SYSTEMS PLANNING

Agile Software Development

Class content

What to expect from this class?

- An introduction to **agile software development**
- **Extreme programming (XP)** as the most famous agile development method
- **Scrum** as **project management** approach to agile development

Source: https://pixabay.com/en/checklist-check-list-marker-2077020/
by TeroVesalainen |CCO Creative Commons

Class learning goals

MCI MANAGEMENT CENTER INNSBRUCK

What should be your learning outcome?

- When to **potentially use agile** and when to better use more **traditional software development** methods
- The main **characteristics of agile development**
- **Advantages** and **challenges** of agile methods
- How **extreme programming** works
- How **Scrum** compares to traditional project management methods

Source: https://pixabay.com/en/learning-hint-school-subject-3245792/ by harishs |CC0 Creative Commons

Rapid Software Development

MCI MANAGEMENT CENTER INNSBRUCK

Rapid software development: Why?

- Businesses operate in **global, rapidly changing environments**
- New **opportunities** and **markets**
- Changing **economic conditions**
- Emergence of **competing products and services**
- Businesses are willing to **trade quality** and **compromise** on requirements
- **While safety-critical control systems** require complete analysis (**plan-driven approach**) **for fast-moving business environments this approach can be too slow!**

Source: https://pixabay.com/en/deadline-stopwatch-clock-time-2636259/ by freeGraphicToday |CC0 Creative Commons

History of agile software development

1980s
- IBM introduced incremental development (Mills et al., 1980)
- Fourth-generation languages (Martin, 1981)

1990s
- Dynamic Systems Development (Stapleton, 1997)
- Scrum (Schwaber & Beedle, 2001)
- Extreme Programming (Beck, 1999)

Source: https://pixabay.com/en/chalkboard-story-blogging-believe-620316/ by 742680 |CC0 Creative Commons

Goal: Produce useful software quickly, incrementally including new system functionality.

In the old days… (i.e. 1980s and early 1990s)

It was thought the best way to achieve software was through:
- Careful **project planning**
- Formalized **quality assurance**
- **Analysis and design methods**

Because software was developed
- By **large teams**
- Working for **different companies**
- **All over the world**

Source: https://pixabay.com/en/africa-namibia-solitaire-oldtimer-1788022/ by wiggijo |CC0 Creative Commons

This type of SD is necessary for
- Handling **multiple** development **teams**
- Developing **critical systems**
- And when **different teams** are involved in **maintaining the software over time**

The rise of agile methods…

However:
When **heavyweight**, **plan-driven development methods** are applied to small and medium-sized businesses:

- they **dominate the software development process**
- and consequently they may lead to **dissatisfaction**

In the 1990s a number of **agile methods** were proposed that allowed the team to **focus on the software** itself **rather than on its design and documentation**.

Source: https://pixabay.com/en/lifting-box-wood-heavy-crate-24401/ by Clker-Free-Vector-Images |CC0 Creative Commons

The Agile Manifesto

We are uncovering **better ways of developing software** by doing it and helping others. Through this work we have come to value:

Manifesto for Agile Software Development

- **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 Software Development

MCI MANAGEMENT CENTER INNSBRUCK

Fundamental characteristics of agile software development

- Specification, design and implementation are **interleaved;** documentation is **generated automatically**;
- System is developed in **a series of versions**
- User interfaces are often developed using a development system that allows for quickly **"drawing the interface"** ;
- Increments are small and new releases are made available every two or three weeks **"release early and often"**;
- The **customer is involved** in the development process to get rapid feedback;
- It uses **informal communication instead of formal** written meeting minutes;

Source: https://pixabay.com/en/person-character-characteristics-662740/ by johnhain |CC0 Creative Commons

Agile Software Development

MCI MANAGEMENT CENTER INNSBRUCK

Challenges:

- Customer involvement: **Do I have a customer who has time and represents all stakeholder views?**
- People not process: **Do I have the right people for this type of teamwork?**
- Embrace change: **Difficult with different stakeholders; every stakeholder has different priorities!**
- Maintain simplicity: **Is there enough time to carry out system simplification?**
- Organization: **Does my organization allow for such informal development processes?**

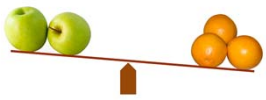
Source: https://pixabay.com/en/chess-strategy-chess-board-316658/ by PublicDomainPictures |CC0 Creative Commons

Some more challenges…

- **Requirements document** (functional specification) is usually part of the contract
- **Another type of contract** may be necessary
- Customer needs to **pay for the time required** rather than the development of a set of requirements.

