Chapter 3
Software Process Models

Program life cycle

1. Conception
2. Requirements gathering/exploration/modeling
3. Design
4. Coding and debugging
5. Testing
6. Release
7. Maintenance/software evolution
8. Retirement

Project Management Models

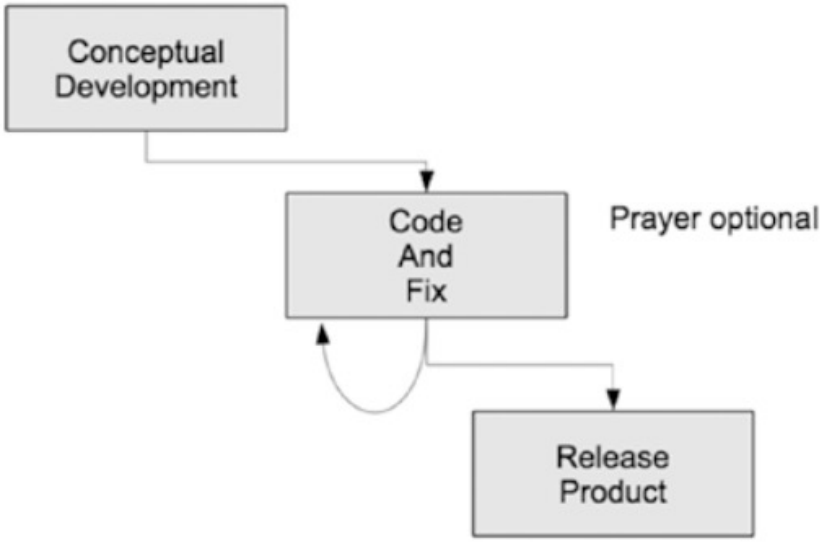
- **plan-driven models**
  - models have more clearly defined phases, and more requirements for sign-off on completion of a phase
- **agile development models**
  - the agile models are inherently incremental and make the assumption
    that small, frequent releases produce a more robust product than larger, less frequent ones.

The Four Variables

- The four variables of software development projects are as follows:
  - ***Cost*** is probably the most constrained
  - ***Time*** is your delivery schedule and is unfortunately many times imposed on you from the outside.
  - ***Quality*** is the number and severity of defects you’re willing to release with.
  - ***Features*** (also called *scope*) are what the product actually does.

Code and Fix Model

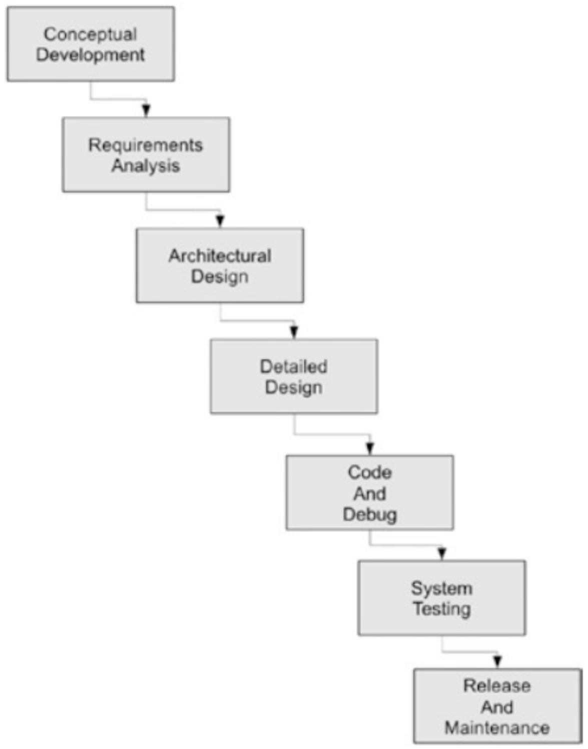
- is often used in lieu of actual project management.



