Software Engineering

Lecture 2.2

Common Software Process Models

SAURABH SRIVASTAVA

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IIT (ISM) DHANBAD

Revision – Process Flows

We discussed some process flows in the last Lecture

- We talked about Linear, Iterative, Evolutionary and Parallel Process Flows

In reality, it is hard to follow any particular flow religiously

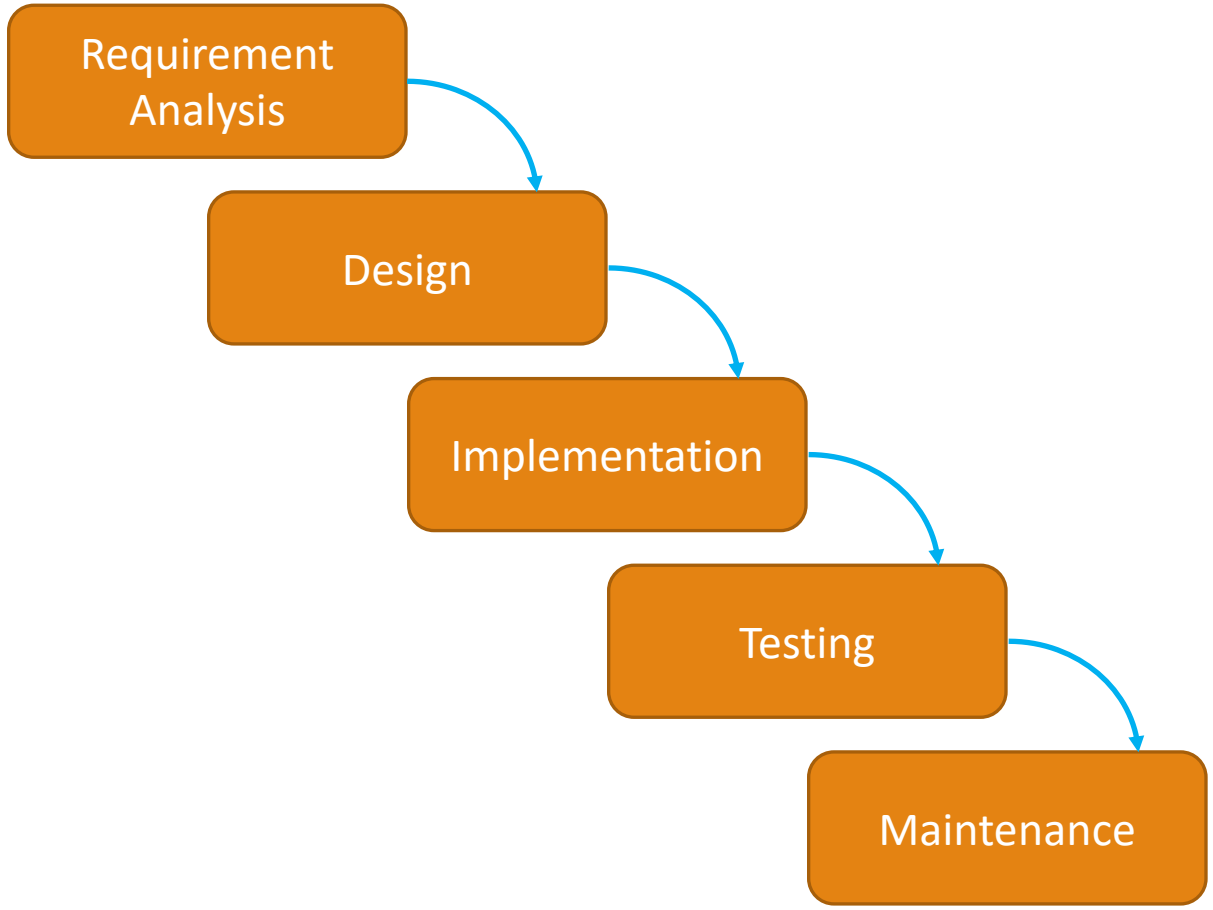
- All practice, the actual Process flows are amalgamations of the different flows, at different stages

