Overview of Software Process Models

Hui Chen [a]

[a]CUNY Brooklyn College, Brooklyn, NY, USA

February 1, 2022

Outline

Outline

Process Model

Process model describes

1. (what) what tasks need to be performed in
2. (when) what sequence under
3. (whom) what conditions by
4. whom to

achieve the "desired results."

Recall Software Development Effort Examples

Size (LOC)	Example	
10^2	Class Exercise	Programming
10^3	Small Project	
10^4	Term Project	
10^5	Business Application	Software Engineering
10^6	Word Processor	
10^7	Operating System	

Do we need a process model?

There is a need to scale up with complexity. For instance, if we are to develop a business application, we must consider

- How do we ensure we know what the customers really want when they asked for a feature?
- How do we split the work among a team of people?
- ...

Process models provide "guidance" for a systematic *coordination and control* of

- the *tasks* and
- the *personnel* who perform the tasks

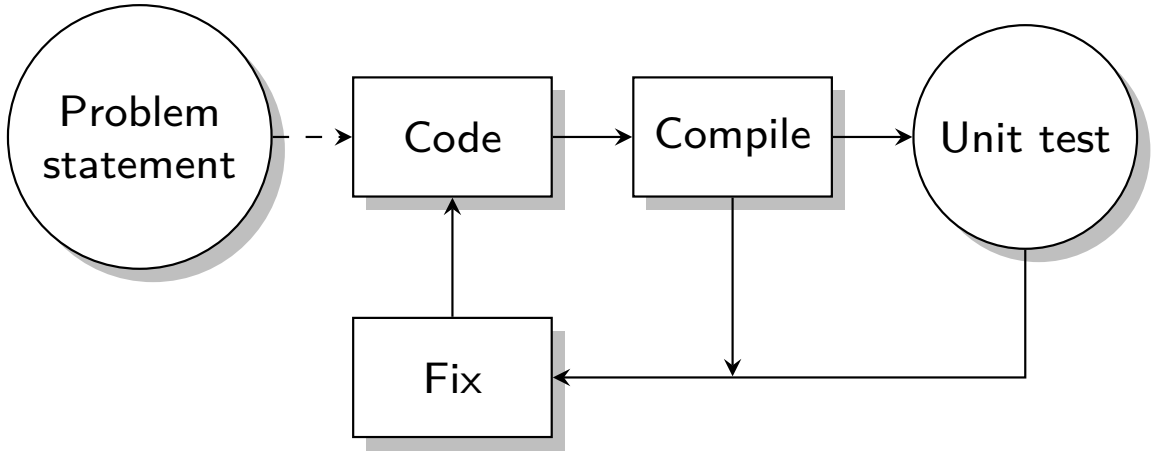
Do we need a process model?

What if the project requires just one person or at most two people?

Why?

Why not ?

A "Simple and Familar" Process



- Most people perform and follow this simple process, but unfortunately some skip unit testing or debugging.
- Also, some proceed without thoroughly considering & understanding the "problem statement" (which is the requirement).

Extending the "Simple" Process

As projects get larger and more complex:

- Need to clarify and stabilize the requirements
- Need to test more functionalities
- Need to design more carefully
- Need to use more existing software and tools
- Need more people to be involved

which results in more tasks and more people and more coordination and control $\rightarrow$ there is a need to describe the process.

