Topic 7

Managing the Software Process

Part 1 – Software Process Models

What is Project Management?

It encompasses all the activities needed to plan and execute a project

- Deciding what needs to be done
- Estimating costs
- Ensuring there are suitable people to undertake the project
- Defining responsibilities
- Scheduling
- Making arrangements for the work
- continued ...

What is Project Management?

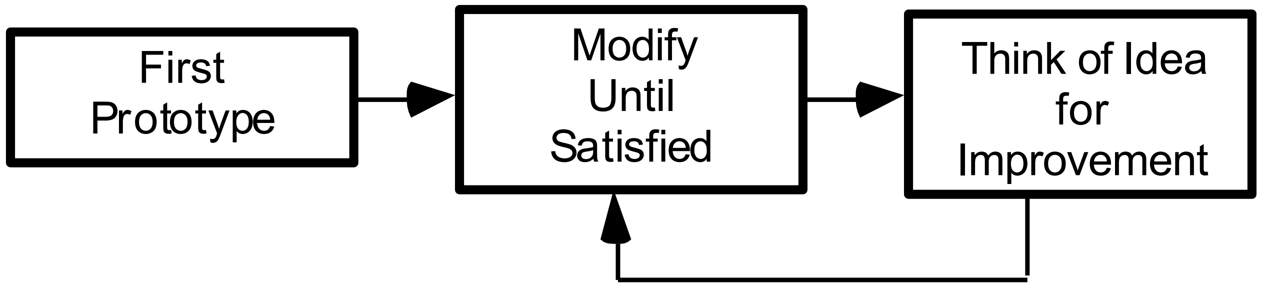
- ...continued
- Directing
- Being a technical leader
- Reviewing and approving decisions made by others
- Building morale and supporting staff
- Monitoring and controlling
- Co-ordinating the work with managers of other projects
- Reporting
- Continually striving to improve the process

Software Process Models

General approaches for organizing a project into activities.

- Help the project manager and/or team 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.