Source: https://pixabay.com/en/board-game-checkmate-chess-1846400/ by Pexels |CC0 Creative Commons

Plan-driven vs. Agile Development



Source: https://pixabay.com/en/compare-comparison-scale-balance-643305/ by Tumisu |CC0 Creative Commons

Plan-driven
- Separate stages
- Output from one stage is used as basis for the next stage
- Iteration occurs within activities (stages)

Agile Development
- Design and implementation are central
- This incorporates requirements elicitation and testing
- Iteration occurs across activities

Note: Most software products include practices from plan-driven and agile approaches.

Plan-driven or Agile Development?

1. **Is detailed specification important?**
   Probably you need a plan-driven approach
2. **Is incremental delivery realistic?**
   If yes, you should consider agile methods
3. **How large is the system to be developed?**
   The larger the system the more likely it is that you'd need a plan-driven approach
4. **What type of system is being developed?**
   Critical systems vs. non-critical systems

Source: https://pixabay.com/en/scale-question-importance-balance-2635397/ by qimono |CC0 Creative Commons

Plan-driven or Agile Development?

5. **What is the expected system lifetime?**
   Long lifetime may require more information.
6. **What support technologies are available?**
   Agile methods require good tools.
7. **How is the development team organized?**
   Distributed teams make it hard for agile methods to work.
8. **Are there cultural issues to consider?**
   Traditional engineers have a culture of plan-based development

Source: https://pixabay.com/en/scales-justice-balanced-black-310962/ by Clker-Free-Vector-Images |CC0 Creative Commons

Plan-driven or Agile Development?

9. **How good are the designers/programmers?**
   Agile methods usually require better skills.
10. **Is the system subject to external regulation?**
    External regulators usually require detailed documentation.

Note: In reality, whether the project is labeled as plan-driven or agile is **not important** as long as the client receives an executable software system that meets their needs.

Source: https://pixabay.com/en/scales-balance-weighing-weight-1333455/ by aitoff |CC0 Creative Commons

Some Examples

Source: https://pixabay.com/en/example-for-example-z-b-2427501/ by AxxLC |CC0 Creative Commons

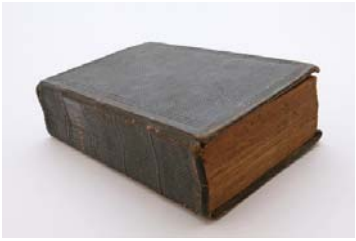
- **Extreme Programming (XP)** (Beck, 1999)
- **Scrum** (Schwaber & Beedle, 2001)
- Crystal (Cockburn, 2004)
- Adaptive Software Development (Highsmith, 2000)
- Dynamic System Development (DSDM) (Stapleton, 1997)
- Feature Driven Development (Palmer & Felsing, 2002)

Integration with traditional methods:
- Agile modeling (Ambler & Jeffries, 2002)
- Agile instantiation of RUP (Larman, 2002)

Principles

All these agile methods are **based on incremental development**. Thus, they share a set of principles:

- **Customer involvement**
- **Incremental delivery**
- **People not process**
- **Embrace change**
- **Maintain simplicity**



Source: https://pixabay.com/en/afterlife-bad-baptist-belief-bible-1238610/ by Meditations |CC0 Creative Commons

Extreme Programming (XP)