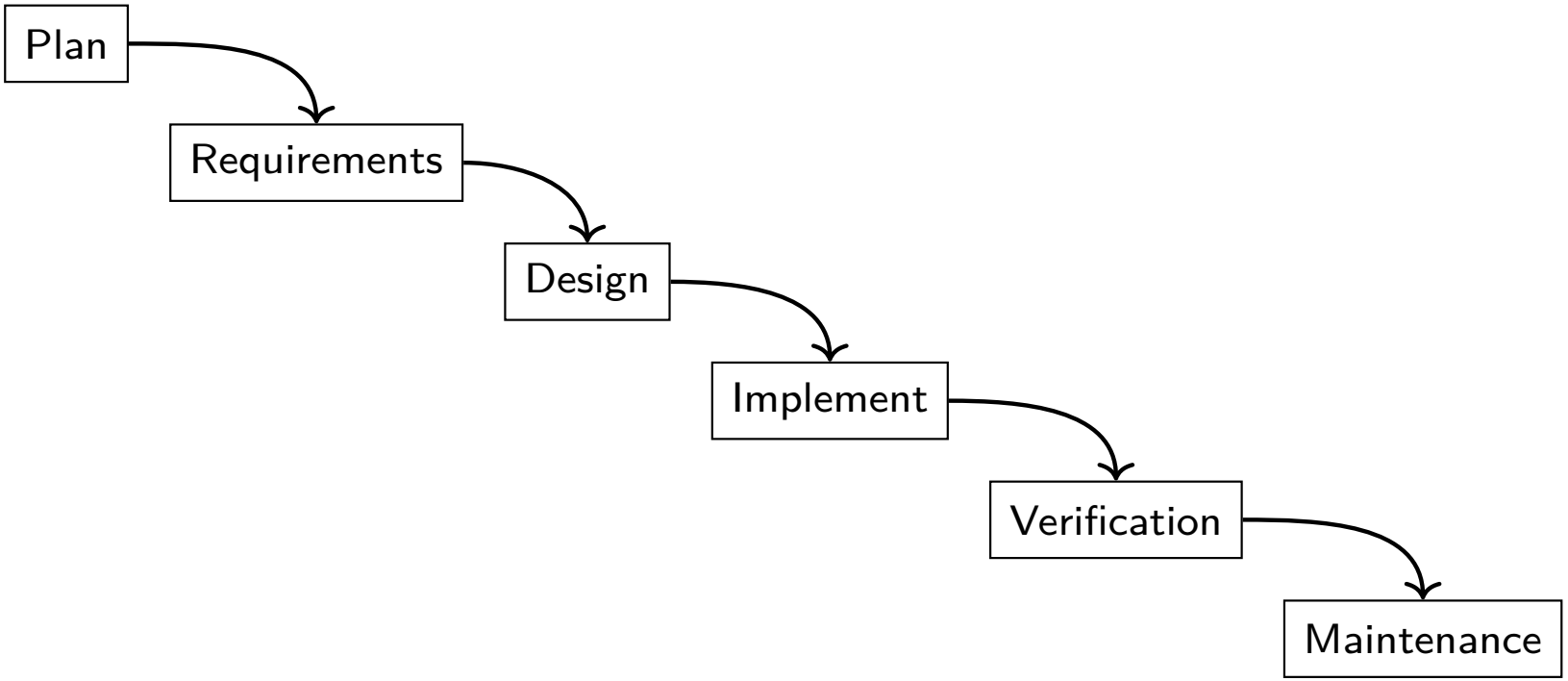
Outline

Software Development Process Models

The recognition of the need for formal processes was initially driven by failures in developing large complex software.

- Waterfall model: earliest process and coping with no process
- Incremental model: coping with decomposing the large systems
- Spiral model : coping with risk management
- Rational Unified Process model: coping with multiple development and management issues

Waterfall Model



Waterfall Model

1. Requirements must be specified.
2. Four main tasks must be completed in sequence: requirements, design, code, and test, followed by integration.
3. Output of one stage feeds into the next stage in sequence, and thus is easily tracked ("controlled") by management.

Waterfall Model

"And the users exclaimed with a laugh and a taunt: 'It's just what we asked for, but not what we want."' — Anonymous