In this lecture, we will talk about some common Software Process Models

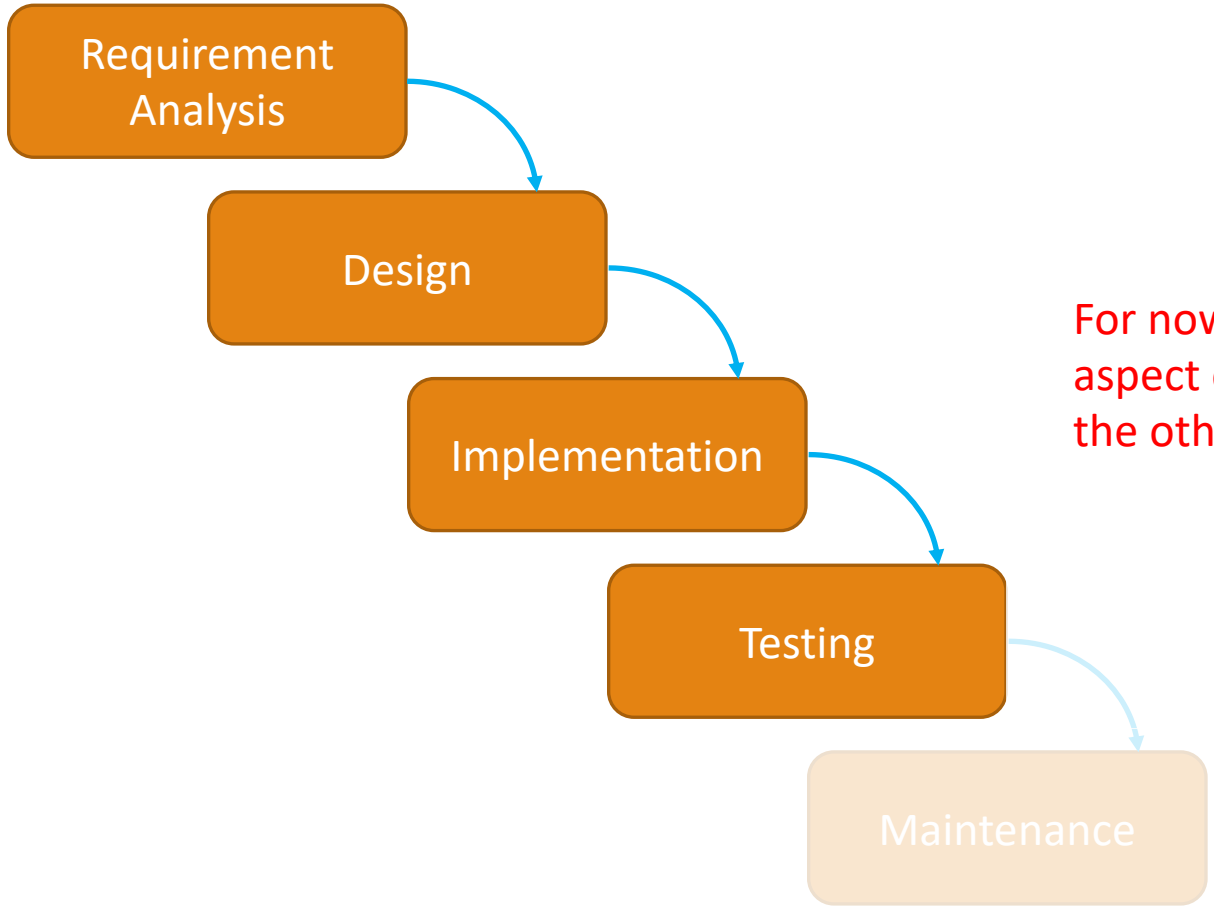
- Keep in mind that an actual Development Process may be “close” to any one of the common models …
- … but may not resemble it precisely