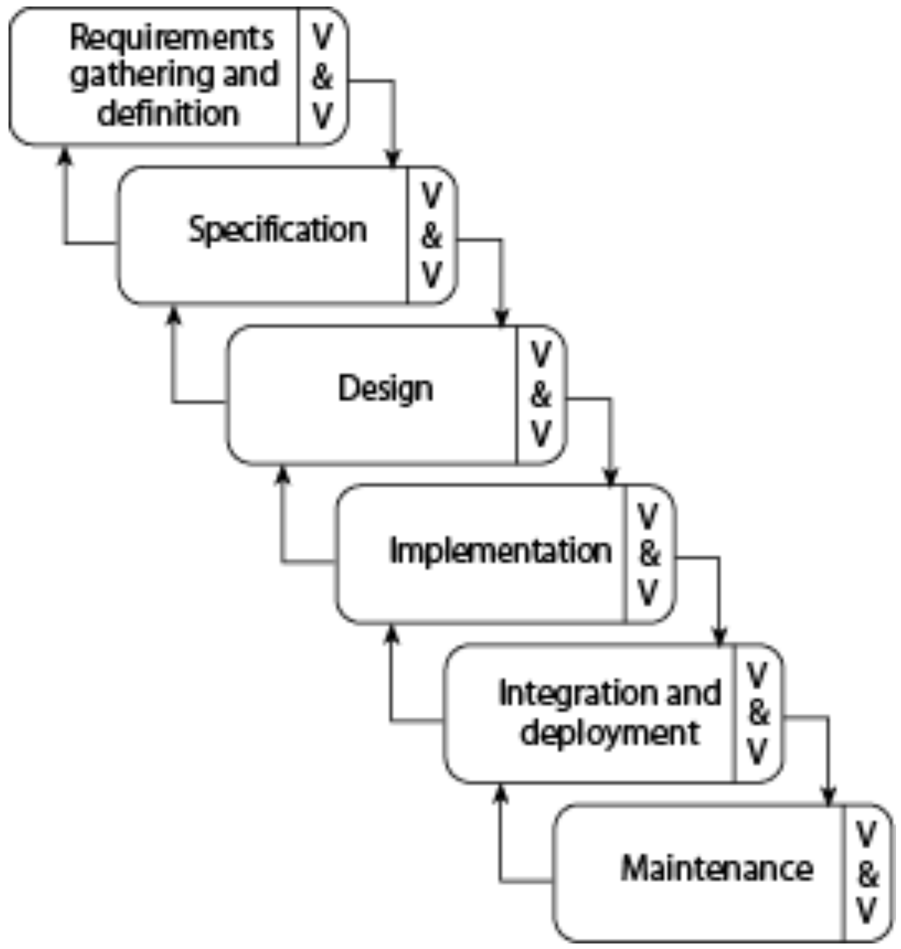
The optimistic approach



The optimistic approach

- … is what occurs when an organization does not follow good engineering practices.
  - Sometimes called “code and fix”
  - Does not acknowledge the importance of working out the requirements and the design before implementing a system.
  - Since there are no plans, there is nothing to aim towards.
  - There is no explicit recognition of the need for systematic testing and other forms of quality assurance.
  - The above problems make the cost of developing and maintaining software very high.
  - There may not be time to develop until satisfied

The Waterfall model



The Waterfall model

- The classic way of looking at SE 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.
  - QUESTION: What is wrong with getting all the requirements completed upfront?

The Waterfall model

- Circa 1970's
- Document-driven approach
  - Originally involved many documents in the requirements, design and testing stages
- Limited interaction with users/ customers after requirements phase

