Software Engineering I

Process Models

Dr. Elham Mahmoudzadeh
Isfahan University of Technology
mahmoudzadeh@iut.ac.ir
202۲

Software Process models

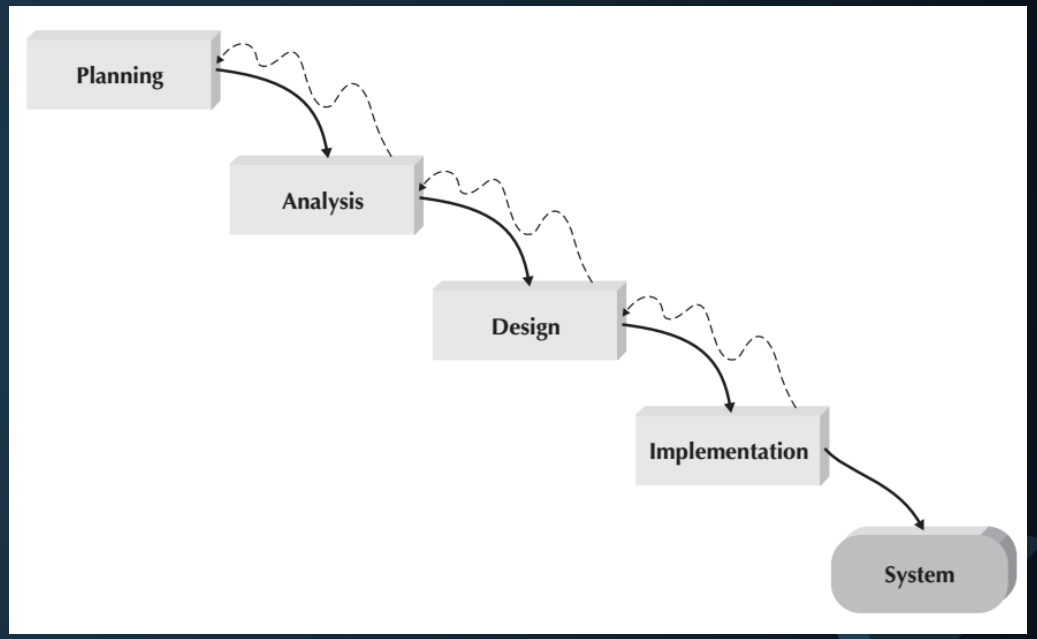
- Describes the sequences of SDLC steps.
- Is a sequence of activities that leads to the production of a software product.

Process Models types

- Structured
  - Waterfall
  - parallel
- RAD
  - Phased
  - Prototyping
  - Throwaway-prototyping
- Agile

Waterfall(I)

- Proceed in sequence from one phase to the next.
- The key deliverables for each phase are typically very long (often hundreds of pages in length) and are presented to the project sponsor for approval as the project moves from phase to phase. Once the sponsor approves the work that was conducted for a phase, the phase ends and the next one begins.
- It moves forward from phase to phase in the same manner as a waterfall. Although it is possible to go backward in the SDLC (e.g., from design back to analysis), it is extremely difficult.
- Because of the cascade from one phase to another, this model is known as the ‘waterfall model.’



Waterfall(II)

- Key advantages
  - It identifies system requirements long before programming begins.
  - It minimizes changes to the requirements as the project proceeds.

- Key disadvantages
  - Design must be completely specified before programming begins
  - A long time elapses between the completion of the system proposal in the analysis phase and the delivery of the system (usually many months or years).
  - If the project team misses important requirements, expensive post-implementation programming may be needed. A system can also require significant rework because the business environment has changed from the time when the analysis phase occurred.