The Good

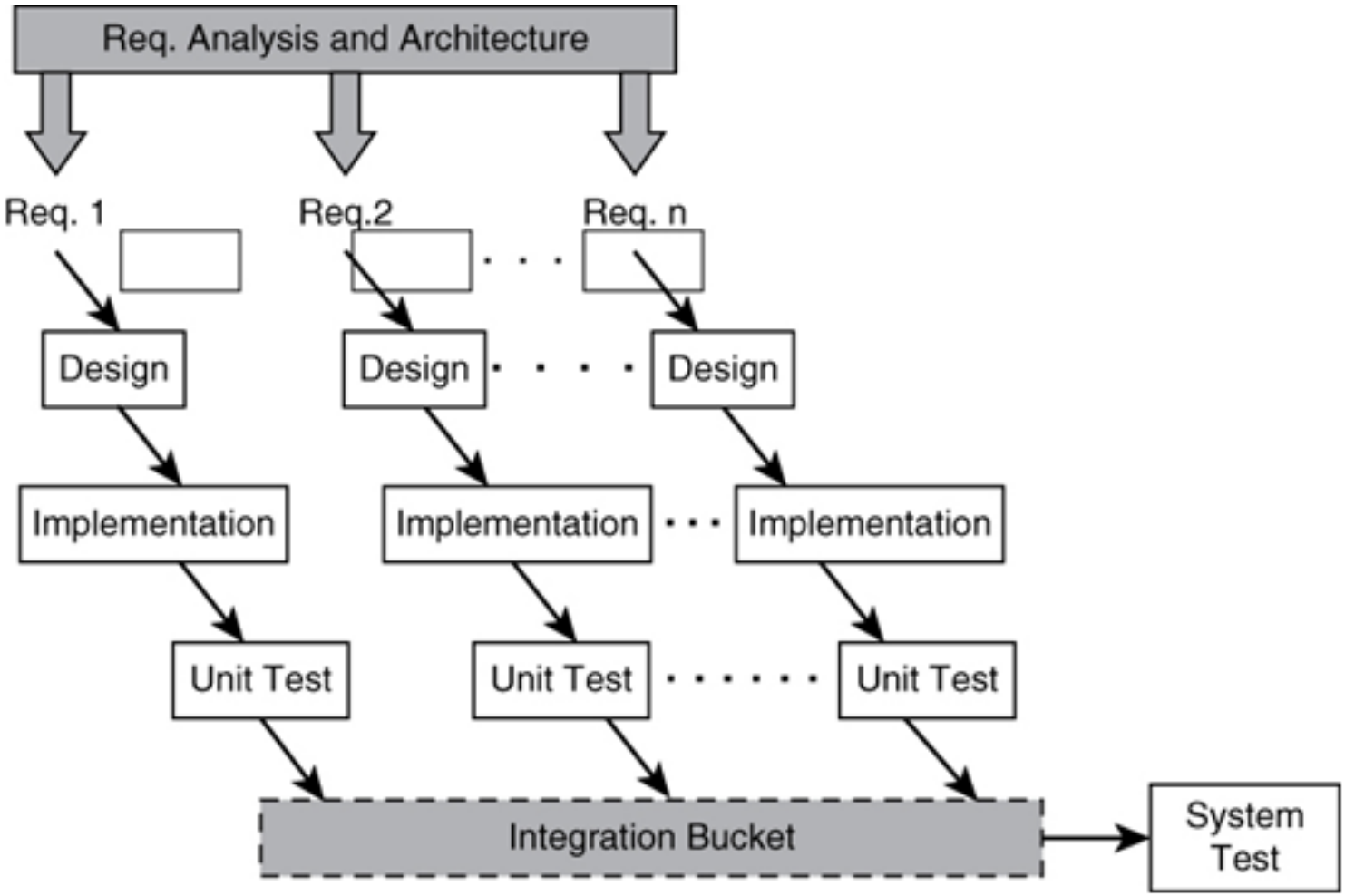
- Simple!!!
- Plenty of documentation, which is good (allows for management of project)
- Still in use since the 70s

The Bad

- Testing is towards the end of the model, and it may expose fundamental problems, requiring rework Customer is not involved
- ...

Continuous Integration Model

Incremental Model (A): "Continuous Integration"



