
  - If an input or output condition specifies a range of bounded values, for example, 0-2400, test cases should be designed with values 0 and 2400 and just above/below 0 and 2400
  - If an input or output condition specifies a number of values, for example (8, 12, 24, 48, and 72), focus on min and max values and values just above/below min and max
  - If there are any known internal limits (e.g., table has a defined limit of 1000 records), exercise data structure at its boundary and just beyond

GRAPH-BASED TESTING



- Based on objects that are modeled in the software and the relationships between these objects
- Approach:
  - Create a graph of key objects (e.g., subsystems, states, or events) and their relationships
  - Create a set of tests that will evaluate all objects and the relationships between them

SECURITY-RELEVANT TESTING

- Defined testing that focuses on security-relevant aspects of the system
  - User privileges
  - Data security and accuracy
  - Required account/data protections
- Ad hoc testing that tries to "break-in" to software system
  - Subvert protections built into the software
  - Find "trojan horses" that allow "unauthorized access"

COMPARISON TESTING

- Used in situations where system reliability and availability are absolutely critical
- Software test teams use the redundant software in testing to compare results when using identical inputs
- Assumption: If results are consistent, then the processing (developed by independent teams) must be correct… if results ar inconsistent, further analysis required to determine if one or both systems are incorrect
- Test team may decide to develop redundant version of software t support testing activities
- Beware – both could be wrong!

DATABASE TESTING

- Purpose
  - Uncovers problems with
    - » COTS interfaces
    - » Data corruption
    - » Unauthorized access
- Characteristics
  - Known starting point and expected end points are mandatory
  - Tests should be automated to facilitate retest

BLACK BOX TESTING: TYPES AND TECHNIQUES

- Graph-based testing
- Equivalence partitioning
- Boundary value analysis
- Comparison testing
- Database testing
- Usage-based testing
- Performance testing
- Security relevant testing

PERFORMANCE TESTING

- Load testing
- Stress testing
- User response time
- Transaction throughput
- Network utilization
- CPU utilization
- Memory utilization

Generally requires commercial (vendor) tools to conduct these tests… Most tools are complex and specific to hardware platforms and operating systems…

USAGE-BASED TESTING

- Focuses on the expected user interactions with the system
  - High-usage and/or critical functions/features are tested very rigorously
  - Lower-usage/non-critical functions/features tend to be minimally tested
- Used to prioritize testing areas when test time is limited

DATABASE TESTING (CONTINUED)

- Database features to test:
  - Triggers
  - Stored procedures
  - Data integrity checks
  - Data access constraints
- Initialize database tables prior to testing
- Use DBMS query, display, and report capabilities to support testing

example from our project

c. ICSM Common Case to be used for development and rationale
   i. The ICSM common case to be used for development is the family of systems. This is because the purpose of the application is to tie together several differe systems due to the large amount of common data and common components th they share.

d. Methodologies/techniques to be used in development and rationale
   i. Due to the time constraints caused by the small amount of work hours availab for each individual team member and the short timeframe to accomplish the t we plan on implementing the Architected Agile methodology. We have 3 mai reasons for choosing this methodology. Firstly, since most of the necessary documentation has already been written as part of the project description, the normal lack of documentation and ideation that is usually part of an Agile methodology would not have much of an impact on development. Secondly, since we only have 3 months to develop the application while also taking other high level courses and, for some members of the team, working full time jobs, we felt that short sprints with smaller tasks would help ease development. Lastly, the architected agile methodology works perfectly in tandem with the Common Case being used for development. In other words, since we are approaching our application as a family of systems, the first step we took before starting development was identifying all the common features between the various submodules of the application. These identified common features are what we converted into the first set of tasks that we will be working on during our first sprint.

What requirements qualify as functional requirements? Specify in what manner they need to be achieved.