The code and fix process model

Cruising over the Waterfall

- First, it generally requires that you finish phase N before you continue on to phase N+1



The waterfall process model

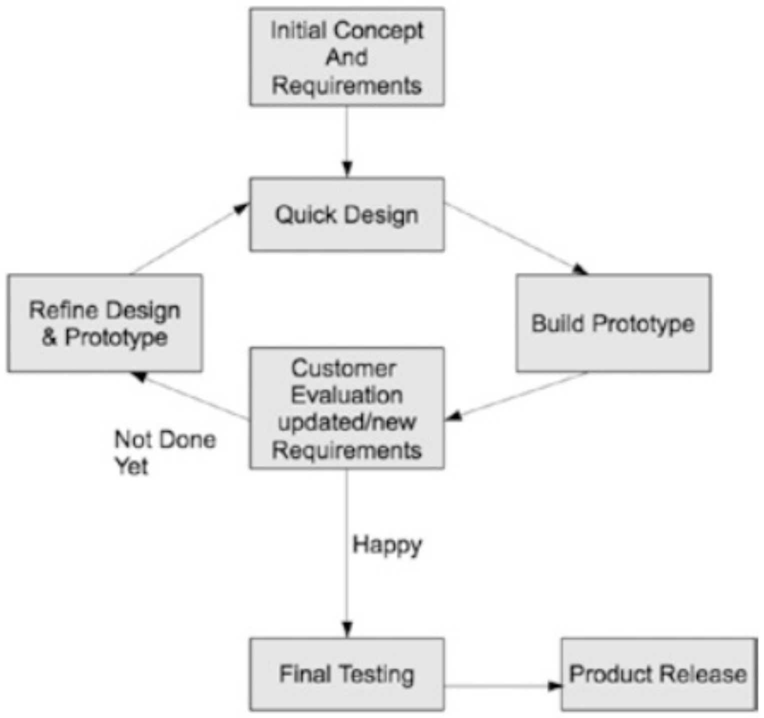
Iterative Models

The best practice is to iterate and deliver incrementally, treating each iteration as a closed- end "mini-project," including complete requirements, design, coding, integration, testing, and internal delivery. On the iteration deadline, deliver the (fully-tested, fully-integrated) system thus far to internal stakeholders. Solicit their feedback on that work, and fold that feedback into the plan for the next iteration.

- (From "How Agile Projects Succeed")

Evolving the Iterative Model

- A traditional way of implementing the iterative model is known as evolutionary prototyping.


Initial Concept And Requirements
Quick Design
Build Prototype
Customer Evaluation updated/new Requirements
Refine Design & Prototype
Not Done Yet
Happy
Final Testing
Product Release

Evolutionary prototyping process model

Risk: The Problem with Plan-Driven Models

- **Risk** is the most basic problem in software.
- *Risk* manifests itself in many ways: schedule slips, project cancelation, increased defect rates, misunderstanding of the business problem, false feature richness (you’ve added features the customer really doesn’t want or need), and staff turnover.

Agile Methodologies

- Starting in the mid 1990s,
- this new process model was lightweight.
- It required less documentation and fewer process controls.

Agile Values and Principles

- The values are as follows
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan

Agile Values and Principles

- The principles run as follows:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Agile Values and Principles

7. Working software is the primary way to measure progress.
8. Agile processes promote sustainable development. The sponsors, developers,
9. and users should be able to maintain a constant pace indefinitely.
10. Continuous attention to technical excellence and good design enhances agility.
11. Simplicity—the art of maximizing the amount of work not done—is essential.
12. The best architectures, requirements, and designs emerge from self-organizing
13. teams.
14. At regular intervals, the team reflects on how to become more effective and then tunes and adjusts its behavior accordingly.

eXtreme Programming (XP)

- Kent Beck and Ward Cunningham created XP around 1995.
- XP is a “lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software.”11

XP Overview

- XP relies on the following four fundamental ideas:
  - *Heavy customer involvement*: XP requires that a customer representative be part of the development team and be on site at all times.
  - *Continuous unit testing (also known as test-driven development, or TDD)*: XP calls for developers to write the unit tests for any new features before any of the code is written.
  - *Pair programming*: XP requires that pairs of developers write all code. In a nutshell, pair programming requires two programmers—a driver and a navigator—who share a single computer.
  - *short iteration cycles and frequent releases*: XP typically uses release cycles in the range of just a few weeks or months, and each release is composed of several iterations, each on the order of three to five weeks.

The Four Basic Activities

- XP describes four activities that are the bedrock of the discipline:
  - *Designing*: Design while you code.
  - *Coding*: The code is where the knowledge of the system resides, so it’s your main activity.
  - *Testing*: The tests tell you when you’re done coding.
  - *Listening*: To your partner and to the customer.

Implementing XP: The 12 Practices

- *The planning game*: Develop the scope of the next release by combining business priorities and technical estimates.
- *Small releases*: Put a simple system into production quickly and then release new versions on a very short cycle.
- *Metaphor*: “A simple shared story of how the whole system works.” The metaphor replaces your architecture.
- *Simple design*: Keep the design as simple as you can each day and re-design often to keep it simple.