The Waterfall Model of development

Requirement analysis
Design
Implementation
Testing
Maintenance

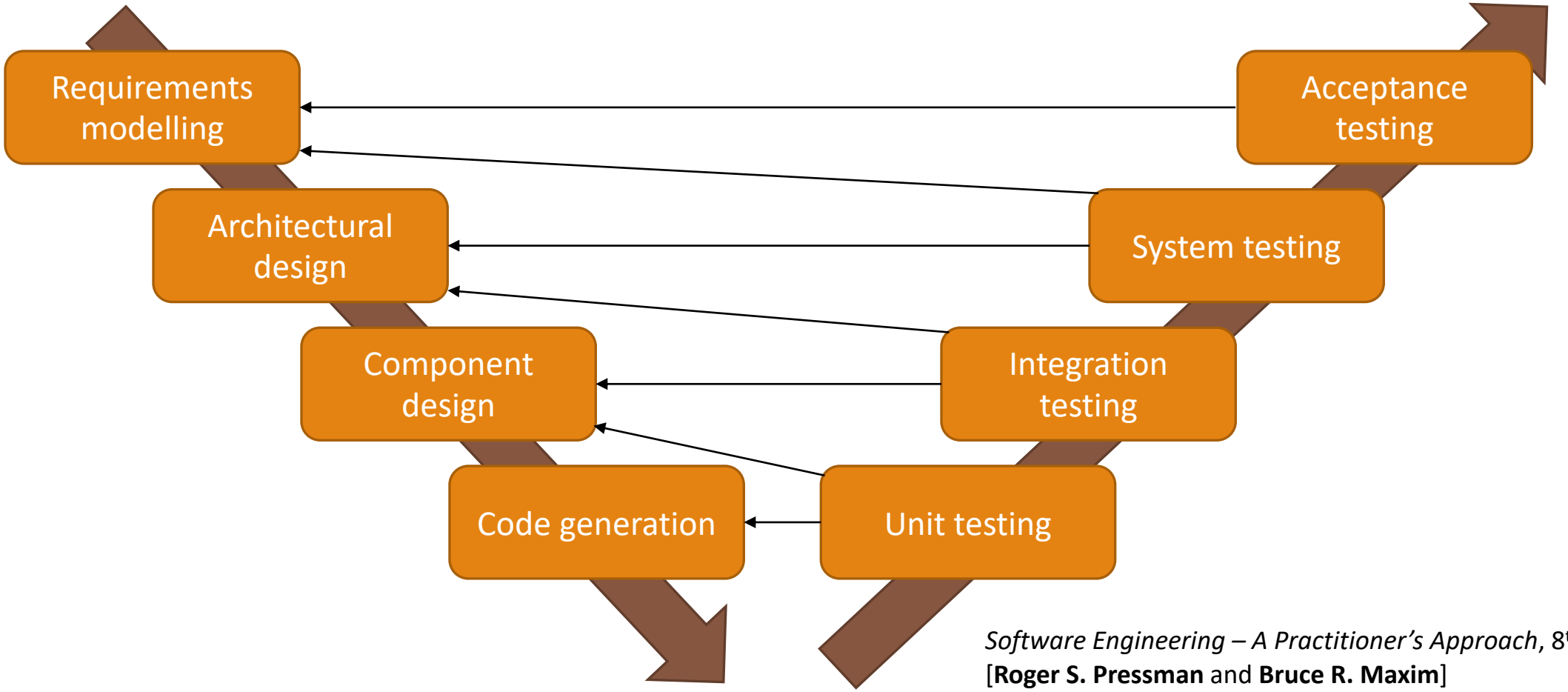


The Waterfall Model of development



For now, let us leave the Maintenance aspect of the model, and focus more on the other aspects

Variation of Waterfall Model – The **V** Model



Software Engineering – A Practitioner's Approach, 8th Edition
[**Roger S. Pressman** and **Bruce R. Maxim**]

Waterfall Model and Quality Assurance

The V Model can be seen as the “Waterfall Model with Quality Assurance” tasks

- The development proceeds sequentially as the Waterfall Model prescribes
- At the end of the implementation phase, a rigorous process is initiated to “test” the product
- The tests are considered as part of the overall activity to maintain Quality of the product