Continuous Integration

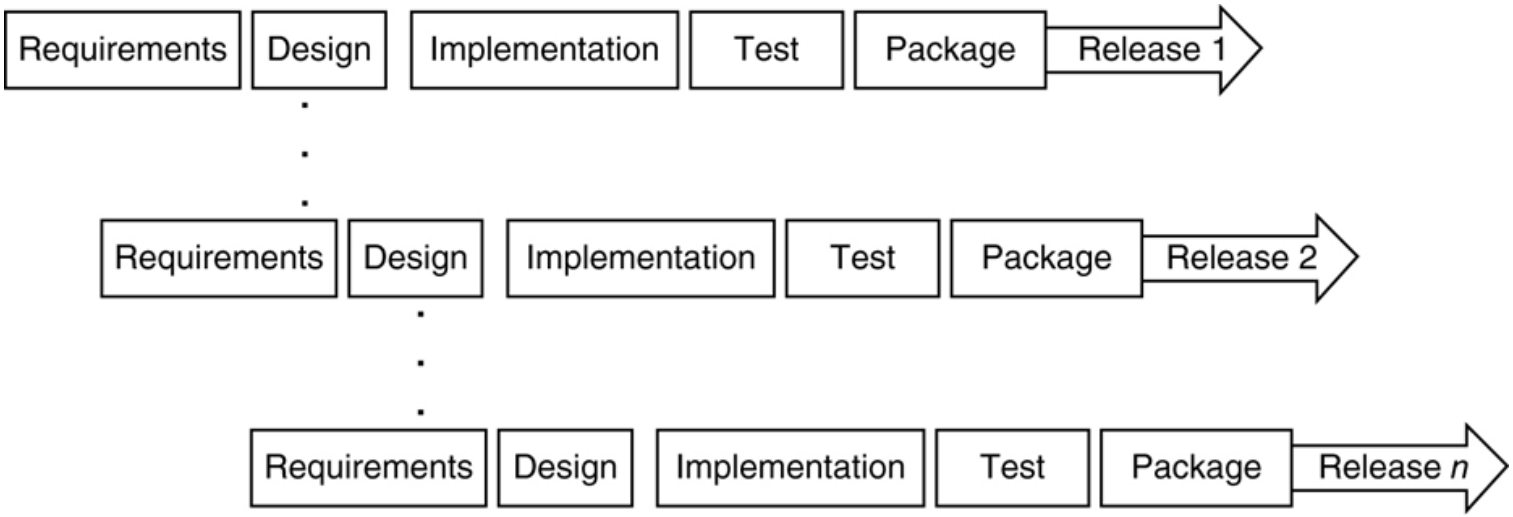
Incremental Model (A): "Continuous Integration"

1. Each "major requirement/item" is developed separately through the same sequence of: requirement, design, code, and unit test.
2. As the developed pieces are completed, they are continuously merged and integrated into a common bucket for integrated system test.

Multiple Releases

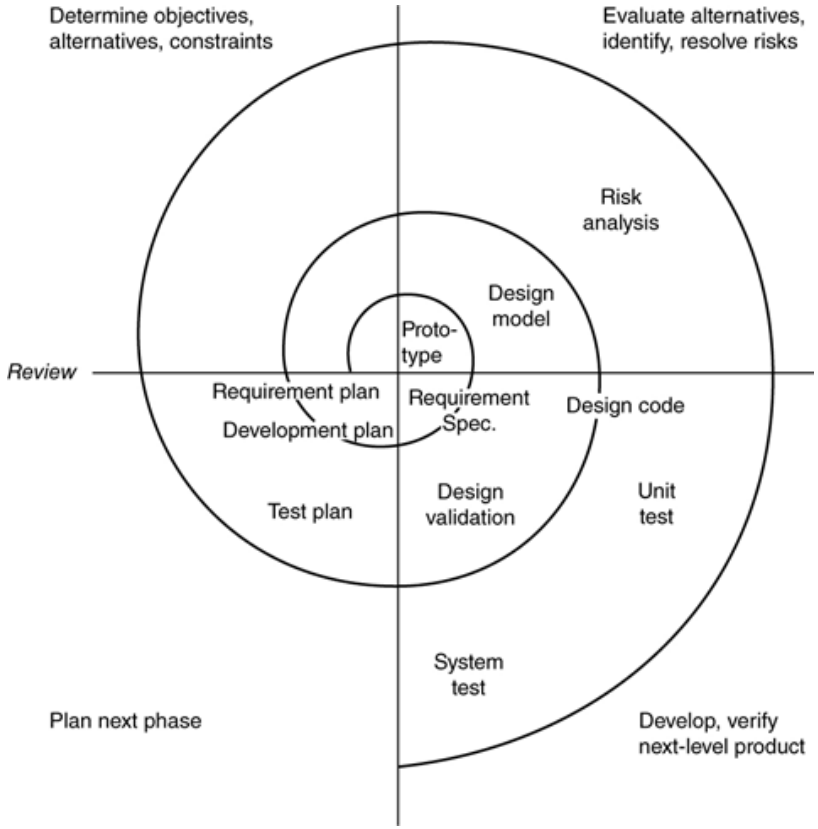
Incremental Model (B): "Multiple Releases"

- Each small set of requirements is developed, packaged, and released in a multiple release fashion.



Spiral Model

Software development activities are cycled through four phases.