Waterfall Main drawback

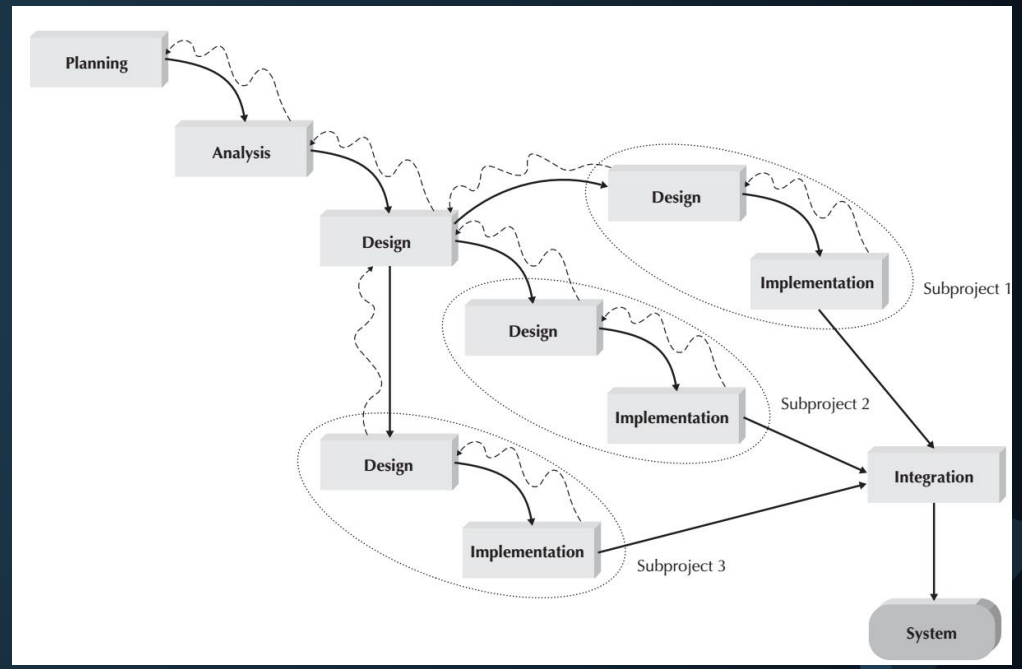
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway.
- In principle, a phase has to be complete before moving onto the next phase.

Waterfall Model - Usage

- In principle, the waterfall model should only be used when the requirements are well understood and unlikely to change radically during system development.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
- In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Parallel Development(I)

- Attempts to address the problem of long delays between the analysis phase and the delivery of the system.
- Instead of doing design and implementation in sequence, it performs a general design for the whole system.
- Then divides the project into a series of distinct subprojects that can be designed and implemented in parallel.
- Once all subprojects are complete, the separate pieces are integrated and the system is delivered.