Different types of QA tasks are associated for outcomes of different activities

- The *Acceptance Test* also called User Acceptance Test is performed over the software product as a whole
- Its objective is to verify – usually in the presence of the user – that the product meets all its requirements
- The *Unit Tests* are localised tests that (generally) test the functionality of a small piece of the software
- They are usually performed by the developer of the code fragment, loosely referred to as a “unit”
- The *Integration Tests* check the compatibility of a unit with other units
- They may be performed within Development Teams or across multiple Teams
- A set of *System Tests* check end-to-end functionality of the software product
- They usually cover invocations of code fragments from multiple units

Incremental and Evolutionary Models (1/2)

Clearly, the Waterfall Model may be too rigid for a changing world like ours

- Thus, the activities that we discussed as part of the Waterfall Model, may not be doable in a sequence

There are two alternatives that are more practical – *Incremental* and *Evolutionary* models

- Note that you can see "evolutionary process flow" in both type of models

Incremental Models build a product in *increments*

- It may be used when the requirements are "mostly well-understood" but there are resource or time constraints
- The development team picks up some slices of these requirements and prepare a working version of the product
- The user can evaluate these "stripped-down" versions of the final product and provide feedbacks
- In the next iteration, a newer, richer version of the product is prepared, achieving more slices of requirements
- There is a possibility to include new requirements or make slight changes in the requirements …
- … as long as the overall product requirements are not changed drastically

Incremental and Evolutionary Models (2/2)

Clearly, the Waterfall Model may be too rigid for a changing world like ours

- Thus, the activities that we discussed as part of the Waterfall Model, may not be doable in a sequence

There are two alternatives that are more practical – *Incremental* and *Evolutionary* models

- Both these models are *iterative* in nature – i.e., one or more activities may be performed multiple times

Evolutionary Models build a product in *iterations*

- It may be used when the requirements are "not clearly stated or known" but the development has to be started
- The development team starts development of parts of the software, for which the requirements are clear(er)
- A popular Evolutionary Model, called the *Spiral Model* stresses on doing iterations with a consideration to *Risks*
- It is because it *may* be possible that the product development fails due to a lack of understanding
- Similar to the Incremental Models, the earlier versions of the product may be presented to the user …
- … however, it is possible that some iterations may produce artefacts that may not be evaluated wholistically …
- … e.g., an iteration may produce a prototype that just mimics the product (say its UI), but is not usable

Revisit – The Agile Philosophy

We had talked about the Agile Movement before as well

The key aspects that we highlighted were

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

Next, we will discuss a popular Software Development Process models, based on Agile philosophy

Scrum

A fairly popular Software Engineering Process Model based on Agile Philosophy is *Scrum*

- To be honest, Scrum is more a Project Management Process and can be easily adapted to other disciplines

A project managed through Scrum goes through *Sprints*, typically spanning 30 days

- A sprint is equivalent to an iteration in any iterative process

For each sprint the team focusses on a small, related set of *Backlogs*

- A backlog is a glorified term for processed requirements
- A significant part of it is a collection of *User Stories* (see your Homework)

A significant aspect of Scrum is that the teams be kept short and there is good communication

- The *Scrum Master* – a role that is similar to conventional roles like Team Lead or Project Manager …
- … makes sure that every member is on track and there are no hurdles appearing in their path
- To do so, a 15-minute daily call is scheduled, where each member reports the tasks done since the last call …
- … and what they intend to do next; these are called the *Scrum Meetings*

DevOps – The “buzzword”

You may have heard the term *DevOps* every now and then in the context of Software Projects

- It is kind of a buzzword – something in trend, and fashionable – though there are no clear definitions, yet !

The term was coined after probably joining two terms – “development” and “operations”

- Remember our discussion about how “developers” are not the only one who write code?
- There are significant code fragments that are part of modern-day projects that are not added by developers
- In terms of conventional wisdom, this code can be seen as part of the “project’s operations” …
- … i.e., the code (or configuration files) required for the seamless deployment and operation of the product
- The idea behind DevOps is to modify Software Processes and Teams such that …
- … the lines between development and operation activities are blurred

DevOps is essentially common sense, if you wish to have short deployment cycles

- Agile Teams, e.g., those using Scrums, had to remove the barricades between dev and ops anyway
- This was done so that the user gets a newer version of the product as soon as possible
- DevOps simply says make the same team/person responsible for deployment, who developed it !!