- Best known and **most widely used** agile method
- Iterative development pushed to **'extreme' levels**
- **E.g.** several new versions **developed, integrated and tested in only one day**
- **Requirements are** expressed through **scenarios (user stories)**
- Programmers **work in pairs**
- **Tests** are developed before **writing code**

Source: https://pixabay.com/en/motorcycle-speed-sports-go-to-fly-293571/ by Bergadder |CC0 Creative Commons

Extreme Programming (XP)

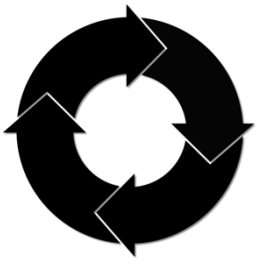
Integrates the agile principles:

- **Incremental development:** small, frequent releases of systems; customer stories drive development;
- **Customer involvement:** continuous engagement of the customer; customer is part of the development team;
- **People not processes:** pair programming, collective ownership; sustainable development process;
- **Change:** regular releases; test-first; continuous integration;
- **Maintaining simplicity:** refactoring

Source: https://pixabay.com/en/ethics-right-wrong-ethical-moral-2991600/ by Tumisu |CC0 Creative Commons

The XP Process

- Customers are intimately involved **prioritizing system requirements**
- **No list of system functions** but a number of scenarios
- **'Story cards'** are used to encapsulate customer needs
- Story cards are input for **'the Planning Game'**
- Stories are **broken down into tasks**
- Customer prioritizes 'stories', identifying those that can already deliver useful **business support**
- A 'story' should be implementable **within 2 weeks**



Source: https://pixabay.com/en/cycle-circuit-process-change-2019535/ by kmicican |CC0 Creative Commons

The XP Process

- Some prototyping may be required to explore possible solutions. In XP this is called **Spike**
- **Spike = an increment where no 'real' programming is done** (only prototyping)
- XP is an **extreme approach** to incremental development i.e. software may be built **several times per day**; releases to **customers roughly every 2 weeks**
- New software needs to **pass all automated tests** (new ones and those designed for earlier features)

Source: https://pixabay.com/en/iguana-portrait-profile-close-up-1738056/ by skeeze |CC0 Creative Commons

The XP Process and Change

- Traditional SE emphasized **'design for change'**

XP says:
- Designing for change is often **wasted effort**
- Anticipated changes **never materialize**
- **Different change** may be requested

"Change will happen, we deal with it when it actually occurs!"

Source: https://pixabay.com/en/think-switch-arrows-rethinking-2177839/ by geralt |CC0 Creative Commons

How the XP Process implements change

Traditional SE:
- Incremental development tends to degrade software structure
- **It focuses on finding workarounds to problems**
- **Result: duplicated code; inappropriate use of features;**

In XP:
- Software should constantly be **refactored**
- Programming teams **search for improvements**
  e.g. reorganization of class hierarchies; tiding up and renaming of attributes and methods; defining program libraries;

Source: https://pixabay.com/en/road-sign-arrow-advance-change-1076229/ by geralt |CC0 Creative Commons

Testing in XP

XP emphasizes the importance of testing

- **Test-first** development
- Incremental **test** development **from scenarios**
- **User involvement** in the test development and validation
- The use of automated **testing frameworks**

Source: https://pixabay.com/en/checklist-clipboard-questionnaire-1622517/ by 472301 |CC0 Creative Commons

Test-first development

Write test before the code

- **Ambiguities** have to be **clarified**
- Each **task** generates one or more **unit tests**
- **Customer** develops **acceptance tests**
- Tests should be **stand-alone**, simulate the submission of **input**, and check whether the results meet the **output** specification
- **Automated testing frameworks**, such as JUnit (Massol & Husted, 2003) help writing tests

Source: https://pixabay.com/en/checklist-clipboard-pen-paper-1643781/ by Deedster |CCO Creative Commons

Pair programming

Sit together to develop software

- Supports **collective ownership**
- Supports **informal review**, which is much **cheaper than formal code reviews** such as code inspections
- Helps **refactoring** as it leads to immediate support

Source: https://pixabay.com/en/monitor-binary-binary-system-1307227/ by geralt |CCO Creative Commons

Pair programming efficiency

Williams et al. (2000)
- **Productivity** with pair programming seems to be **comparable** with that of two people
- Pairs **discuss the software before** development which leads to **fewer false starts** and **less rework**

Parish et al. (2004)
- Could not reproduce these results

Nevertheless: Sharing knowledge during programming helps reducing risk.