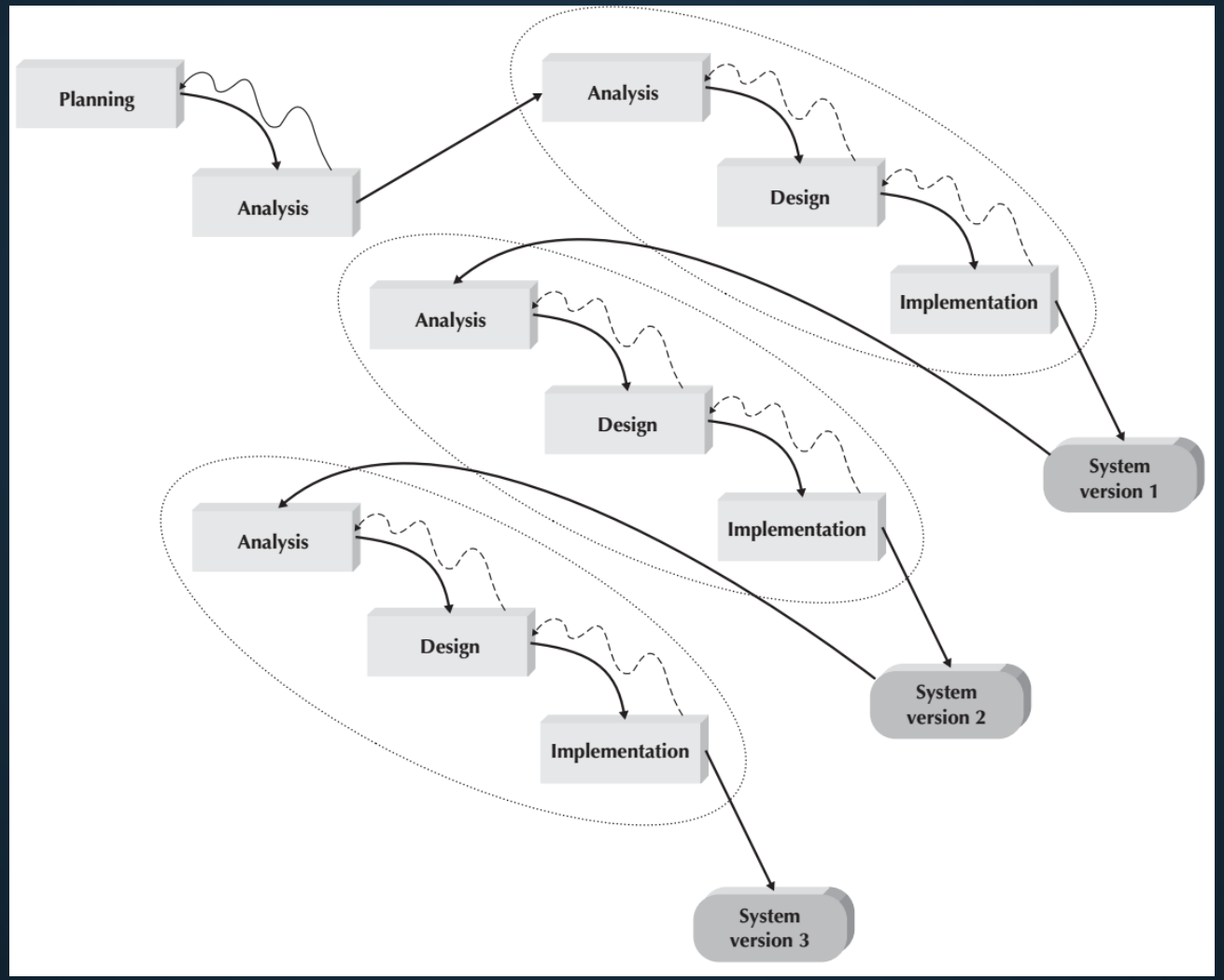
Parallel Development(III)

- The primary advantage is that it can reduce the time to deliver a system;
- However, sometimes the subprojects are not completely independent; design decisions made in one subproject can affect another.
- At the end of the project, it requires significant integration efforts.

Phased Development(I)

- Breaks an overall system into a series of *versions* that are developed sequentially. The analysis phase identifies the overall system concept, and the project team, users, and system sponsor then categorize the requirements into a series of versions.
- The most important and fundamental requirements are bundled into the first version of the system.
- The analysis phase then leads into design and implementation—but only with the set of requirements identified for version 1. Once version 1 is implemented, work begins on version 2. Additional analysis is performed based on the previously identified requirements and combined with new ideas and issues that arose from the users' experience with version 1. Version 2 then is designed and implemented, and work immediately begins on the next version. This process continues until the system is complete or is no longer in use.

Phased Development(II)



Phased Development(III)

- It has the advantage of quickly getting a useful system into the hands of the users.
- Although the system does not perform all the functions the users need at first, it does begin to provide business value sooner than if the system were delivered after completion.
- Likewise, because users begin to work with the system sooner, they are more likely to identify important additional requirements sooner than with structured design situations.
- The major drawback is that users begin to work with systems that are intentionally incomplete. It is critical to identify the most important and useful features and include them in the first version and to manage users' expectations along the way.

