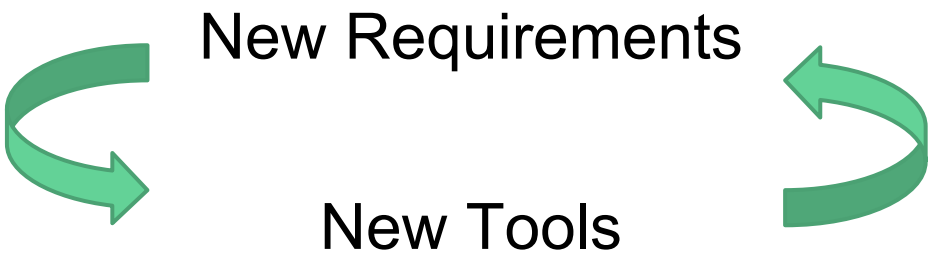
Software Processes

Organizing a software development team

Last Week

- Started talking about collaboration
- Focused on version control, and noticed a repeating pattern:



- Currently, the standard is Distributed Version Control Systems

This Week

- Collaboration between people
- Outside of the source code
- Goals:
  - Get things right
  - Be efficient

Goals

- Get things right
  - Solve a problem/need in the best possible way
  - Deliver the most value to the user
- Be efficient
  - Deliver fast
  - Build easily maintainable systems

Q: Can you think of a way to quantify these goals?

Get It Right + Be Efficient

- More difficult than it sounds
- Especially with uncertainty
  - Changing requirements
  - Unknown requirements
  - External factors (e.g. new technologies, competition)
- Requires a plan ...

Software Process

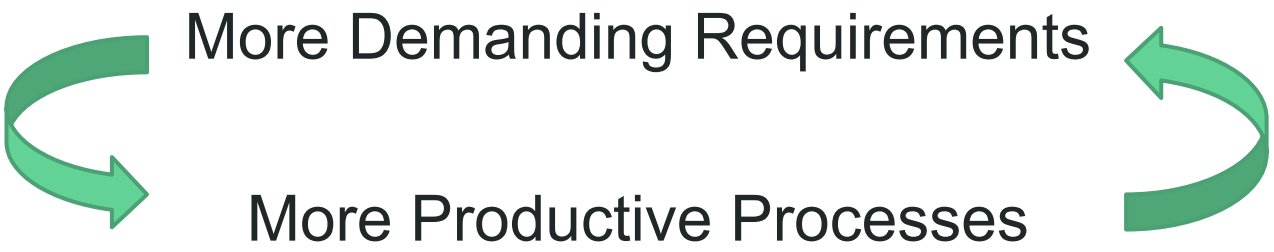
- Many “definitions”:
  - A structured set of activities, used by a team to develop software systems.
  - A team’s standards, practices and conventions.
  - The team’s plan.

Software Process

- Many synonyms:
  - Software Development Process
  - Software Development Methodology
  - Software Development Life Cycle

Software Process

- Like version control tools, software processes keep evolving



- Let’s take a quick chronological tour of some popular software processes ...

Waterfall

Project is divided into a sequence of phases:

1. Requirements
2. Design
3. Implementation
4. Integration
5. Verification
6. Deployment
7. Maintenance

Waterfall

- Originates in the Construction & Manufacturing industries.
- Suitable when “reverting” is costly (or impossible)
- Today it is considered as an anti-pattern (i.e. bad practice) in most software industries.
- A detailed description of the process

Waterfall

- Some arguments for it:
  - The later a bug is found, the more costly it is.

    By spending a long time on Requirements and Design early on, we discover problems/bugs early on.
  - Requirements & design phases produce documentation, which preserves knowledge.
  - The predictions made by the detailed waterfall plan are largely science fiction, but one learns a lot about the problem space.

Waterfall

- Some arguments against it:
  - Not suitable for changing requirements.
  - Many decisions are hard to get right, especially if they must be made before you've written a single line of code.

Prototyping

- The project is done in “cycles”.
- In each cycle, the sequence of phases is:
  - Basic requirements.
  - Implement prototype (e.g. UI with all the functionality mocked/faked).
  - Collect feedback from users.
  - Adjust

Prototyping

- Arguments for it:
    - At the end of each cycle, the team can assess their work, adjust to new requirements and improve its work.
    - More interaction with users helps us ensure that we are building the right product.

Prototyping

- Arguments against it:
    - Prototype ≠ Product.
      This model runs the risk of ignoring the complexities of a real system.
    - You don’t always have patient/forgiving users who will be willing to try your prototypes.

Iterative Incremental Processes

- Build the product incrementally in short iterations.
- In each iteration, the sequence of phases is
  - Plan
  - Design & implement
  - Test
  - Review

Iterative Incremental Process

- Arguments for it:
  - Shorter iterations = More opportunities to adjust and improve.
  - “Fixes” one of the problems with Prototyping - We are developing our product, not just prototypes.
  - We can release at the end of an iteration. Therefore, we can collect feedback from users frequently.

Iterative Incremental Process

- Arguments against it:
  - Unclear long term vision.
  - Might be inappropriate for specific industries.
    - Ex: Microprocessor manufacturers will most likely choose something closer to Waterfall.

The Trend

- With time, software teams
  - Become more flexible and adaptive to changing requirements.
  - Collect user feedback more frequently
  - Release code more frequently
- And then came the term Agile ...

The Agile Manifesto

- The Agile Manifesto is a high-level, general description of a certain type of
  - Software process
  - Team culture
- *Agile* is a big buzzword in the industry.

The Agile Manifesto

- Keep in mind, it was written in 2001.
- In practice, most modern teams follow (most) Agile principles anyway.
  - Make sense for software development
  - Help us get things right and do it efficiently
- In CSC301, we will evaluate some of the popular Agile processes out there ...

Let’s see an example of an Agile process ...

XP - Extreme Programming

- An agile process
- A lot of hype in the late 90’s.
- Prescriptive - Consists of many rules and practices.
- XP: A Gentle Introduction

XP - Extreme Programming

- Some highlights of XP:
  - Iterative incremental model
  - Pair programming
  - Customer’s decisions drive the project.
  - Dev team works directly with a domain expert.
  - Accept changing requirements (even near the deadline).
  - Focus on delivering working software, instead of documentation.

XP - Extreme Programming

- XP is a very detailed model.
- In practice, a team may choose to adopt only a subset of its rules.

Before moving on to other agile processes, let’s focus on one aspect of XP …

Testing

Test Driven Development

- Old concept, rediscovered by Kent Beck
- Part of XP
- Used (to some extent) by many Agile teams
- Main ideas:
  - Write tests, before you write the code.
  - Tests are the specifications that drive the development.

Test Driven Development

- Traditionally, TDD means:
  - Write a failing test.
  - Write the (least amount of) code to pass the test.
  - Repeat.
  - Every now and then refactor/cleanup code.

Test Driven Development

- Some arguments for it:
  - Helps focus on the important features
  - Thinking about how the code will be used, before writing it, leads to better design.
  - Tests serve as clear specifications

Test Driven Development

- Some arguments against it
  - In some cases, testing infrastructure/scaffolding can be very expensive and/or complex to build
  - Too many unit tests, not enough system tests.
  - Sometimes it’s hard to tell whether the side effects of your code are correct.
    Ex: Your code depends on 3rd party API’s.
  - Strict test-first approach is not for everybody (and not for every project).

Test Driven Development

Your individual assignments will highlight some (but not all) aspects of TDD...