client?s business team. System Test ensures that expectations from an application developer are met.

4. **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

When to use V-Model?

- When the requirement is well defined and not ambiguous.
- The V-shaped model should be used for small to medium-sized projects where requirements are clearly defined and fixed.
- The V-shaped model should be chosen when sample technical resources are available with essential technical expertise.

Advantage (Pros) of V-Model:

1. Easy to Understand.
2. Testing Methods like planning, test designing happens well before coding.
3. This saves a lot of time. Hence a higher chance of success over the waterfall model.
4. Avoids the downward flow of the defects.
5. Works well for small plans where requirements are easily understood.

Disadvantage (Cons) of V-Model:

1. Very rigid and least flexible.
2. Not a good for a complex project.
3. Software is developed during the implementation stage, so no early prototypes of the software are produced.
4. If any changes happen in the midway, then the test documents along with the required documents, has to be updated.

Spiral Model:

The spiral model, initially proposed by Boehm, is an evolutionary software process model that couples the iterative feature of prototyping with the controlled and systematic aspects of the linear sequential model. It implements the potential for rapid development of new versions of

the software. Using the spiral model, the software is developed in a series of incremental releases. During the early iterations, the additional release may be a paper model or prototype. During later iterations, more and more complete versions of the engineered system are produced.

The Spiral Model is shown in fig:

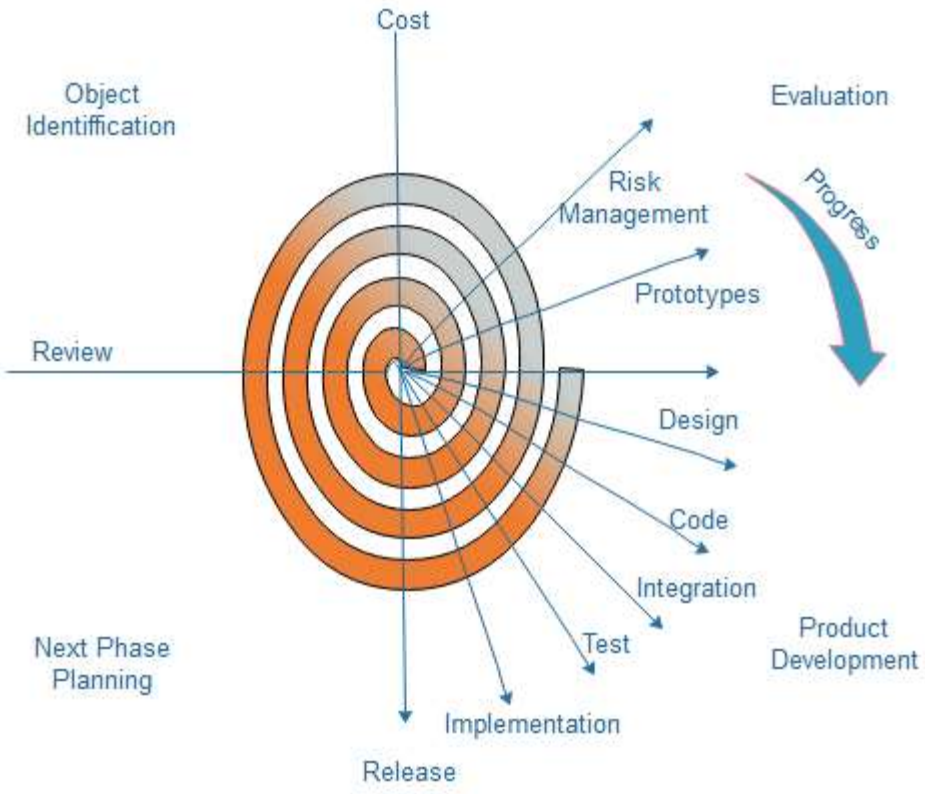


Fig. Spiral Model

Each cycle in the spiral is divided into four parts:

Objective setting: Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.

Risk Assessment and reduction: The next phase in the cycle is to calculate these various alternatives based on the goals and constraints. The focus of evaluation in this stage is located on the risk perception for the project.