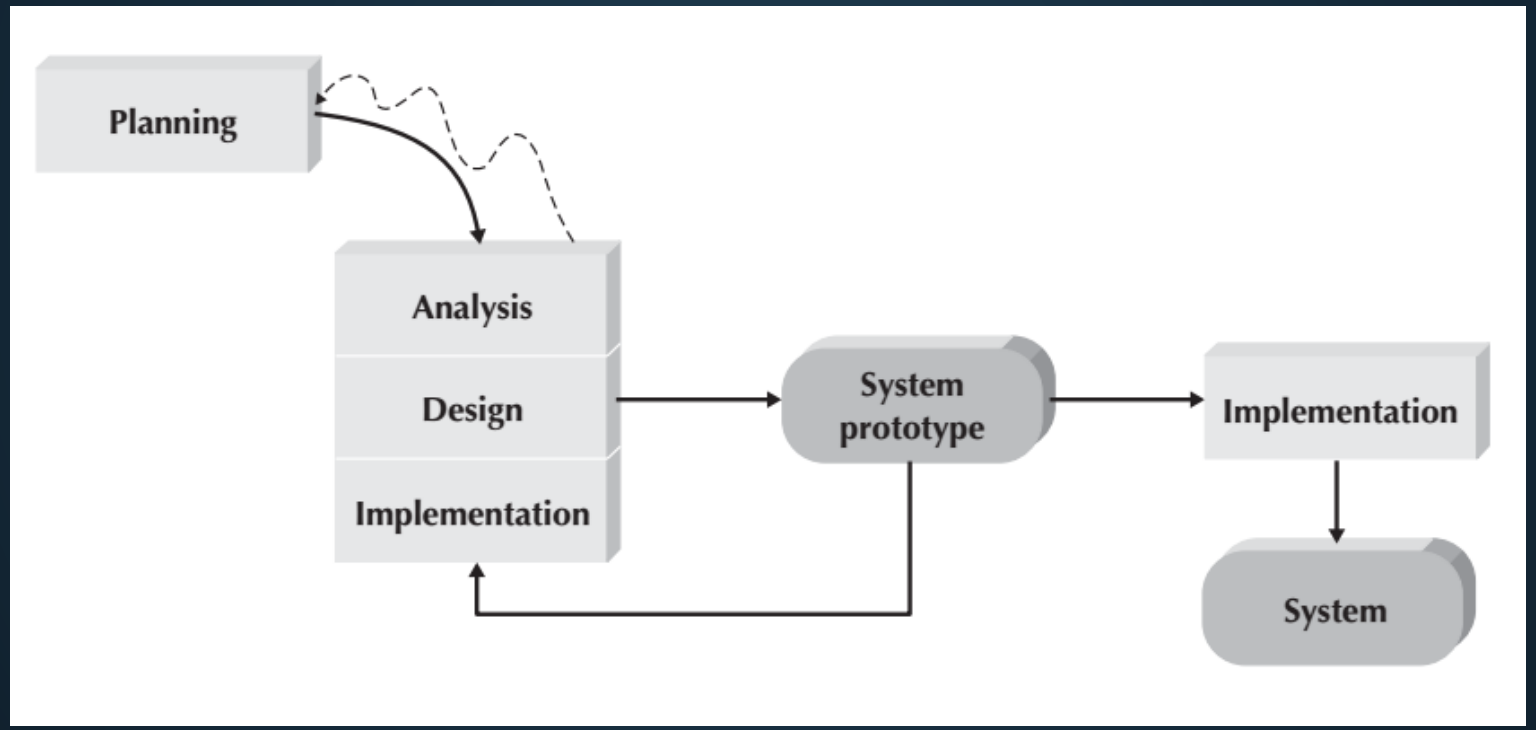
Prototyping (I)

- Performs the analysis, design, and implementation phases concurrently, and all three phases are performed repeatedly in a cycle until the system is completed.
- Basics of analysis and design are performed, and work immediately begins on a *system prototype*, a quick-and-dirty program that provides a minimal amount of features. The first prototype is usually the first part of the system that is used.