Spiral Model

The Good

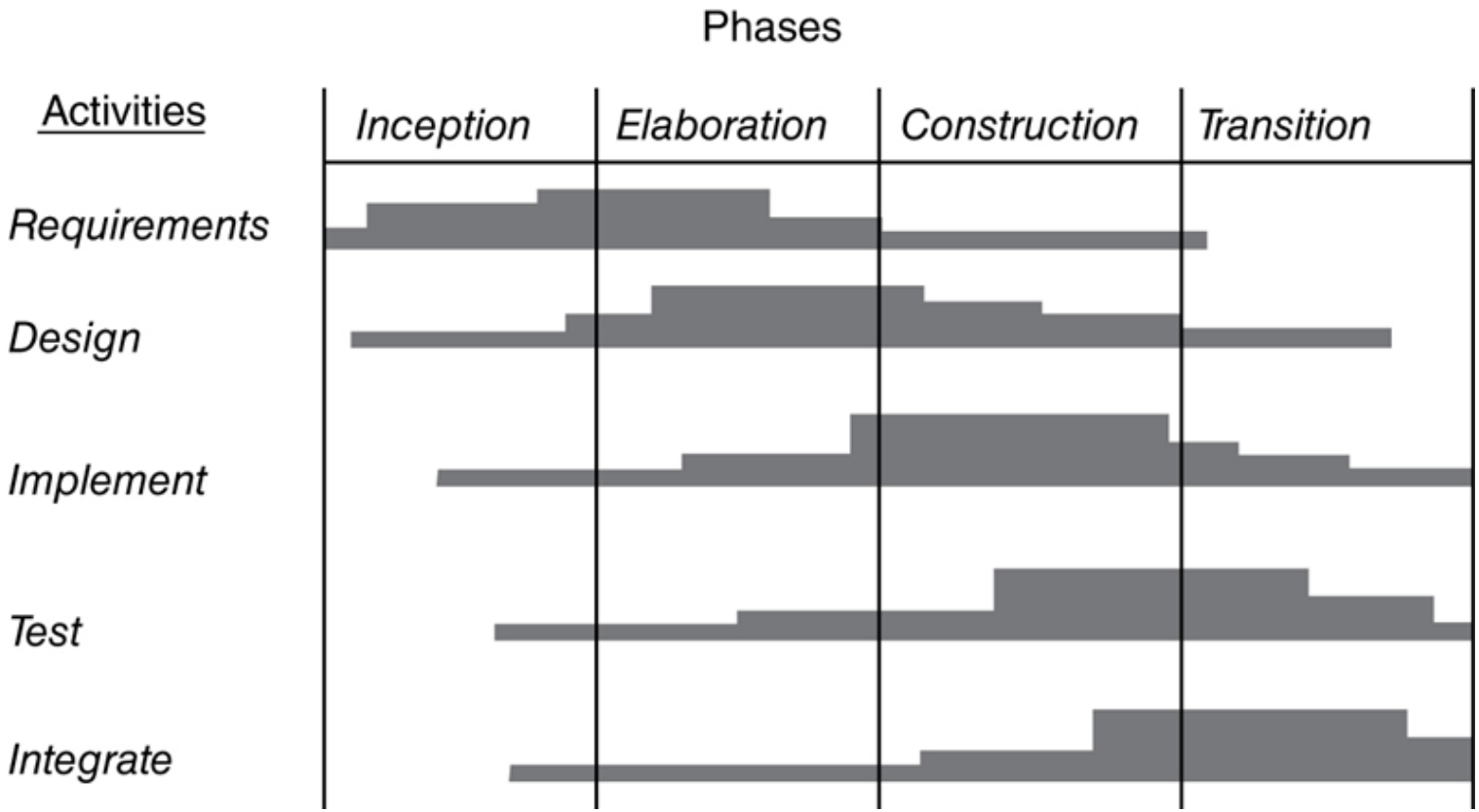
- Iterations involve the customer before the product is completed
- Reduces chances of misunderstandings
- Risk management part of lifecycle
- Project monitoring easy
- Schedule & cost more realistic over time

The Bad

- Iterations 6 to 24 months long
- Time for customers to change their minds!
- Lots of documentation per iteration
- Lots of rules to follow, hard for whole project
- Cost of process is high
- Hard to meet budget and schedule targets

Rational Unified Process (RUP)

Every software development activity is "addressed" in the four phases of inception, elaboration, construction, and transition.