- Input formats (What is the format for the input data? How should it be stored? What will serve as line separators on the files? What is the input character set?)
- Sorting (Specifies how to sort a file by characters, numbers, etc. We define sorting among characters in numerical order, and the sorting of the file to be in ascending order)
- Special cases, boundaries, and error conditions (Specifies how to handle special cases, boundaries and error conditions)

Which decisions are those taken by the software engineer about the best ways (processes, techniques, and technologies) to achieve the requirements

1) What will the user interface look like?
2) What will typical and maximum input sizes look like? The size of the inputs will determine what kind of algorithms should be used and how much time should be spent on performance optimizations.
3) Which platforms should be chosen to run a program
4) Which programming language to use that is most suitable according to design constraints

Describe a way to simplify a complex problem

Lessen the relationships, number of functionalities, amount of interactions, etc. Techniques:
Decomposition *(Breaking down a large, complex system into smaller parts that are easier to understand and, by extension, develop.)*
Modularization *(Separate all independent functions of a system into a set of modules that are interchangeable and can be developed in isolation of one another.)*
Separation (*Form of abstraction that functions by first breaking the system down into sets of information that will affect program execution and then creating well-defined interfaces around each such piece of information. (Kind of a combination of decomposition and modularization)*
Incremental iterations *(Only introduce upgrades to the system in small increments divided into specific periods of time which start and end with a requirements analysis to determine which aspect of the system will be focused on next.)*

List two technical concerns in developing large systems.

(1) Architecture and design decomposition (2) Choices of language, database, network, middleware (3)Processes and methodologies to be used for development

From the definition of software engineering, list three areas that software engineering must touch upon

(1) Technical and business processes (2) Specific methodologies and techniques (3) Product characteristics and metrics (4) People, skills, teamwork (5) Tools and training (6)Project Coordination and management

List two of the three strategies cited by the 2004 U.S Gen. Acc. Office report as key to ensuring delivery of successful software

(1) Attention must be given to software development environment (2) The development process should be disciplined (3) The usage of metrics to gauge cost, schedule, and other performance

What are the eight principles for software engineering code of ethics

Software engineers shall: (1) Act consistently with the public interest (2) Act in a manner that is in the best interests of their client and employer (3) Ensure that their products and modifications meet the highest professional standards possible. (4) Maintain integrity and independence (5) Promote and ethical approach to the management of development and maintenance (6) Advance the integrity and reputation of the profession (7) Be fair and supportive of their colleagues (8) Participate in lifelong learning regarding the practice of their profession and promote an ethical approach to their practice.

What is the goal of a software process model?

Provides guidance for a systematic coordination and controlling of: (1) List of tasks to be accomplished (2) Input/Output of each task (3) Pre and Post-Conditions of each task (4) Sequence flow of the tasks (5) Personnel who perform the tasks

What are the entry and exit criteria to a process?

The entry criteria for a process that has to be met before initiating
The exit criteria are a set of conditions that must be met before the current process can be considered completed.

What motivated software engineers to move forward from the waterfall model to the incremental or spiral model?

The main problem with the Waterfall model was its difficulty in dealing with risk containment and identification. The incremental model addresses this by dividing the project into independent sections which are then incrementally developed. The Spiral model of development provides a risk-driven approach to the software process (Waterfall is a document-driven approach) by implementing similar concepts to the incremental model.

Explain some of the characteristics of Agile methodologies

(1) 'Short' releases and multiple iterations (2) Incremental design where design decisions are delayed as much as possible (3) Informal, written communication using internet tools (4) User involvement through the use of customer feedback (5) Expectation that any aspect of the project can and will likely change

Compare and contrast Agile and traditional methods.

Agile methods prioritize flexibility, collaboration, adaptability to change. Traditional methods prioritize predictivity, planning, and control
Agile*: (1) Requirements: Assumes changes (2) Planning: Not much planning (3) Scheduling: Only the next few activities are scheduled (4) Design: (informal and iterative) (5) User-involvement: crucial, frequent, throughout the whole process (6) Documentation: minimal, only what is necessary.*
Traditional Methods*: (1) Requirements: Assumes they will not change during the project. (2) Planning: Most activities planned upfront. (3) Scheduling: Schedules are inflexible and should be watched for (4) Design: Formal and done up front, after all requirements complete (5) User-involvement: Required only at beginning(requirements stage) and end (testing stage). (6) Documentation: Usually requires heavy formal doc of every phase of project.*

What is test-driven programming, and which Agile process advocates it?

(1) Test-driven programming: Ensure that testing is done continuously and is automated as much as possible. Write unit tests for all code. (2) XP (extreme programming) advocates it.