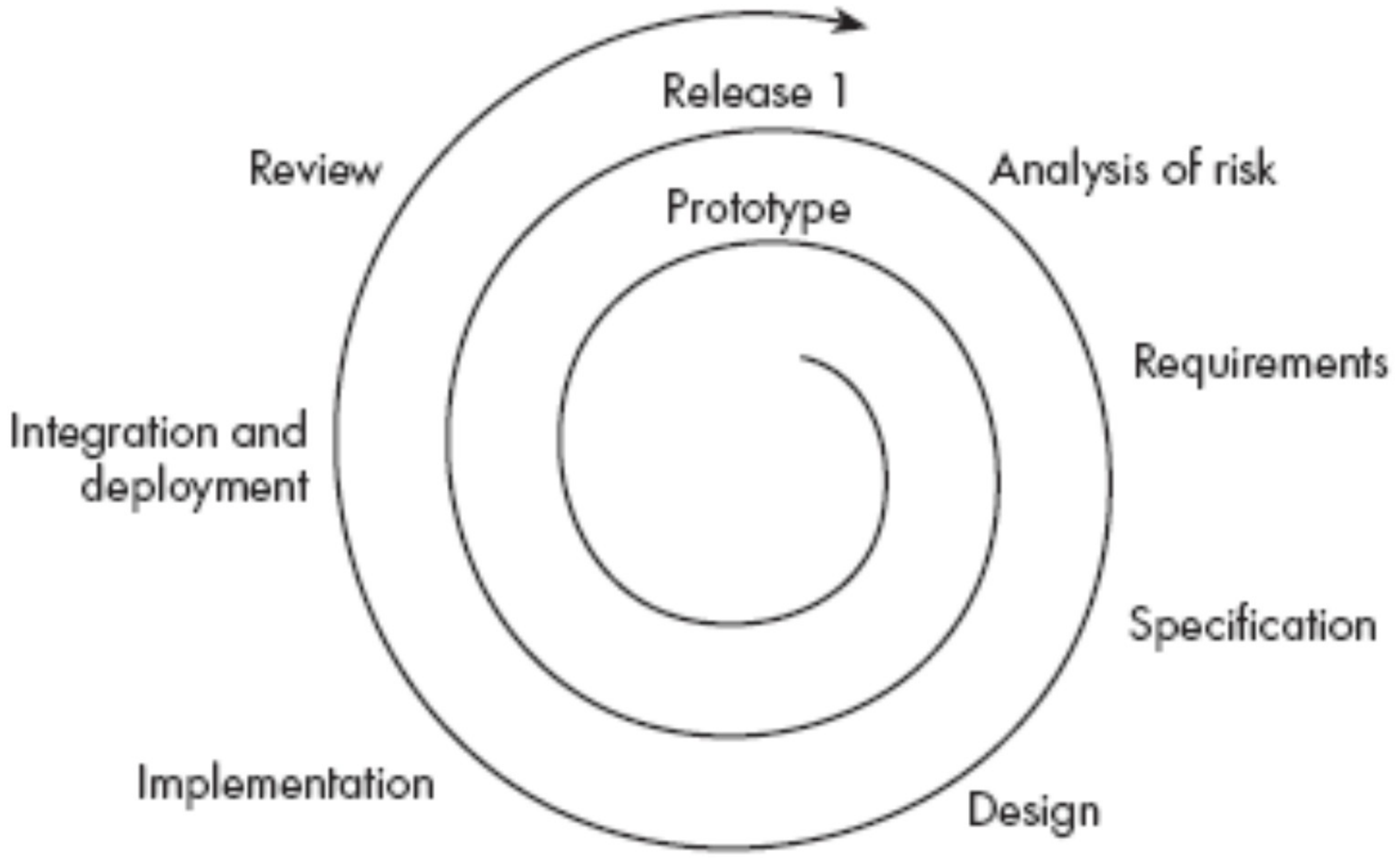
Limitations of the waterfall model

- Implies that you should complete a given stage before moving on
  - Does not account for the fact that requirements may (constantly) change.
  - It also means that customers can not use anything until the entire system is complete.

Limitations of the waterfall model

- 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 model implies that once the product is finished, everything else is maintenance.

Spiral Model



Spiral Model

- Explicitly embraces prototyping and an iterative approach to software development.
  - Start by developing a small prototype.
  - Followed by a mini-waterfall process, primarily to gather requirements.
  - Then, the first prototype is reviewed.
  - In subsequent loops, the project team performs further requirements, design, implementation and review.
  - The first thing to do before embarking on each new loop is risk analysis.
  - Maintenance is simply a type of on-going development

Agile Software Development

- Development in small iterations
- Well suited for projects that involve uncertain, changing requirements or other high risks
- Values face-to-face conversation
- User and business involvement at all stages
  - vs. Spiral $\twoheadrightarrow$ users/business only at requirements and review stages
- Aims to maximize customer profit over the lifetime of the project
  - Earlier release
  - Sustained improvement and maintenance

1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Self-organizing teams
12. Regular adaptation to changing circumstance

- Agile Manifesto

AGILE Method

DILBERT – By Scott Adams http://www.unitedmedia.com/comics/dilbert/
e-mail: scottadams@aol.com
WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
scottadams@aol.com www.dilbert.com
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
11-26-07 ©2007 Scott Adams, Inc./Dist. by UFS, Inc.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.

Agile Development Models

"based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change."

Agile Process Models

- eXtreme Programming (XP)
- Scrum

Agile Process Practices

- Pair programming
- Test-driven Development

eXtreme Programming (XP)

- All stake holders work closely together
- User stories instead of requirements document
- Small and frequent releases: 1 to 3 weeks
- Test-first development
- A large amount of refactoring is encouraged
- Pair programming is recommended

Pair Programming

- Two people at one machine
  - One person at machine
  - The other watches and reviews
  - Switch Frequently
- Promotes higher quality code