Development and validation: The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.

Planning: Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project.

The development phase depends on the remaining risks. For example, if performance or user-interface risks are treated more essential than the program development risks, the next phase may be an evolutionary development that includes developing a more detailed prototype for solving the risks.

The **risk-driven** feature of the spiral model allows it to accommodate any mixture of a specification-oriented, prototype-oriented, simulation-oriented, or another type of approach. An essential element of the model is that each period of the spiral is completed by a review that includes all the products developed during that cycle, including plans for the next cycle. The spiral model works for development as well as enhancement projects.

When to use Spiral Model?

- When deliverance is required to be frequent.
- When the project is large
- When requirements are unclear and complex
- When changes may require at any time
- Large and high budget projects

Advantages

- High amount of risk analysis
- Useful for large and mission-critical projects.

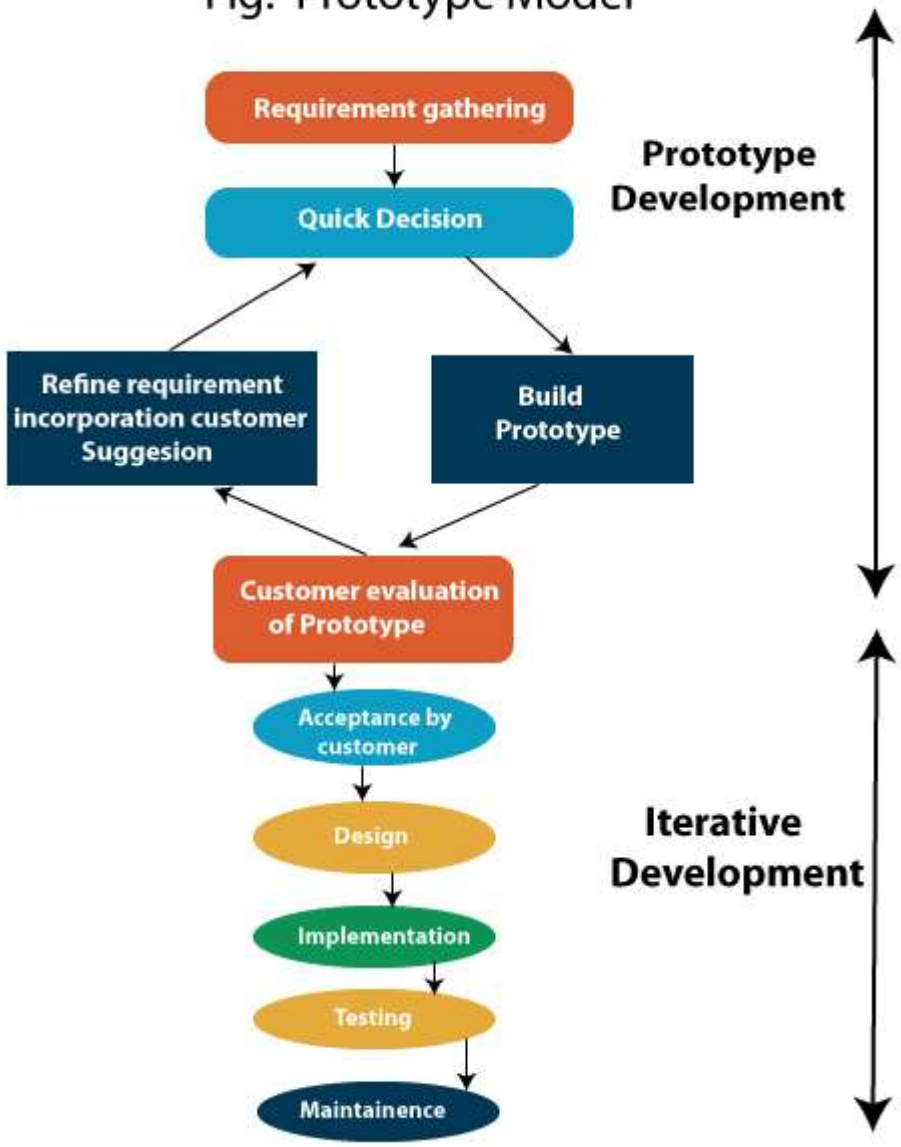
Disadvantages

- Can be a costly model to use.
- Risk analysis needed highly particular expertise
- Doesn't work well for smaller projects.

