THE UNIVERSITY of EDINBURGH
informatics

Lecture 2: Introduction to Software Development Activities and Processes

Adriana Sejfia

School of Informatics
University of Edinburgh

Summary

- Overview of software engineering activities
- Notion of a software development process
- Brief history of software engineering
- Software projects vs software products
- Introduction to plan-driven and agile software development processes

Software engineering activities

- Requirements capture
- Design
- Construction/implementation
- Testing, debugging
- Maintenance/evolution

A software (development) process is a description of ***how the above activities are ordered, planned and monitored***.

The ***management of the software process*** is also a key software engineering activity

Requirements capture

- Identifying what the software must do (not how).

- **Interesting issues:**
  - Multiple stakeholders often with different requirements -- how to resolve conflicts?
  - Prioritisation. Which requirements should be met in which release?
  - Maintenance: managing evolving requirements.

Design

- Requirements: what the software must do.
- Design: how should it do that?
  - Higher level than code.
  - Often involves use of a modelling language (e.g. UML)
- Multiple levels of design:
  - architectural design
  - high-level design
  - detailed design

➢ **Interesting issues: understandability ("elegance"); robustness to requirement change; security; efficiency; division of responsibility ("buildability").**

Construction/implementation

- More general than "coding", includes:
  - detailed design (the level that doesn't get written down)
  - coding
  - unit testing
  - managing code evolution
  - writing developer-oriented documentation

➢ **Interesting issues: scale: managing large amounts of detail, esp. code. Need systems that work when it's not possible for one person to know everything.**

Testing and debugging

- Testing happens at multiple levels
  - unit tests written by developers
  - customer acceptance testing.

- Debugging covers things like:
  - "which line of code causes that crash?"
  - “why can't users work out how to do that?"

➢ **Interesting issues: containing cost -- how to test and debug efficiently; when to write tests; software tools to support testing and debugging**

Maintenance/Evolution

- Any post-(major)-release change.
  - fixing bugs
  - enhancing existing functionality
  - coping with a changing world
  - Improving maintainability
- Traditionally an after-thought - mistakenly!
  - In the "total cost of ownership" (TCO) of software system, maintenance/evolution costs often dwarf development costs.

➢ **Interesting issues: retaining flexibility; when to evolve system and when to replace**

What are processes about?

- Ordering activities
- Outcomes of activities.
- Arrangement of people & resources to carry out activities
- Planning in advance of execution, predicting time/cost/resources
- Risk reduction
- Monitoring
- Enabling their own adaptation

➢ Processes are complex and creative.

Brief history of software engineering: origins

- 1935: Alan Turing: idea of "software" as a computer program
- 1958: John Tukey used "software" term in print
- 1963/4: Margaret Hamilton (Apollo 11, Skylab space shuttle) first coined the term "software engineering" to distinguish her work from hardware engineering and give it legitimacy
- 1960s-1980s: "software crisis" due to rapid computing power increase at decreasing cost and difficulties to develop large complex software systems
- 1968-1969: NATO conferences on software engineering

Brief history of software engineering: the 1970s-early 1990s

- Microcomputer revolution
- Software engineering develops as a discipline
- Virtually all professional software meant to automate businesses: custom, "one-off', usually long-life
- **Software projects set up to develop such systems: external customers required (through contract: the requirements document) and paid for custom functionality, the development, maintenance of the system, and any changes**
- **Development of plan-driven software development processes: controlled and rigorous ways of developing software; assumption that a lot of preparation is needed before writing a program; need for thorough documentation including graphical models of the software**

Brief history of software engineering: since the 1990s

- Realisation that most businesses could manage with generic software built for common problems (cheaper, quicker to get)
- Increase in the need for small to medium software systems
- Dissatisfaction with plan-driven approach due to considerable overheads, late delivery, difficulty to respond to changes
- **Developers started coming up with generic software products, with full control over their features, implementation and lifetime; paying customers only after release**
- **Development of agile software development processes: focus on software itself; delivering working software quickly to customers; avoiding work with dubious long-term benefits; reducing documentation**

Project vs product based software engineerin

Project-based SE	Product-based SE
Initiated by an external customer who presents a problem	Initiated by the developer who identifies an opportunity
Requirements captured from customer starting from problem	Features decided by developer starting from opportunity
Developed based on customer needs defined in requirements	Developed as generic solution, not for specific customer
Long-lived, maintained for customer	Life duration decided by developer
Changes decided and paid by customer	Changes decided by the developer

Important notes:

These were typical of the described moments in history. However, not all systems were the same and characteristics can be mixed.
Plan-driven best suited for software projects, and agile for software products, with processes in their pure forms
In reality, processes and system types may be swapped; Moreover, the two types of processes often mixed.