Implementing XP: The 12 Practices

- *Testing*: Programmers constantly write unit tests. Tests must all pass before integration.
- *Refactoring*: Restructure the system "without changing its behavior" to make it simpler—removing redundancy, eliminating unnecessary layers of code, or adding flexibility.
- *Pair programming*: Two programmers at one machine must write all production code in an XP project.
- *Collective ownership*: The team owns everything, implying that anyone can change anything at any time.

Implementing XP: The 12 Practices

- *Continuous integration*: Integrate and build every time a task is finished, possibly several times a day (as long as the tests all pass).
- *40-hour week*: Work a regular 40-hour week. Never work a second week in a row with overtime.
- *On-site customer*: A customer is part of the team, is on-site, writes and executes functional tests, and helps clarify requirements.
- *Coding standards*: The team has them, follows them, and uses them to improve communication.

Scrum

- The second agile methodology.
- S*crum* is a means of restarting play after a rules infraction.

Scrum Roles

- the requirements normally take the form of *user stories*—features that can be summarized by sentences like
- “As a <type of user>,
- I want to <do or create something>,
- so that <some value is created>.

Scrum Roles

- Scrum defines three roles in a development project.
  - ***product owner***, the person who generates the requirements for the product and prioritizes them.
  - ***Scrum master*** whose job it is to manage the backlogs, run the daily Scrum meetings, coach the team, and protect the team from outside influences during the sprint.
  - the ***development team*** itself is self-organizing; the members of the Scrum team decide among themselves who will work on what user stories and tasks, assume collective ownership of the project, and decide on the development process they'll use during the sprint.

The Sprint

- Scrum is characterized by the *sprint*, an iteration of between one and four weeks.

Scrum Artifacts

- Scrum requirements are encapsulated in two backlogs.
  - The ***product backlog*** is the prioritized list of all the requirements for the project; the product owner creates it.
  - The ***sprint backlog*** is the prioritized list of user stories for the current sprint.

Sprint Flow

- Before the first sprint starts, Scrum has an initial planning phase that creates the product list of the initial requirements, decides on an architecture for implementing the requirements, divides the user stories
  into prioritized groups for the sprints, and breaks the first set of user stories into tasks to be estimated and assigned.

Lean Software Development

- Lean software development comes from the just-in-time manufacturing processes (also known as the Toyota Production System, among other names) that were introduced in Japan in the 1970s and then made their way around the world in the 1980s and 1990s, encouraged by the publication in 1990 of *The Machine That Changed The World* by Womack, et. al.1

seven key principles for software development:

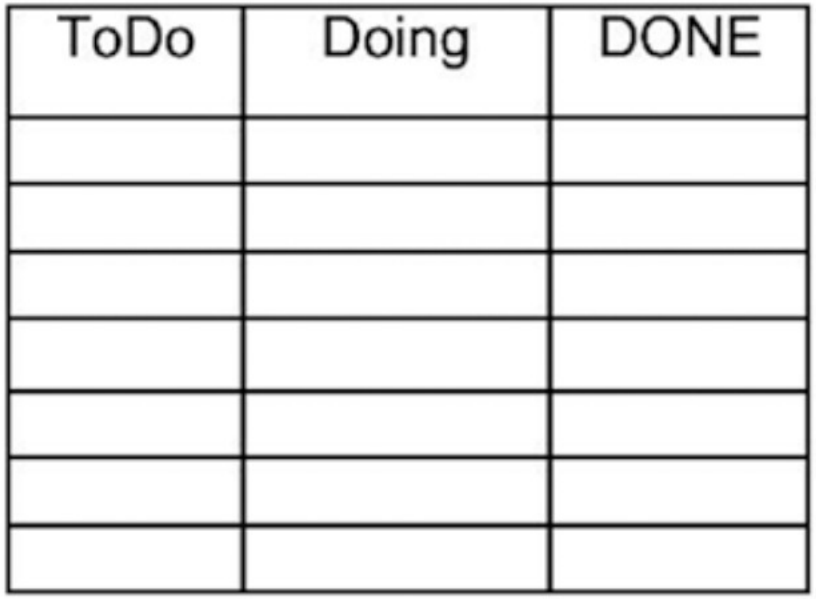
1. **Eliminate Waste.** eliminate anything that doesn't add value to the product.
2. **Build Quality In.** Quality issues are the bane of every software developer.
3. **Create Knowledge.** The team must learn new things constantly
4. **Defer Commitment.** *put off decisions* (particularly irreversible ones) as long as you can and only make them when you must.
5. **Deliver Fast.** "First to market"
6. **Respect People.** all about building strong, productive teams.
7. **Optimize the Whole.** keep the entire product picture in sight as you develop.