What are the six main dimensions of requirements that you need to address when collecting requirements

(1) Individual functionality (2) Business flow (3) Data, formats, and information needs (4) User interfaces (5) Interfaces with other systems (6) Performance, reliability, and security constraints

List and describe three items that you will need to consider when prioritizing requirements

(1) Current customer demands (2) Competition and current market condition (3) Future customer needs (4) Immediate sales advantage (5) Critical problems in existing product

What are the four types of requirements traceability?

(1) Forward from traceability: Links the requirement to design and implementation. (2) Backward from traceability: Links the requirement to the document source or the person who created it. (3) Forward to traceability: Links documents preceding the requirements to the requirements. (4) Backward to traceability: Links design and implementation back to the requirements.

Briefly explain the concepts of static analysis, and to which software products it can be applied.

Static analysis: Examination of the static structures of executable and non-executable files with the aim of detecting error-prone conditions.
Cases: (1) Intermediate documents: Check for completeness, traceability, and other characteristics. (2) Source code: Check for programming style issues or error-prone programming practices. (3) Executable files: to detect certain conditions, understanding that much information is usually lost in the translation from source code to executable files. (4) FindBugs is an open source bytecode checker for the Java language.

Consider the sample case of testing 2 variables, X and Y, where X must be a non-negative number, and Y must be a number between -5 and +15. Utilizing boundary value analysis, list the test cases.

Test Case 1: X = 10 Y = 0 (X is a non-negative and y is withing range.)
Test Case 2: X = 0 Y = -5 (Both the variables are at their min values)
Test Case 3: X = 0 Y = 0 (Min for X since its not negative and mid range of Y -5 to +15)
Test Case 4: X = -4 Y = 17 (Both variables are out of their min value range.)

Describe the steps involved in a formal inspection process and the role of a moderator in this process.

Planning *(Team and moderator are given the work product days prior to the meeting. Moderator makes sure the work satisfies some entry criteria.)* ***Overview*** *(Walk-through of the work product)* ***Preparation*** *(Every person is expected to study the work product thoroughly in preparation for the actual meeting)* ***Examination*** *(The meeting itself. Meeting is dedicated to finding defects. After meeting, product is either accepted as is, or corrected by the author with the moderator verifying the results)* ***Rework*** *(Author corrects all defects if any)* ***Follow-up*** *(Corrections are checked by moderator, or corrected work is inspected again)*

What is the difference between performance testing and stress testing?

Performance testing: Checks if a program meets its specified performance. The purpose of Performance testing is to identify potential issues and ensure the system meets performance requirements.
Stress testing: Checks if the program behaves correctly and handles high loads or low resource availability gracefully. Stress testing goes beyond performance specifications to identify breaking points.
*Differences: (****Purpose****) Performance testing is done to measure how well a system performs under normal conditions. Stress testing is done to measure how well a system performs under extreme conditions. (****Load****) Performance testing involves testing a system under varying levels of load to simulate normal operating conditions. Stress testing involves testing a system under higher load levels. (****Test scenario****) Performance testing is done using scenarios that simulate user behavior. Stress testing is done using scenarios that simulate abnormal user behavior. (****Results****) Performance testing provides information on how well a system performs under normal conditions. Stress testing provides information on how well a system performs under extreme conditions like overload and recovering from failures.*

What are the three components of risk management?

(1) Risk Identification (2) Risk Prioritization (3) Risk Mitigation

Using function point methodology, compute the unadjusted function point for an average project that has ten external inputs, seven external outputs, five external inquires, three internal logical files, and four external interface files.

Unadjusted Function Point: *(External Input x External Input Weight) + (External Output x External Output Weight) + (External Inquires x External Inquires Weight) + (Internal Logical Files x Internal Logical Files Weight) + (External Interface Files x External Interface Files Weight)*
= (10 x 4) + (7 x 5) + (5 x 4) + (3 x 10) + (4 x 7)
= 40 + 35 + 20 + 30 + 28 = 153

Assume that the total complexity factor for the software project in the previous question is 1.1 and the productivity figure is 20 function points per person-month. What is the function point for the project? What is the estimated effort in person-months for the project?

*To calculate the estimated effort in person-months for the project we need to first find the function point which the calculation is Unadjusted Function Point * Total Complexity Factor: 153 x 1.1 = 168.3*
The calculation for Estimated Effort is as such: Function Point / Productivity where we are shown the function point per person-month is 20.
So Estimated Effort is 168.3 / 20 = 8.415 = 8.42