Prototyping Delivery:

The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software. In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.

Fig: Prototype Model



Steps of Prototype Model

1. Requirement Gathering and Analyst
2. Quick Decision
3. Build a Prototype
4. Assessment or User Evaluation
5. Prototype Refinement
6. Engineer Product

Advantage of Prototype Model

1. Reduce the risk of incorrect user requirement

2. Good where requirement are changing/uncommitted
3. Regular visible process aids management
4. Support early product marketing
5. Reduce Maintenance cost.
6. Errors can be detected much earlier as the system is made side by side.

Disadvantage of Prototype Model

1. An unstable/badly implemented prototype often becomes the final product.
2. Require extensive customer collaboration
    - Costs customer money
    - Needs committed customer
    - Difficult to finish if customer withdraw
    - May be too customer specific, no broad market
3. Difficult to know how long the project will last.
4. Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
5. Prototyping tools are expensive.
6. Special tools & techniques are required to build a prototype.
7. It is a time-consuming process.

Evolutionary Process Model

Evolutionary process model resembles the iterative enhancement model. The same phases are defined for the waterfall model occurs here in a cyclical fashion. This model differs from the iterative enhancement model in the sense that this does not require a useful product at the end of each cycle. In evolutionary development, requirements are implemented by category rather than by priority.

For example, in a simple database application, one cycle might implement the graphical user Interface (GUI), another file manipulation, another queries and another updates. All four cycles must complete before there is a working product available. GUI allows the users to interact with the system, file manipulation allow the data to be saved and retrieved, queries allow user to get out of the system, and updates allows users to put data into the system.

Benefits of Evolutionary Process Model

Use of EVO brings a significant reduction in risk for software projects.EVO can reduce costs by providing a structured, disciplined avenue for experimentation.

EVO allows the marketing department access to early deliveries, facilitating the development of documentation and demonstration.

Better fit the product to user needs and market requirements.

Manage project risk with the definition of early cycle content.

Uncover key issues early and focus attention appropriately.

Increase the opportunity to hit market windows.

Accelerate sales cycles with early customer exposure.

Increase management visibility of project progress.

Increase product team productivity and motivations.

Albrecht function point analysis:

Function point metrics provide a standardized method for measuring the various functions of a software application. It measures the functionality from the user's point of view, that is, on the basis of what the user requests and receives in return. Function point analysis is a standard method for measuring software development from the user's point of view.

The Function Point measure originally conceived by Albrecht received increased popularity with the inception of the International Function Point Users Group (IFPUG) in 1986. In 2002, IFPUG Function Points became an international ISO standard – ISO/IEC 20926.

What is a Function Point?

FP (Function Point) is the most widespread functional type metrics suitable for quantifying a software application. It is based on five users identifiable logical "functions", which are divided into two data function types and three transactional function types. For a given software application, each of these elements is quantified and weighted, counting its characteristic elements, such as file references or logical fields.

The resulting numbers (Unadjusted FP) are grouped into Added, Changed, or Deleted functions sets, and combined with the Value Adjustment Factor (VAF) to obtain the final number of FP. A distinct final formula is used for each count type: Application, Development Project, or Enhancement Project.

Applying Albrecht's Function Point Method

Let us now understand how to apply the Albrecht's Function Point method. Its procedure is as follows −

Determine the number of components (EI, EO, EQ, ILF, and ELF)

- **EI** − The number of external inputs. These are elementary processes in which derived data passes across the boundary from outside to inside. In an example library database system, enter an existing patron's library card number.
- **EO** − The number of external output. These are elementary processes in which derived data passes across the boundary from inside to outside. In an example library database system, display a list of books checked out to a patron.
- **EQ** − The number of external queries. These are elementary processes with both input and output components that result in data retrieval from one or more internal logical files and external interface files. In an example library database system, determine what books are currently checked out to a patron.
- **ILF** − The number of internal log files. These are user identifiable groups of logically related data that resides entirely within the applications boundary that are maintained through external inputs. In an example library database system, the file of books in the library.
- **ELF** − The number of external log files. These are user identifiable groups of logically related data that are used for reference purposes only, and which reside entirely outside the system. In an example library database system, the file that contains transactions in the library's billing system.

