3. Evolutionary Process Models

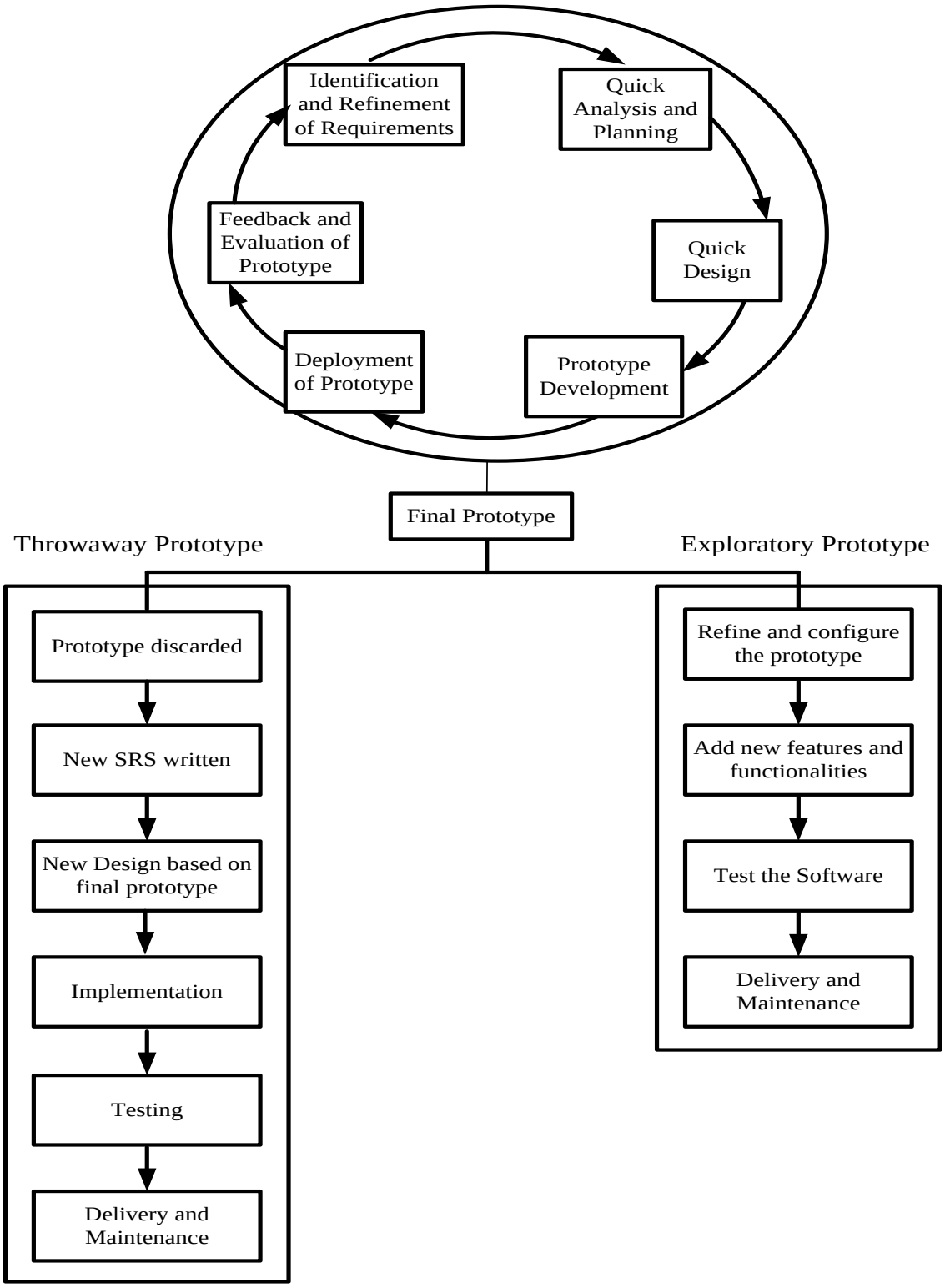
- Software, like all complex systems, evolves over a period of time.
- Business and product requirements often change as development proceeds:
  - making a straight line path to an end product unrealistic;
  - tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure;
  - a set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined.
- In these and similar situations, you need a process model that has been explicitly designed to accommodate a product that evolves over time.
- Evolutionary models are iterative.
- They are characterized in a manner that enables you to develop increasingly more complete versions of the software.

There are two types of **Evolutionary Process Models**: Prototyping and Spiral Model

3.1 Prototype Model

- Often, a customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements for functions and features.
- In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human/machine interaction should take.
- In these, and many other situations, a *prototyping paradigm* may offer the best approach.
- The prototyping paradigm begins with requirements gathering.
- Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.
- A "quick design" then occurs.
- The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g., input approaches and output formats).
- The quick design leads to the construction of a prototype.
- The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed.
- Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.
- Ideally, the prototype serves as a mechanism for identifying software requirements.
- If a working prototype is built, the developer attempts to use existing program fragments or applies tools (e.g., report generators, window managers) that enable working programs to be generated quickly.
- The prototype can serve as "the first system." The one that Brooks recommends we throw away. But this may be an idealized view. It is true that both customers and developers like the prototyping paradigm.
- Users get a feel for the actual system and developers get to build something immediately.

Based on the way of implementation, and the requirement of the customers there are two approaches of prototyping: Throwaway prototyping and Exploratory prototyping



i) Throwaway prototyping

- Once the development team and the customer finalize the prototype, the prototype is kept aside.
- Taking the final prototype as basis model, new SRS with detailed configurations is written.
- A quick and new design is built with new tools and technologies.
- The finalized prototype is discarded and exactly the same operational software is developed.
- The discarded prototype works as a dummy guide, which is used to refine the requirements, as shown in Figure.

ii) Exploratory Prototyping

- In exploratory prototyping, the finalized prototype is not discarded; rather it is treated as the semifinished product.
- New features and additional functionalities are added to make the prototype operational.
- After reviews by the customer and tested by the developer it is deployed to the customer's site.

Advantages of Prototype Model:

- Prototyping is the best suitable paradigm for requirements elicitation from the customers.
- Non technical customers may be unaware or uncertain about the requirements of their software, after seeing working or dummy prototypes of similar products, customers can explore more specific requirements.
- Once the development team and customer finally agreed on a particular prototype then the development of final product takes minimum time in comparison to other paradigms of software development.
- Analysis, design or coding faults made by the development team can be identified early in the development phases, and can be rectified without delay.
- It improves the overall life cycle of the software.
- Out of all the traditional development paradigms, customer's involvement is at the highest level in prototyping model.
- In all iterations customer's feedback is regarded and incorporated in the next prototype iteration.
- Prototyping is one of the most flexible development paradigms in terms of incorporating changes or extensions in the requirements.

Disadvantages of Prototype Model:

- One of the major criticisms of prototype paradigm is the concern of long run quality issues. Sometimes in the rush of delivering operational software, quality is compromised.
- To fulfil customer's demand as early as possible, inefficient, incompetent or poor programming languages can be chosen, inappropriate database are selected, or the developer may chose the technologies according to his convenience rather than as the demand of the proposed software.
- If customer's demand changes too frequently then the time of developing prototype is larger than the development of the actual software.
- Sometimes it is difficult to convince other stakeholders about the prototype paradigm of development.
- When after a period of time customer is informed that the developed product so far is only a prototype, he may raise objections regarding time and cost invested.
- For large and complex projects prototype model may become unmanageable.