Brief history of software engineering: another approach since the 2000s

- More reuse of existing software rather than developing software from scratch: integration, configuration for customer
- Development of reusable software (stand-alone application systems, reusable components or packages, web services)
- Adoption of reuse-based development processes: mix of plan-driven and agile, because requirements gathered in advance (like in plan-driven) but things can be changed and components reconfigured, often incremental (like in agile, see below)

Plan-driven processes are processes where all the process activities are **planned in advance** and **progress is measured against this plan**

(Ian Sommerville)

Plan-driven processes - main characteristics and applicability

- A lot of time and effort are spent in planning the system
- Everything is thoroughly documented; attempt to always keep documentation up-to-date
- Use of modelling (e.g. UML) for documenting requirements and design
- The system is specified in detail before implementation begins
- Errors, omissions and misunderstandings in the requirements often discovered late in the implementation (costly to fix)
- Reticence to change; inability to respond quickly to it
- Most appropriate for long lifetime, critical and embedded systems

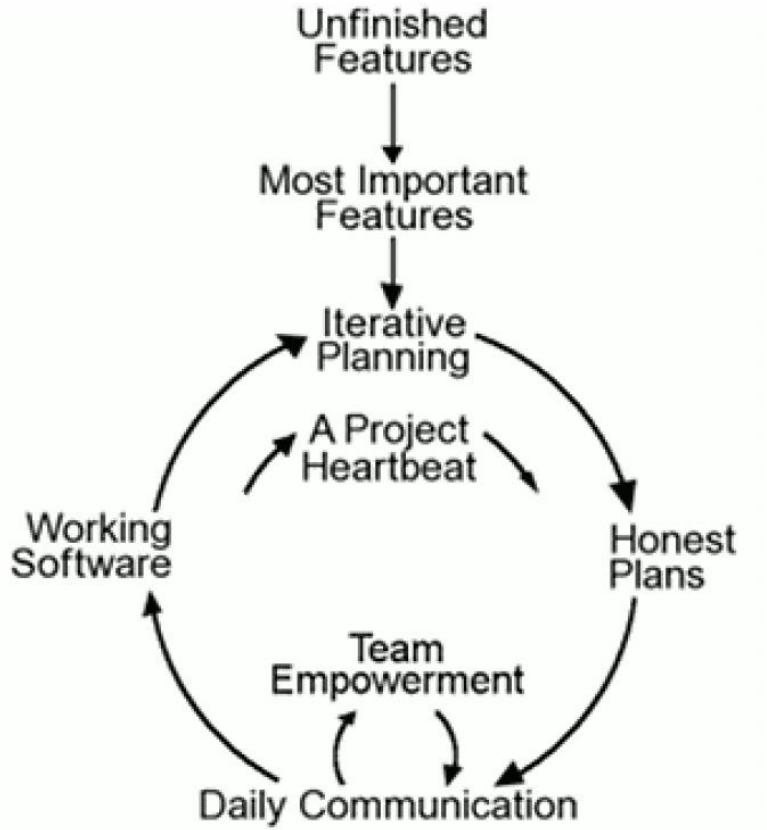
Agile processes - introduction

- The Agile Manifesto http://agilemanifesto.org:

- "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
  - **Individuals and interactions over processes and 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."

***IMPORTANT! By "customer" meant initiator of requirements (even business person from development company!) here and in following slides.**

Agile flowchart



12 principles of agile (from Agile Manifesto)

- 1. "Highest priority is to satisfy the customer through early and continuous delivery of valuable software".
- 2. "Welcome changing requirements, even late in development".
- 3. "Deliver working software frequently: couple of weeks - couple of months, with a preference to the shorter timescale".
- 4. "Business people and developers must work together daily".

12 principles of agile (from Agile Manifesto)

- 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 (...) is face-to-face conversation".
- 7. "Working software is the primary measure of progress".
- 8. "Sustainable development (...) to maintain a constant pace indefinitely".

12 principles of agile (from Agile Manifesto)

- 9. "Continuous attention to technical excellence and good design enhances agility".
- 10. "Simplicity -- the art of maximizing the amount of work not done -- is essential".
- 11. "The best architectures, requirements, and designs emerge from self-organizing teams".
- 12. "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly".

Where to apply plan-driven vs agile processes

Agile home ground	Plan-driven home ground
Low criticality	High criticality
Senior developers	Junior developers
Requirements change often	Requirements do not change often
Small number of developers	Large number of developers
Culture that responds to change	Culture that demands order

Reading

- **Essential:**
  - On software engineering activities: Sommerville SE Chapter 2 section 2.2.
  - On software projects and software products and their engineering: Sommerville ESP Chapter 1 up to 1.1.
  - On software development processes (overview): Sommerville SE Chapter 2 until 2.2
  - On intro to agile software development processes (and comparison with plan-driven): Sommerville ESP Chapter 2 up to 2.2.
  - An excellent overview of processes by Ian Sommerville: https://www.youtube.com/watch?v=q8X2Rk5sRFI&t
- **Recommended:**
  - Stevens Chapter 1
  - The Agile Manifesto: http://agilemanifesto.org/