- This is shown to the users and the project sponsor, who provide comments. These comments are used to reanalyze, redesign, and re-implement a second prototype, which provides a few more features. This process continues in a cycle until the analysts, users, and sponsor agree that the prototype provides enough functionality to be installed and used in the organization.

- After the prototype (now called the “system”) is installed, refinement occurs until it is accepted as the new system.

Prototyping(II)



Prototyping(III)

- The key advantage is that it *very* quickly provides a system with which the users can interact, even if it is not ready for widespread organizational use at first.
- Prototyping reassures the users that the project team is working on the system (there are no long delays in which the users see little progress), and prototyping helps to more quickly refine real requirements.
- The major problem is that its fast-paced system releases challenge attempts to conduct careful, methodical analysis. Often the prototype undergoes such significant changes that many initial design decisions become poor ones. This can cause problems in the development of complex systems because fundamental issues and problems are not recognized until well into the development process.

Throwaway Prototyping(I)

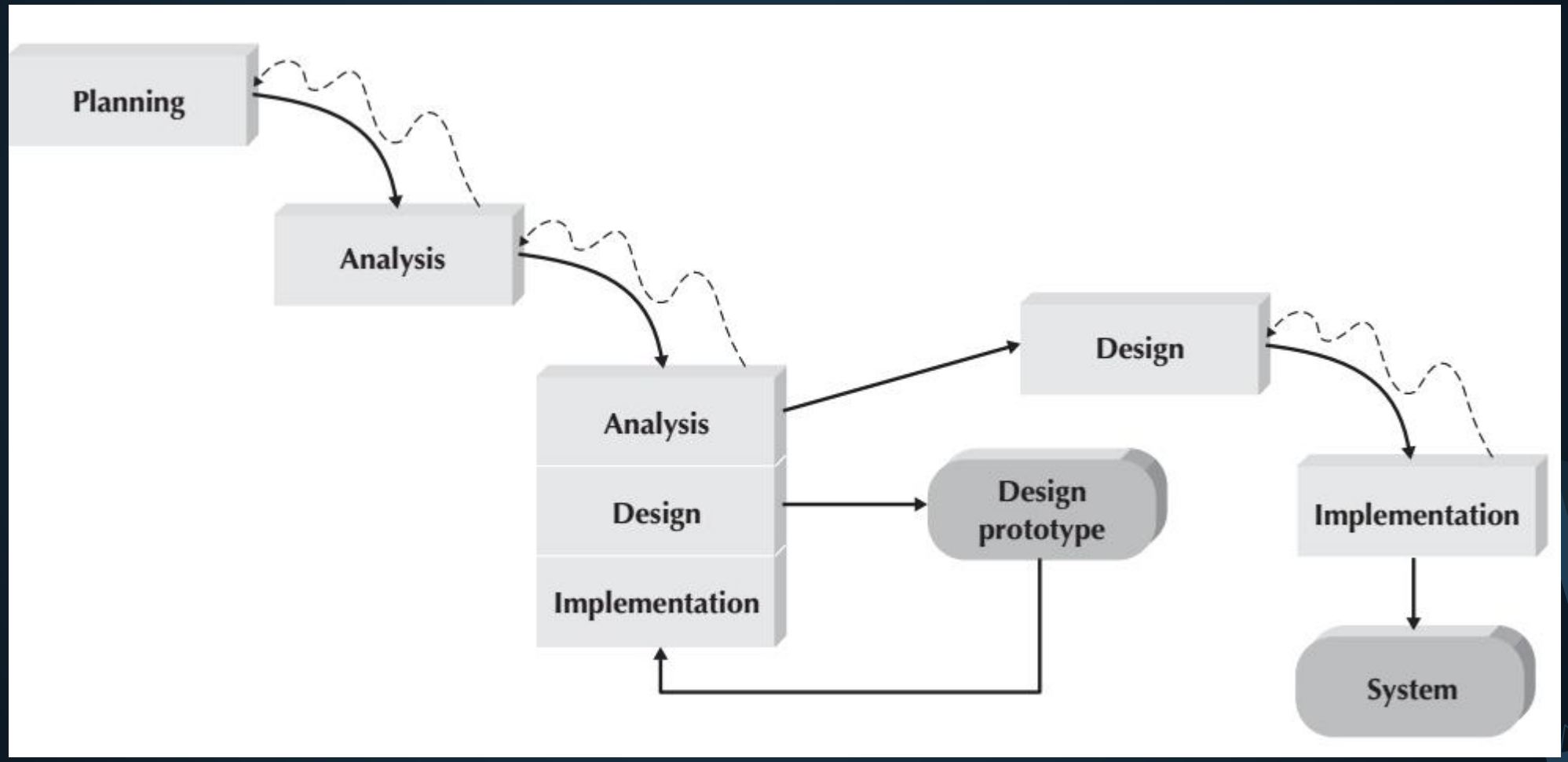
- These prototypes are used for a very different purpose than those previously discussed, and they have a very different appearance.