Kanban

- The Kanban method is a practice derived from lean manufacturing (the Toyota Production System) and other change-management systems; it draws most of its original ideas from the just-in-time manufacturing processes.
- Kanban uses three ideas to influence a development process:
  - ***Work-in-progress*** is the total number of tasks the team is currently working on
  - ***Flow*** is the passage of tasks from one state to another on the way to completion.
  - ***Lead time*** is the amount of time it takes a task to move from its initial state to its completed state.

The Kanban board, WIP, and Flow

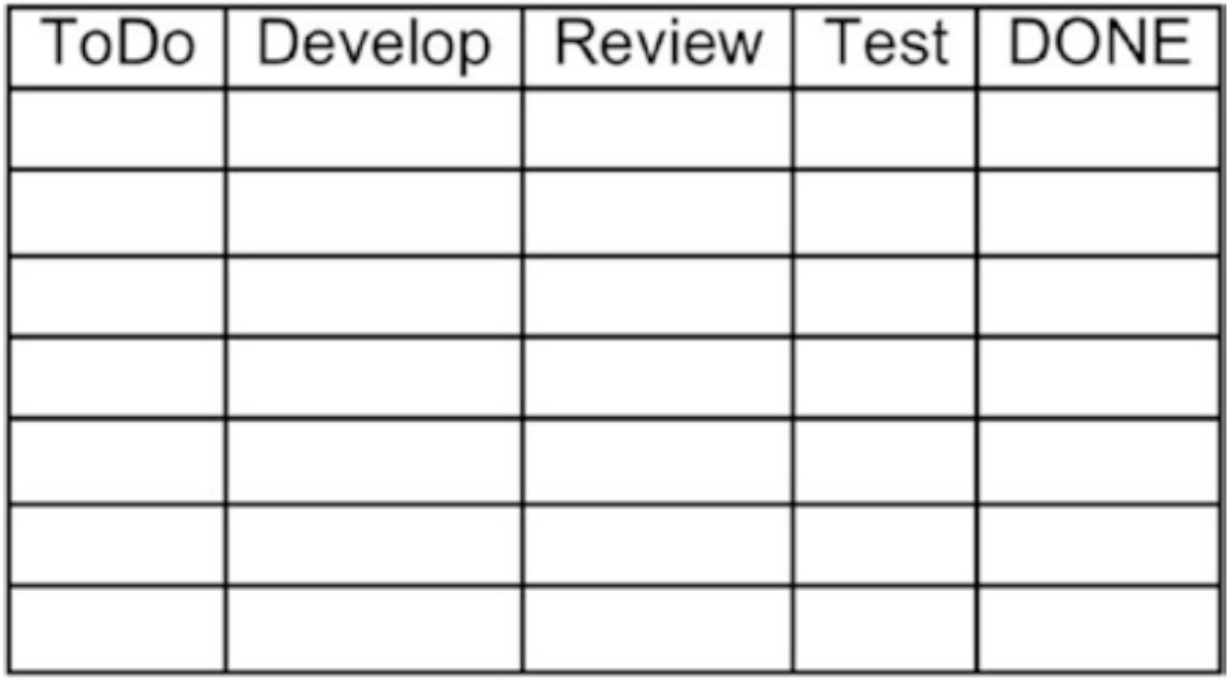
- These boards are often physical white boards that occupy one wall of a common space for the team.