Entry and Exit Criteria

In order for a process model to be more than just a "guideline," it must include a list of conditions or requirements that define the:

- Entry criteria prior to performing an activity in a process.
- Exit criteria before an activity in the process is deemed completed.

Outline

Problems with "Traditional" Processes

- Focused on and oriented towards "large projects" and lengthy development time (years) (lengthy development time)
- Inability to cope with changes in requirements and technology fast enough ("formal" change management)
- Assumes requirements are completely understood at beginning of project
- Starting to rely on "non-sustainable" heroic and lengthy development effort by the developers (hard to maintain "constantly high" productivity.)
- Complex set of activities (needed process experts).
- Waste or duplication of effort, especially in documentation (formal documentation needed for long and large project communications.)

Agile Methodologies

A family of software development methodologies:

- "Short" releases and multiple iterations
- Incremental design/development
- User involvement (especially for in-house)
- Minimal documentation
- Informal communications
- Assumes changes

The Agile Manifesto

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

Individuals and interactions over processes & 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."

Example Agile Methodologies

- Extreme Programming (XP) – the first by Beck (1990s)
- Crystal Clear/Orange – by Alister Cockburn
- Scrum – currently popular, not really part of Agile – partially agile)
- RUP (rational unified process)
- Microsoft Solutions Framework (tool/process)

XP's Core Values

- Communication (between team and with customers)
- Simplicity (in design and code)
- Feedback (at many levels)
- Courage (to make and implement difficult decision)

XP's Fundamental Principles

- Rapid feedback
- Simplicity
- Incremental change
- Embrace change
- Quality work

Directly from the "core" values

XP's Lesser/Other Principles

- Ongoing learning
- Small initial investment
- Playing to win
- Concrete experiments
- Open, honest communications
- Working with people's instincts
- Accepting responsibility
- Local adaptation
- Traveling light
- Honest measurement