Compute the Unadjusted Function Point Count (UFC)

- Rate each component as **low, average,** or **high**.
- For transactions **(EI, EO, and EQ)**, the rating is based on **FTR** and **DET**.
  - **FTR** − The number of files updated or referenced.
  - **DET** − The number of user-recognizable fields.
  - Based on the following table, an **EI** that references 2 files and 10 data elements would be ranked as **average**.

FTRs	DETs		
	1-5	**6-15**	**>15**
0-1	Low	Low	Average
2-3	Low	Average	High
>3	Average	High	High

- For files **(ILF and ELF)**, the rating is based on the **RET** and **DET**.
  - **RET** − The number of user-recognizable data elements in an **ILF** or **ELF**.
  - **DET** − The number of user-recognizable fields.
  - Based on the following table, an **ILF** that contains 10 data elements and 5 fields would be ranked as **high**.

RETs	DETs		
	1-5	**6-15**	**>15**

1	Low	Low	Average
2-5	Low	Average	High
>5	Average	High	High

- Convert ratings into **UFCs**.

Rating	**Values**				
	EO	**EQ**	**EI**	**ILF**	**ELF**
Low	4	3	3	7	5
Average	5	4	4	10	7
High	6	5	6	15	10

Compute the Final Function Point Count (FPC)

- Compute value adjustment factor **(VAF)** based on 14 general system characteristics **(GSC)**.

General System Characteristic		**Brief Description**
GSC 1	Data communications	How many communication facilities are there to aid in the transfer or exchange of information with the application or system?
GSC 2	Distributed data processing	How are distributed data and processing functions handled?
GSC 3	Performance	Was the response time or throughput required by the user?
GSC 4	Heavily used configuration	How heavily used is the current hardware platform where the application will be executed?
GSC 5	Transaction rate	How frequently are transactions executed daily, weekly, monthly, etc.?

GSC 6	On-Line data entry	What percentage of the information is entered online?
GSC 7	End-user efficiency	Was the application designed for end-user efficiency?
GSC 8	On-Line update	How many ILFs are updated by online transaction?
GSC 9	Complex processing	Does the application have extensive logical or mathematical processing?
GSC 10	Reusability	Was the application developed to meet one or many user's needs?
GSC 11	Installation ease	How difficult is conversion and installation?
GSC 12	Operational ease	How effective and/or automated are start-up, back-up, and recovery procedures?
GSC 13	Multiple sites	Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?
GSC 14	Facilitate change	Was the application specifically designed, developed, and supported to facilitate change?

- Weigh each **GSC** on a scale of 0 to 5 based on whether it has no influence to strong influence.
- Compute the **FPC** as follows −

$$\textbf{FPC} = \textbf{UFC} * (0.65+(\text{sum}(\textbf{GSC}) * .01))$$

Complexity

Complexity is a separate component of size. It is of two types −

- **Complexity of a problem** − It is the amount of resources required for an optimal solution to the problem.
- **Complexity of a solution** − It is the resources needed to implement a particular solution. It has two aspects. They are as follows −
  - **Time complexity** − The resource is computer time.
  - **Space complexity** − The resource is computer memory.

Measuring Complexity

One aspect of complexity is efficiency. It measures any software product that can be modeled as an algorithm.

For example: If an algorithm for solving all instances of a particular problem requires **f(n)** computations, then **f(n)** is asymptotically optimal, if for every other algorithm with complexity g that solves the problem **f** is **O(g)**. Then, the complexity of the given problem is big - **O** of the asymptotically optimal algorithm for the problem's solution.