ToDo	Doing	DONE

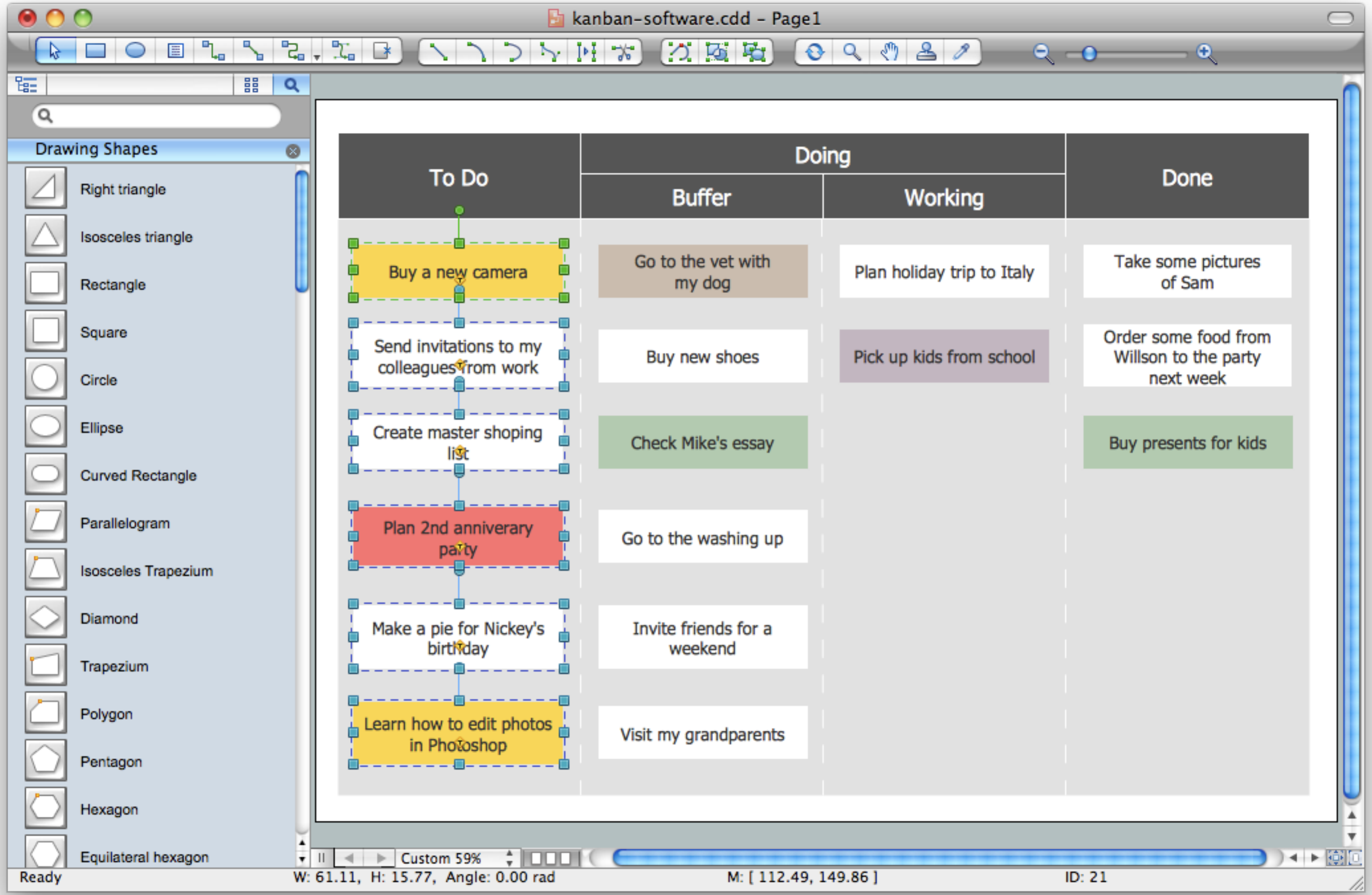
A generic task/Kanban board

A task/Kanban board applied to a software development project



ToDo	Develop	Review	Test	DONE

Kanban Software



To Do	Doing		Done
	Buffer	Working	
Buy a new camera	Go to the vet with my dog	Plan holiday trip to Italy	Take some pictures of Sam
Send invitations to my colleagues from work	Buy new shoes	Pick up kids from school	Order some food from Willson to the party next week
Create master shoping list	Check Mike's essay		Buy presents for kids
Plan 2nd anniverary party	Go to the washing up		
Make a pie for Nickey's birthday	Invite friends for a weekend		
Learn how to edit photos in Photoshop	Visit my grandparents		

Ready W: 61.11, H: 15.77, Angle: 0.00 rad M: [112.49, 149.86] ID: 21

Summary

- As can be seen from the methodologies described in this chapter, iteration is the key, whether you’re using an evolutionary, plan-driven process or an agile development one. Recognize that the best way to build a complex piece of software is incrementally. Learn that designing, writing, testing, and delivering incrementally better code is your first step to writing great software.

End