CSE435 Software Engineering

Software Processes

Acknowledgements: M. Langford, Pfleeger/Atlee, Pressman

Review

- What is the “Software Crisis?”
  - The realization that many software projects were running over-budget and over-time while also resulting in poor quality.

- What is Software Engineering?
  - The study of systematic and effective processes and technologies for supporting software development and maintenance activities.
  - Communication, Planning, Modeling, Construction, Deployment

- Who are stakeholders?
  - Anyone who has an investment in the final software product.
  - End users, Customers, Practitioners, Technical Managers, Senior Managers
Software Processes

- We want to create **high-quality** software reliably and efficiently.
  - A more **disciplinary approach** than hacking code together and hoping it works
- We need to find an **optimal way to develop/maintain** our software.
  - To **reduce the time and cost** it takes to produce
  - To **minimize undesirable software behavior**
  - To **maximize satisfaction of stakeholders**

A **Software Process** is any set of activities/actions/tasks that are required to build high-quality software.

General Process Framework

- **Major activities:**
  - Communication
  - Planning
  - Modeling
  - Construction
  - Deployment
- **Umbrella activities:**
  - Project tracking/control
  - Risk Management
  - Quality Assurance

- The **sequence and timing** of these can differ depending on your organization and software project.

A **Process Flow** is the sequence and timing of activities.
Why Is Process So Important?

- Large-scale software products can have
  - Millions of lines of code
  - Dozens/hundreds of developers
  - Development/Maintenance cycles that last decades

What happens when one of your developers leaves the job?

What do you do when the project starts falling behind schedule?

What do you do when the customer decides they want more features?

Processes are repeatable, predictable, and streamline communication.

Software Process Models

Waterfall Model (continued)
The Waterfall Model

- Each activity is performed in a **linear sequence**.

**Issues**

- **Requirements often change**, need clarification.
- **No working demo** ready until the end of the process.
- Errors and **mistakes are often caught** late in the process.

Software Process Models

V Model (continued)
**V-Model**

- "Validation and Verification" Model
  - **Verification** – check if software works “correctly”
  - **Validation** – check if software satisfies customer’s needs

**Prototyping**

- Customers typically *do not know what they want exactly.*
  - Remember, often the customer is not the same as the end user.
- **Prototypes** facilitate communication between developers and users.
  - Range from *static mock-ups* to fully *interactive* programs.
  - Demonstrate *what the developer believes the customer expects.*
  - Uncover *unexpected consequences.*
- Be clear to the customer that the **prototype is not the final product.**
  - Customer may assume project is ahead of schedule, request more features.

Prototypes are only *demonstration tools* and should be considered *separate from the final product.*
Software Process Models
Waterfall Model with Prototype (continued)

- Waterfall model with prototyping

Tools to Rapidly Create Mock-Ups and Prototypes

- Tools exist to create simple prototypes to *emulate the user experience*
Software Evolution

- Realistically, **software needs change over time**.
  - Changing customer requirements
  - Security vulnerabilities
  - Integration with newer systems

- **Software Evolution** objectives
  - Ensure software stays **functionally relevant** over time.
  - Ensure software stays **secure and reliable** over time.
  - Ensure software is **flexible for change**.

We need a process that supports **continual development**.

Iterative Process Model

- Construct/deploy **multiple versions** of software product
  - Can be in **parallel, sequentially**, or with **overlap**
  - **Updates can be made more easily** to deal with unexpected changes
  - Assumes **major requirements do not change**
Software Versioning

- **Experimental** versions
  - **Alpha** – testing mostly performed within the organization
  - **Beta** – some subset of customers assist with testing
  - **Release Candidates** – beta versions with potential for release

- **Semantic Versioning Specification** (SemVer)
  - Pattern: **Major.Minor.Patch** (ex. 1.2.0 or 1.0.0-beta)
  - **Major** – increments with API changes
  - **Minor** – increments with backwards compatible additions
  - **Patch** – increments with backwards compatible bug fixes

Software Process Models

Spiral Model (continued)
Software Process Models

Agile Methods

- Emphasis on flexibility in producing software quickly and capably

- **Agile manifesto**
  - Value individuals and interactions over process and tools
  - Prefer to invest time in producing working software rather than in producing comprehensive documentation
  - Focus on customer collaboration rather than contract negotiation
  - Concentrate on responding to change rather than on creating a plan and then following it

---

Software Process Models

Agile Methods: Examples of Agile Process

- Extreme programming (XP)
- Crystal: a collection of approaches based on the notion that every project needs a unique set of policies and conventions
- Scrum: 30-day iterations; multiple self-organizing teams; daily “scrum” coordination
- Adaptive software development (ASD)
Software Process Models

Agile Methods: Extreme Programming

- Emphasis on four characteristics of agility
  - **Communication:** continual interchange between customers and developers
  - **Simplicity:** select the simplest design or implementation
  - **Courage:** commitment to delivering functionality early and often
  - **Feedback:** loops built into the various activities during the development process

Process Models

- Idealized views of the process
- Different models are often used for different subprocesses
  - may use spiral model for overall development
    - prototyping for a particularly complex component
    - waterfall model for other components
Members of the Development Team

- **Requirement analysts**: work with the customers to identify and document the requirements
- **Designers**: generate a system-level description of what the system us supposed to do
- **Programmers**: write lines of code to implement the design
- **Testers**: catch faults; detect errors
- **[Trainers]**: show users how to use the system
- **Maintenance team**: fix faults that show up later
- **[Librarians]**: prepare and store documents such as software requirements
- **Configuration management team**: maintain correspondence among various artifacts

Members of the Development Team (continued)

- Typical roles played by the members of a development team
Why is software development so difficult?