Source: https://pixabay.com/en/skills-competition-exhibition-2460820/ by bbpc |CC0 Creative Commons

Agile Software Development

Agile Project Management

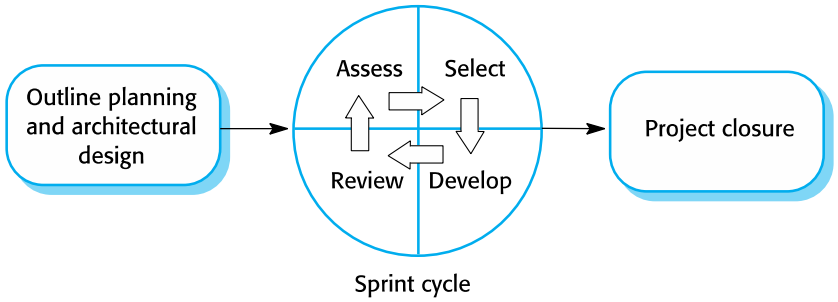
Traditional approach
- Deliver on time and within the planned schedule
- Hence: plan-driven development

Agile approach
- plan-driven does not work as requirements are developed and finalized incrementally
- Hence: **The Scrum approach**

Scrum phases:
- Outline planning phase
- Sprint cycles
- Project closure phase

Source: https://pixabay.com/en/problem-analysis-solution-hand-67054/ by geralt |CC0 Creative Commons

Agile Project Management



Source: Sommerville, I. (2011). Software Engineering, 9th Ed. Boston, MA, USA: Pearson Education Inc.

Sprint Cycle

- **Most central** phase of Scrum
- The planning **unit in which the work is done**

Key characteristics
- Fixed length of usually **2-4 weeks**
- Starting point is the **product backlog** (list of work to be done)
- **Selection phase** involves the selection of features to be developed during the sprint
- **Daly meetings** to review work in which the **'Scrum master'** acts as an interface between developers and the customer(s)
- At the **end** of each sprint the work is **presented to the stakeholder(s)**

Source: https://pixabay.com/en/athlete-training-track-sprint-640335/ by skeeze | CCO Creative Commons

Scrum Team Roles

- **The Scrum Product Owner**
  the **management** role that serves as the **interface** between the team
  and other involved parties (stakeholders);
  Responsible for **business decisions** and **return on investment**;
  Responsible for '**user stories**' and the **selection /prioritization** of **features** to be implemented
- **The Scrum Master**
  a team coach that gives advice and makes sure that all team members '**play by the rules**'
- **The Scrum Team**
  The **development** team

Source: https://pixabay.com/en/teamwork-team-gear-gears-drive-2198961/ by geralt |CC0 Creative Commons

Advantages of Scrum

- Team should empower decision making
- No Project manger but **Scrum master** who arranges daily meetings, tracks backlog, records decisions, measures progress, communicates with customers and product owner;

Rising & Janoff, 2000
Scrum in a **telecommunication SD environment**
- Product is broken down in **understandable chunks**
- Unstable requirements **do not hold progress**
- Whole team has **visibility** which leads to **better communication**
- Customers see **on-time delivery** of increments
- **Trust** between customers and developers is established

Source: https://pixabay.com/en/like-facebook-social-media-icon-3000958/ by raphaelsilva |CC0 Creative Commons

MANAGEMENT CENTER INNSBRUCK

Scaling Agile Methods

Agile methods were developed for use by small programming teams.
Large software development is different:

- Large systems usually involve **separate teams developing sub-systems**, sometimes in **different places and even different time zones**
- Large systems are **'brownfield systems**' i.e. they interact with a number of existing systems
- A significant part of the development is spent on **system configuration**
- Large systems development is often constrained by **external rules and regulations**
- Large systems usually have a diverse **set of stakeholders** for which it is impossible to bring in all sides

Source: https://pixabay.com/en/cloud-monitor-cloud-computing-3017392/ by geralt | CCO Creative Commons

Agile methods have to be **adapted to cope with large systems engineering!**

MANAGEMENT CENTER INNSBRUCK

What you should have taken away from this class:

- Agile methods are **incremental development methods** that focus on rapid development, **frequent releases of the software, reducing process overheads, and producing high-quality code**. They involve the customer directly in the development process.
- The decision on whether to use an agile or a plan-driven approach to development should depend on **the type of software being developed, the capabilities of the development team**, **and the culture of the company** developing the system.
- **Extreme programming** is a well-known method that integrates a **range of good programming practices** such as **frequent releases** of the software, **continuous software improvement**, and **customer participation** in the development team.
- A particular strength of extreme programming is the development of **automated tests** before a program feature is created.
- **The Scrum method** is an agile method that provides a **project management framework**. It is centered around a set of **sprints**, which are **fixed time periods when a system increment** is developed. Planning is based **on prioritizing a backlog of work** and **selecting the highest-priority tasks** for a sprint.
- **Scaling** agile methods for large systems **is difficult**. Large systems need up-front design and some documentation.

References

- Ambler, S. W., & Jeffries, R. (2002). Agile Modeling: Effective Practices for Extrem Programming an the Unified Process. New York, NY, USA: John Wiley & Sons.
- Beck, K. (1999). Embracing Change with Extreme Programming. IEEE Computer, 32(10), 70-78.
- Cockburn, A. (2004). Crystal Clear: A Human-Powered Methodology for Small Teams. Boston, MA, USA: Addison-Wesley.
- Highsmith, J. A. (2000). Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. New York, NY, USA: Dorset House.
- Hopkins, R., & K., J. (2008). Eating the IT Elephant: Moving from Greenfield Development to Brownfield. Boston, MA, USA: IBM Press.
- Larman, C. (2002). Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and the Unified Process. Boston, MA, USA: IBM Press.
- Leffingwell, D. (2007). Scaling Software Agility: Best Practices for Large Enterprises. Boston, MA, USA: Addison-Wesley.
- Martin, J. (1981). Application Development Without Programmers. Englewood Cliffs, NJ, USA: Prentice:Hall.
- Massol, V., & Husted, T. (2003). JUnit in Action. Grennwich, CT, USA: Manning Publications Co.
- Mills, H. D., O'Neill, D., Linger, R. C., Dyer, M., & Quinnan, R. E. (1980). The Management of Software Engineering. IBM Systems Journal, 19(4), 414-477.
- Palmer, S. R., & Felsing, J. M. (2002). A Practical Guide to Feature-Driven Development. Englewood Cliffs, NJ, USA: Prentice Hall.
- Parish, A., Smith, R., Hale, D., & Hale, J. (2004). A Field Study of Developer Pairs: Productivity Impacts and Implications. IEEE Software, 21(5), 76-79.
- Rising, L., & Janoff, N. S. (2000). The Scrum Software Development Process for Small Teams. IEEE Software, 17(4), 26-32.
- Schwaber, K., & Beedle, M. (2001). Agile Software Development with Scrum. Englewood Cliffs, NJ, USA: Prentice Hall.
- Schwaber, K., & Beedle, M. (2001). Agile Software Development with Scrum. Englewood Cliffs, NJ, USA: Prentice Hall.
- Smits, H., & Pshigoda, G. (2007). Implementing Scrum in a Distributed Software Development Organization. AGILE.
- Stapleton, J. (1997). DSDM Dynamic System Development Method. Harlow, UK: Addison-Wesley.
- Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the Case for Pair Programming. IEEE Software, 17(4), 19-25.

Thank you for your attention!

mentoring the motivated.