- It has a relatively thorough analysis phase that is used to gather information and to develop ideas for the system concept. However, users might not completely understand many of the features they suggest, and there may be challenging technical issues to be solved. Each of these issues is examined by analyzing, designing, and building a *design prototype*. A design prototype is not a working system; it is a product that represents a part of the system that needs additional refinement, and it contains only enough detail to enable users to understand the issues under consideration.

Throwaway Prototyping(II)

- Each of the prototypes is used to minimize the risk associated with the system by confirming that important issues are understood before the real system is built.
- Once the issues are resolved, the project moves into design and implementation. At this point, the design prototypes are thrown away, which is an important difference between these methodologies and prototyping methodologies, in which the prototypes evolve into the final system.
- It can take longer to deliver the final system as compared to prototyping-based methodologies, but produces more stable and reliable systems.

Throwaway Prototyping(III)

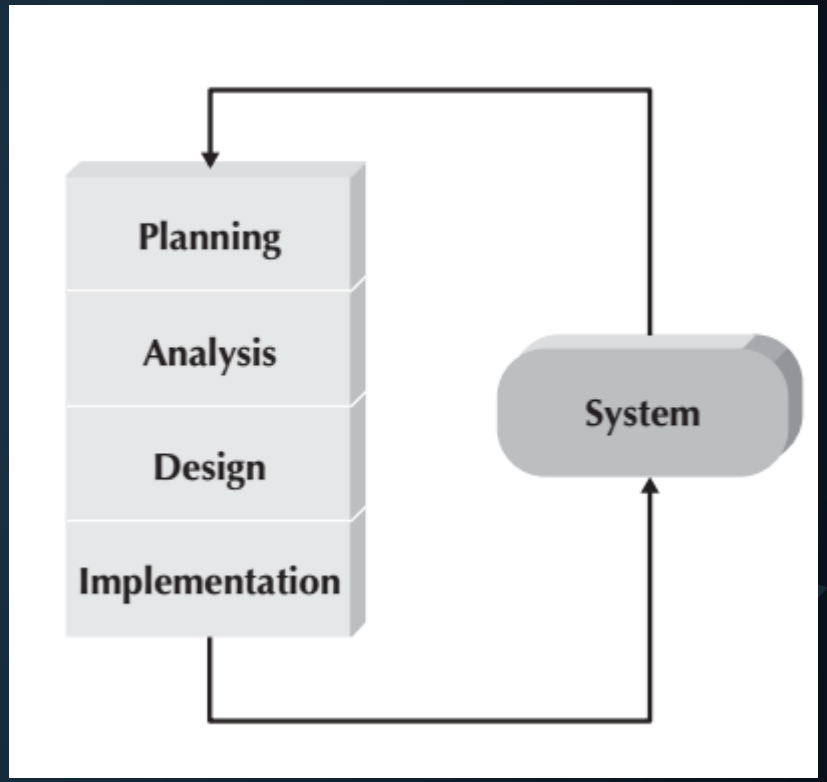


Agile

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.