Scrum Agile

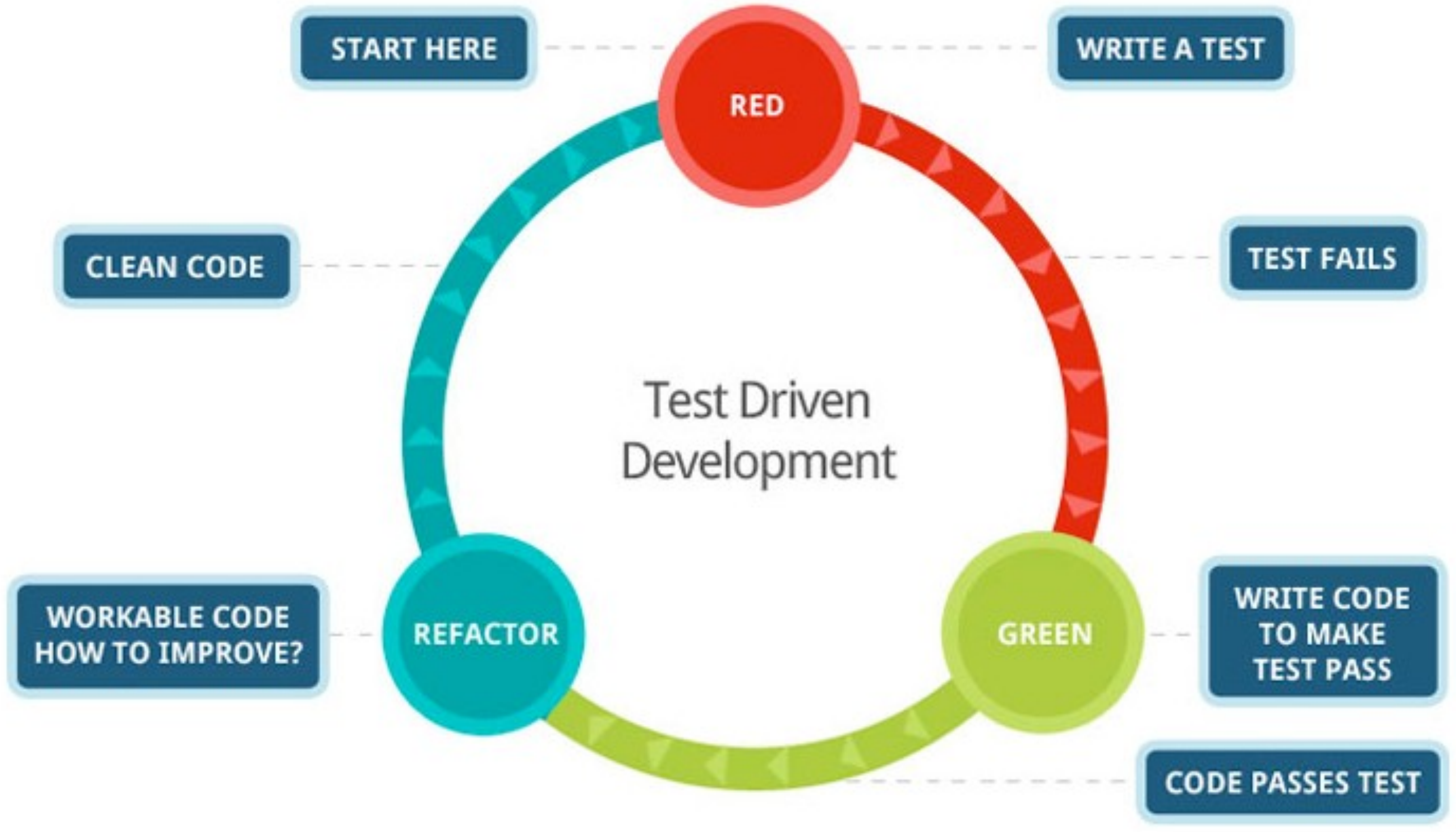
- Scrum Team:
  - Product owner – main manager
  - Scrum master – manages sprints & dev team
  - Dev team – developers
- Sprint
  - A single development iteration (typically short)
  - Starts with sprint planning meeting
  - Daily scrum meetings (face to face)
  - Ends with sprint retrospective

Test Driven Development

- Used by several Agile models
- Tests are written before code

- Three laws:
  - You are not allowed to write any production code until you have first written a failing test
  - You are not allowed to write more of a unit test than is sufficient to fail - not compiling is failing
  - You are not allowed to write more production code than is sufficient to pass the currently failing test

Test Driven Development



Test Driven Development

- Tests and code grow together
- Code is always verified
  - Can see if adding code has broken things that were working (regression testing)
- Refactoring happens often
  - clean up any messy implementation
  - can immediately test
- Built-in documentation
  - tests help define expectations of units/classes

Criticisms of Agile

- Heavy reliance on team communication and cohesiveness
- More suitable for smaller projects or subteams
- Requires frequent access to customer
- Minimal documentation generated

"Good code is its own best documentation"