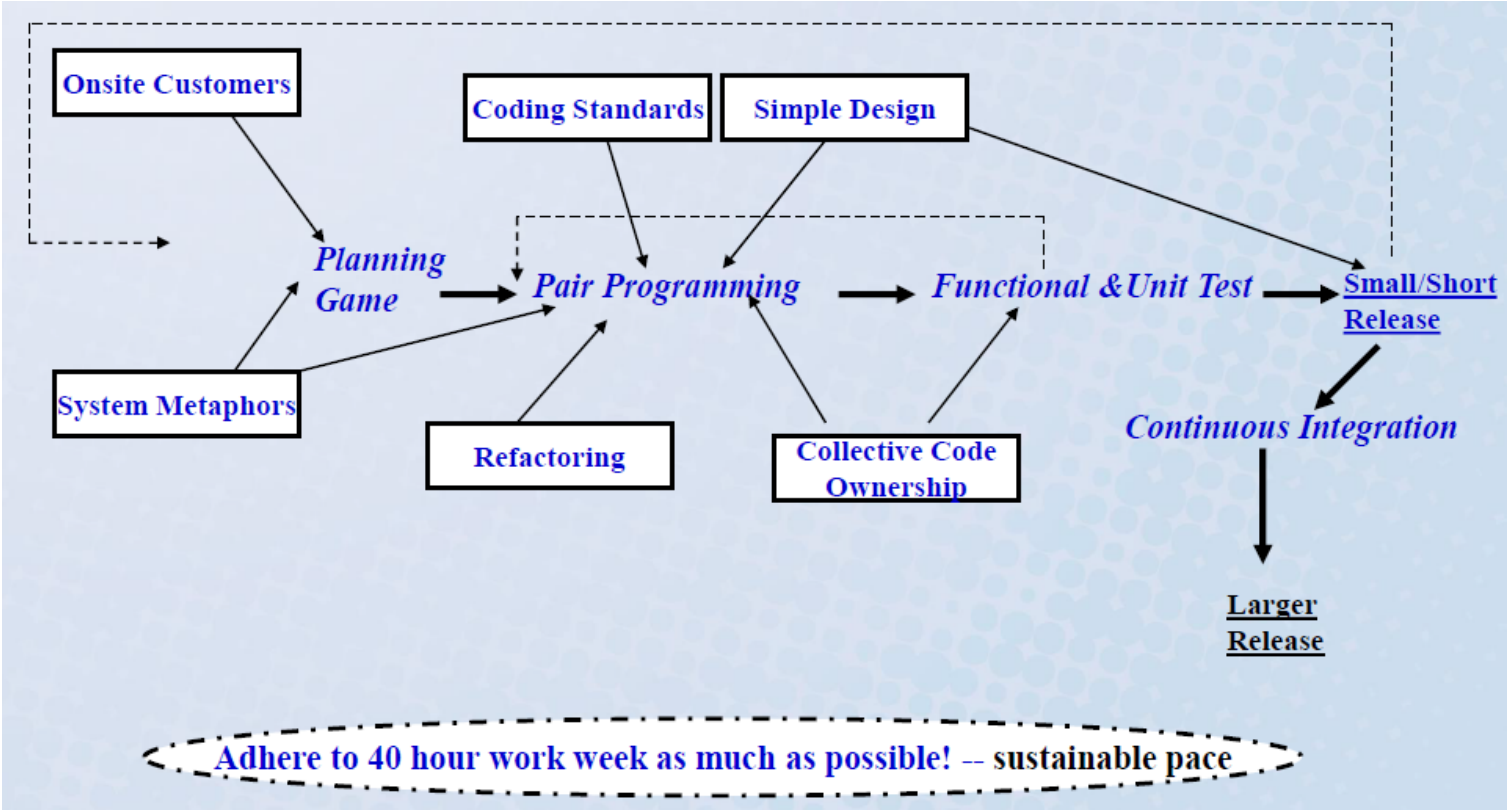
XP's 12 Key Practices

Based on the concept of quick and constant "feedback mechanism" involving:

- Planning Game (small units of requirements)
- Onsite Customer (immediate and better feedback)
- Metaphor (use one set of metaphor for design/architecture)
- Simple Design (just enough to cover what's needed)
- Coding Standard (facilitates better communication)
- Collective Code Ownership (peer pressure to improve code)
- Pair Programming (feedback and shared improvements)
- Refactoring (continuous examination for duplicative design/code)
- Continuous Functional and Unit Testing (100% completion)
- Small/Short Releases
- Continuous Integration (integrating of small releases)
- 40 Hour Work (high morale and energy level)

which includes some recognition of "human" aspects

XP's"Process"


Onsite Customers
Coding Standards
Simple Design
Planning Game
Pair Programming
Functional &Unit Test
Small/Short Release
System Metaphors
Refactoring
Collective Code Ownership
Continuous Integration
Larger Release
Adhere to 40 hour work week as much as possible! -- sustainable pace

Agile, then and now?

Controversial in 2001

> *"... yet another attempt to undermine the discipline of software engineering ... nothing more than an attempt to legitimize hacker behavior." – Steven Ratkin, "Manifesto Elicits Cynicism," IEEE Computer, 2001*

Found acceptance in 2013

- A 2012 study of 66 projects found majority using Agile, even for distributed teams

Agile vs. Non-Agile?

Consider answers to the following questions (Yes $\rightarrow$ Non-Agile; No $\rightarrow$ Agile)

- Is a detailed specification required?
- Are customers unavailable?
- Is the system to be built very very large?
- Is the system to be built very very complex (e.g., real time)?
- Are you using poor software tools?
- Is the system to be built subject to regulation?
- Is team part of a documentation-oriented culture?
- Does the team have very poor programming skills?

Outline

Process Model for Term Project?

What are your answers to the questions in the previous slide?

Agile

- Embraces change as a fact of life: continuous improvement vs. strict phases
- Developers continuously refine working but incomplete prototype until customers happy, with customer feedback on each Iteration ĕvery 1 to 2 weeks)
  - who is your customer?
- Emphasizes Test-Driven Development (TDD) to reduce mistakes, written down *User Stories* to validate customer requirements, *Velocity* to measure progress
  - User stories
  - Velocity
- It is *not* just "hacking".

Test-Driven Development (TDD)

- If short iterations are good, make them as short as possible (weeks vs. years)
- If simplicity is good, always do the simplest thing that could possibly work
- If testing is good, test all the time. Write the test code before you write the code to test.
- If code reviews are good, review code continuously, by programming in pairs, taking turns looking over each other's shoulders.

Outline

In-Class Exercise on Process Models

Check it out on Blackboard

Summary

A brief survey of software development process models

- Traditional (Plan and Document) models
- Agile models
  - Discussion. How to avoid just "hacking"?

Any questions?