



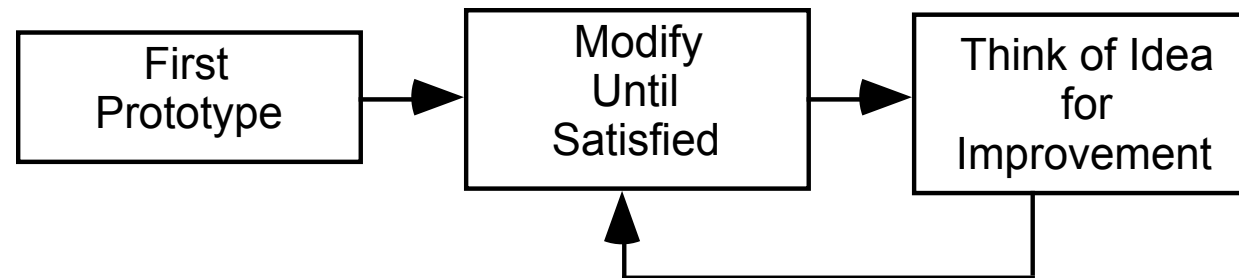
Software Process Models

Software Process Models

- Software process models are general approaches for organizing a project into activities.
 - Help the project manager and his or her team to decide:
 - What work should be done;
 - In what sequence to perform the work.
 - The models should be seen as *aids to thinking*, not rigid prescriptions of the way to do things.
 - Each project ends up with its own unique plan.



The opportunistic approach

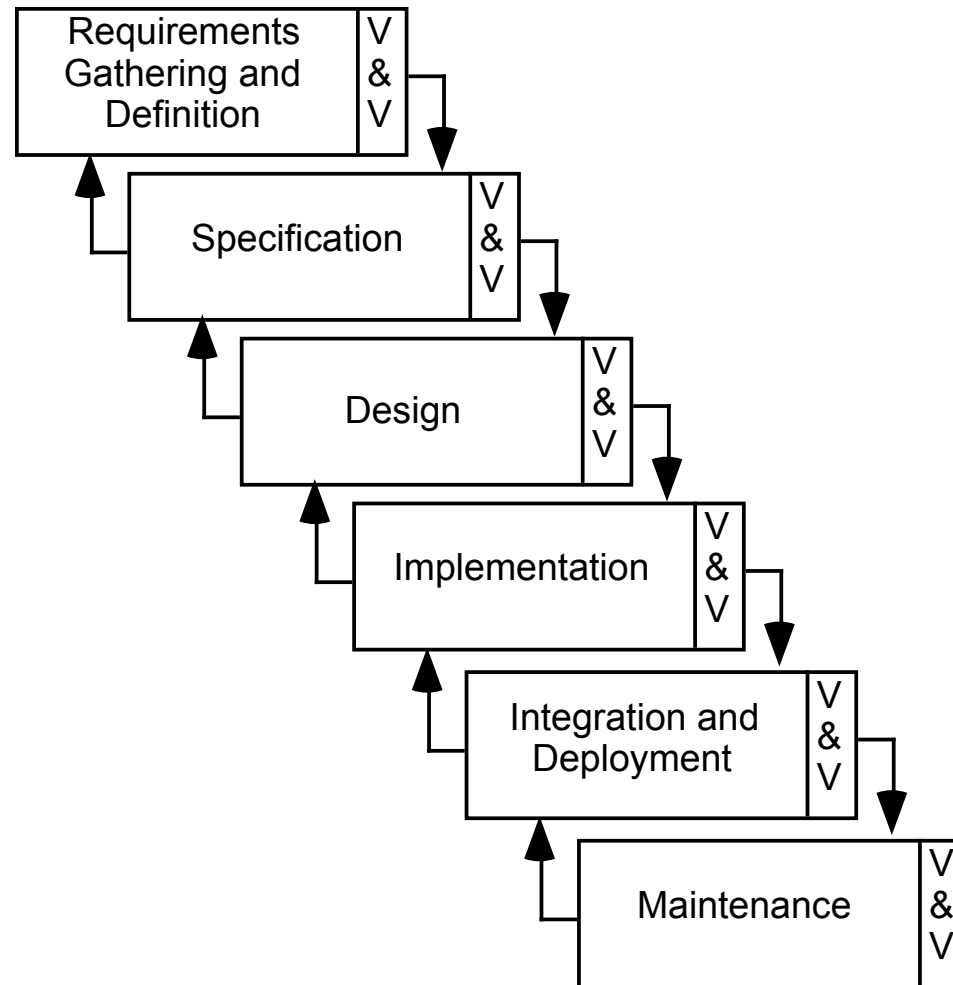


The opportunistic approach

- OK for small, informal projects
- Inappropriate for complex software
- Inappropriate for professional environments where on-time delivery and high quality are expected
 - Requirements and design not worked out before implementation
 - The design of software deteriorates faster if it is not well designed.
 - Since there are no plans, there is nothing to aim towards.
 - No explicit recognition of the need for systematic testing
 - The above problems make the cost of developing and maintaining software very high.



The waterfall model



The waterfall model

- The classic way of looking at S.E. that accounts for the importance of requirements, design and quality assurance.
 - The model suggests that software engineers should work in a series of stages.
 - Before completing each stage, they should perform quality assurance (verification and validation).
 - The waterfall model also recognizes, to a limited extent, that you sometimes have to step back to earlier stages.



Limitations of the waterfall model

- The model implies that you should attempt to complete a given stage before moving on to the next stage
 - Does not account for the fact that requirements constantly change.
 - It also means that customers can not use anything until the entire system is complete.
- The model makes no allowances for prototyping.
- It implies that you can get the requirements right by simply writing them down and reviewing them.
- The entire functionality is developed and then tested all together at the end. Major design problems may not be detected till very late.
- The model implies that once the product is finished, everything else is maintenance.



Incremental Development



Incremental Development

- Development occurs as a succession of releases with increasing functionality
- Customers provide feedback on each release
 - Used in deciding requirements and improvements for next release
- There is no “maintenance” phase – each version includes both problem fixes as well as new features
 - May also include “re-engineering” – changing the design and implementation of existing functionality, for easier maintainability
- The “phased release” and “evolutionary” models discussed in the text are incremental development approaches



Advantages of incremental development

- Customers get usable functionality earlier than with waterfall
- Getting early feedback improves likelihood of producing a product that satisfies customers
 - Reduces market risk: if customers hate the product, find out early before investing too much effort and money
- The quality of the final product is better
 - The core functionality is developed early and tested multiple times (during each release)
 - Only a relatively small amount of functionality added in each release: easier to get it right and test it thoroughly
 - Detect design problems early and get a chance to redesign



Choosing a process model

- Waterfall approach is OK for small projects, and when requirements & design can be done near-perfectly upfront
- Most complex real-world projects these days use an incremental approach with multiple phases
- There are other process models that bring in other ideas
 - The “spiral” model is risk-driven. Risks are analyzed after each iteration to figure out whether to continue and what to do in the next iteration to reduce risks.
 - “Concurrent engineering” brings in the idea of doing different project activities concurrently
- We can borrow ideas from different process models and create an approach that is suited to the characteristics of our particular project