Selecting the Appropriate Development Methodology

Ability to Develop Systems	Structured Methodologies		RAD Methodologies			Agile Methodologies	
	Waterfall	Parallel	Phased	Prototyping	Throwaway Prototyping	XP	SCRUM
With Unclear User Requirements	Poor	Poor	Good	Excellent	Excellent	Excellent	Excellent
With Unfamiliar Technology	Poor	Poor	Good	Poor	Excellent	Good	Good
That Are Complex	Good	Good	Good	Poor	Excellent	Good	Good
That Are Reliable	Good	Good	Good	Poor	Excellent	Excellent	Excellent
With a Short Time Schedule	Poor	Good	Excellent	Excellent	Good	Excellent	Excellent
With Schedule Visibility	Poor	Poor	Excellent	Excellent	Good	Excellent	Excellent

انتخاب روش توسعه مناسب:
1- اگر requirment ها شفاف نباشه:
موازی: نه چون تایم تو مارکت خیلی زیاده و اگر شفاف نباشه و اخر کار بخوایم به کارفرما بدیم ممکنه رفته باشیم توی مسیری که درست نبوده و کلی انرژی گذاشتیم
phased : بینابین است
اون هایی که سریع هستند برای این روش مناسب هستند چون کمک می کنه به فهم requirmentها
2- اگر تکنولوژی رو ندونیم ینی نمی تونیم پلن رو از اول درست بچینیم
throwaway : چون یک دست گرمی اولش انجام میدیم و بعد می ذاریمش کنار و از اول می سازیمش
3- اگر پیچیده باشد:
ابشاری: از یک جهت مناسب است چون همون اولش داریم خیلی خوب تحلیل و طراحیشو انجام میدیم و از یک طرف دیگه چون بازخورد رو خیلی دیر میگیریم ممکنه مسیرو اشتباه رفته باشیم و امکان بازگشت با عقب خیلی پر هزینه است مناسب نیست
throwaway: اول کار میکنیم روی فهم و بعد روی اصل سیستم
4- اطمینان از عملکرد نرم افزار: ینی میزان پاسخ های درستش مهمه چند باشه
اونجاهایی که Reliablity خیلی اهمیت داره بهترینش اینه که throwaway باشه وagile بخاطر تست مکرر و بررسی کیفیت نرم افزار خیلی داریم بررسی میکنیم و بازخورد می گیریم و بخش مهم Reliablity هم براساس همین بازخورد است
5- اگر تایم اسکجولینگمون کم باشه ینی بازه زمانی که میخوایم تحویل بدیم کم باشه:
6- قابل مشاهده و اندازه گیری باشه ینی کارفرما حس اینو پیدا بکنه که پروژه داره می ره جلو:

Software Development Methodology(SDM)

- A framework for applying software engineering practices with the specific aim of providing the necessary means for developing software-intensive systems.
- Have two parts.
  1. A set of modeling conventions comprising a Modeling Language (syntax and semantics)
  2. A Process, which
     - provides guidance as to the order of the activities,
     - specifies what artifacts should be developed using the Modeling Language,
     - directs the tasks of individual developers and the team as a whole,
     - offers criteria for monitoring and measuring a project's products and activities.

Unified Modelling Language (UML)

- Each developer had his or her own methodology and notation.
- A standard set of diagramming techniques, *Unified Modeling Language(UML)*.
- The objective of UML was to provide a common vocabulary of object-oriented terms and diagramming techniques rich enough to model any systems development project from analysis through implementation.

References

- Dennis, Wixon, Tegarden, “System Analysis and Design, An Object Oriented Approach with UML”, 5th Edition, 2015.

What we will talk about next...

- Object-oriented principles
- RUP
- Scrum