- **Communication**
  - Between customer and developer
    - Poor problem definition is largest cause of failed software projects
  - Within development team
    - More people = more communication
    - New programmers need training

- **Project characteristics**
  - Novelty
  - Changing requirements
    - 5x cost during development
    - up to 100x cost during maintenance
  - Hardware/software configuration
  - Security requirements
  - Real time requirements
  - Reliability requirements

- **Personnel characteristics**
  - Ability
  - Prior experience
  - Communication skills
  - Team cooperation
  - Training

- **Facilities and resources**
  - Identification
  - Acquisition

- **Management issues**
  - Realistic goals
  - Cost estimation
  - Scheduling
  - Resource allocation
  - Quality assurance
  - Version control
  - Contracts

Why is software development difficult? (cont.)
Issues with Rigid Processes

- Rigid processes are useful for tracking and measuring progress
  - Repeatable processes enable a more scientific understanding of what works
- Rigid processes can lead to “breaking the rules”
  - Workers can only tolerate so much bureaucracy and red tape
  - Just because a process worked in one instance, does not mean it works for all.
  - “If your only tool is a hammer, every problem looks like a nail.”
- Rigid processes can be overkill for smaller projects
  - The overhead of the process becomes an obstacle to actual development
  - What happens when you spend more time on the process than the product?

No single solution for all organizations and software projects.

“No Silver Bullet”

- “No Silver Bullet,” [Fred Brooks, 1987]

  - “There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity.”
  - “I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation.”
  - “The most important function that the software builder performs for the client is the iterative extraction and refinement of the product requirements.”
  - “Software construction is a creative process. Sound methodology can empower and liberate the creative mind; it cannot inflame or inspire the drudge.”

Brooks argues that processes are only great with great designers, and any great process encourages creativity.
The Horrors of “Crunch Time”

- **“Crunch time”** occurs whenever projects are behind schedule
  - **Overtime** for extended periods of time
  - Anywhere from **50 to 80 hours a week**
- Prevalent in the **gaming industry but also start-ups** in general
  - **53% of game developers** report that crunch is expected (Less than 18% reported overtime compensation) [Take This, 2019]
- Crunch is often caused by
  - **Unrealistic deadlines**
  - **Lack of communication**

*Crunch is a side-effect of a* **poor process**.

Developer Burnout

- **The Happy-Productive Worker Thesis**
  - Happy employees are more productive and perform better.
- What happens when **developers are (un)happy**? [Graziotin et al., 2018]
  - **Productivity decreases**
  - **Code quality decreases**
  - **Work withdrawal**
- Common reasons for **burnout**
  - Performing at **high cognitive levels** for extended periods
  - **Physical isolation** from teammates
  - Working on **projects that repeatedly fail**
  - **Not feeling valued** at work
  - [https://youtu.be/-gYb5GU0dM](https://youtu.be/-gYb5GU0dM)
1.3 What is Good Software?

- Good software engineering must always include a strategy for producing quality software
- Three ways of considering quality
  - The quality of the product
  - The quality of the process
  - The quality of the product in the context of the business environment
What Is Good Software?  
The Quality of the Product

- Users judge external characteristics
  - (e.g., correct functionality, number of failures, type of failures)
- Designers and maintainers judge internal characteristics (e.g., types of faults)
- Thus different stakeholders may have different criteria
- Need quality models to relate the user’s external view to developer’s internal view


1.3 What Is Good Software?  
The Quality of the Product (continued)

- McCall’s quality model

What Is Good Software?
The Quality of the Process

- Quality of the development and maintenance process is as important as the product quality
- The development process needs to be modeled
- Modeling will address questions such as
  - Where to find a particular kind of fault
  - How to find faults early
  - How to build in fault tolerance
  - What are alternative activities

*Pfleeger and Atlee, Software Engineering: Theory and Practice (edited by B. Cheng)*

What Is Good Software?
The Quality of the Process (continued)

- Models for process improvement
  - SEI’s Capability Maturity Model (CMM)
  - ISO 9000
  - Software Process Improvement and Capability dEtermination (SPICE)

*Pfleeger and Atlee, Software Engineering: Theory and Practice (edited by B. Cheng)*
Capability Maturity Model (CMM)

- **Level 1: Initial**
  - ad hoc
  - success depends on people

- **Level 2: Repeatable**
  - track cost, schedule, functionality

- **Level 3: Defined**
  - use standardized processes

- **Level 4: Managed**
  - collect detailed metrics

- **Level 5: Optimizing**
  - continuous process improvement
  - “built-in” process improvement

Software Engineering Institute:
http://www.sei.cmu.edu/cmm/

What Is Good Software?
The Quality in the Context of the Business Environment

- Business value is as important as technical value
- Business value (in relationship to technical value) must be quantified
- A common approach: **return on investment (ROI)** – what is given up for other purposes
- ROI is interpreted in different terms: reducing costs, predicting savings, improving productivity, and costs (efforts and resources)

*Pfleeger and Atlee, Software Engineering: Theory and Practice (edited by B. Cheng)*
Key points

- **Software processes** are the activities involved with producing software
- Major activities include *Communication, Planning, Modeling, Construction*, and *Deployment*
- A *process flow* and *process model* determines how these activities are organized
- Processes should be able to **cope with change**
- **Software evolution** occurs when software changes to meet new requirements