Agile Methods

- It is perhaps the prevailing view that agile methods are a creation of the 1990s onwards, especially with the arrival of Scrum and XP.
- However, agile concepts would appear to have been in use much earlier, and therefore we look to RAD as a possible starting point for the emergence of explicit / defined agile thinking.
- Some discussion on the origins of agile software development can be found here:
- *In search of the origins and enduring impact of agile software development.* In: 2018 International Conference on Software and System Process ICSSP '18, 26-27 May 2018, Gothenburg, Sweden. ISBN 978-1-4503-6459-1

Emergence of Agile Methods

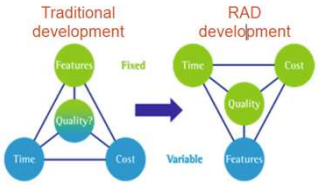
- Concept of **Rapid Application Development** (RAD)
  - Term coined by James Martin in 1991
  - Based on Scott Shultz's **Rapid Iterative Production Prototyping** (RIPP) project in DuPont in 1984
- RAD Mission
  - Increase **speed of development** at a **reduced cost** without sacrificing **system quality**
- RAD was not Radically new
  - Synthesized available tools and techniques
  - But with **new principles** and **focus**
  - Primarily management techniques to **overcome bureaucratic obstacles** to faster development and delivery of systems

Emergence of Agile Methods

- And while **Rapid Application Development** (RAD) is certainly not considered to be a contemporary so-called "agile method", it does deomonstrate that the thinking leading to the Agile Manifesto and various agile development methods has its origins in much earlier concepts.

Fundamental Principles of RAD

- Active User Involvement
- Small Empowered Teams
- Frequent Delivery of Working Products
- Iterative Development and Prototyping
- Computer Assisted Systems Engineering (CASE) Tools



Fundamental Principles of RAD (1)

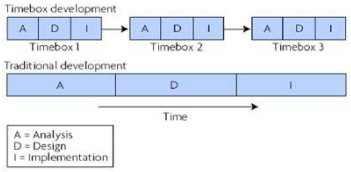
- Active User Involvement
  - Gulf between developers and users can result in (often significant) **delays** in traditional approaches
  - Users know business needs best
  - Users are heterogeneous – different background/expertise
  - Involvement vs. Commitment (IMPORTANT difference)

Fundamental Principles of RAD (2)

- Small Empowered Teams
  - Creative dynamic through skilled highly-motivated teams
  - Eliminate bureaucratic and delayed communication and decision making
    - **Avoidance** of decision-making characteristic of traditional approach — user sign-off etc., extended review and inspection cycles.
  - Small teams – communication channels increase significantly as size increases
  - See Brooks's Law
- Roles **not** Individuals – **no 'I' in team**
  - Project manager (Scrum Master), designer, analyst, tester, technical specialist, database specialist, product owner, customer, QA etc.

Fundamental Principles of RAD (3)

- **Frequent Delivery of Working Products**
  - **Rigid time-boxing** — 2 to 6 month slots (Scrum typically 2-4 weeks)
  - Project management simpler
    - Easier to keep **focused on necessary** activities in shorter time-scale: unnecessary activities avoided
  - **Product-focused** rather than **process** (activity) focused



Fundamental Principles of RAD (4)

- **Iterative and incremental development**
  - Delivery of sub-systems followed by refinement
  - **Not linear** sequence of phases
  - Systems evolve: **never complete** — new requirements emerge
  - **Impossible** to specify requirements completely in advance—specified at a level appropriate to knowledge at time
  - Iteration **inevitable** — not seen as rework delaying development
    - Design -> Testing -> Redesign -> Retesting etc.
- Prototyping
  - Working model of the system — parallels in other disciplines
  - Requirements specification is a **learning process**

Fundamental Principles of RAD (5)

- Computer Assisted Systems Engineering (CASE) Tools
  - Need to eliminate or automate routine, time-consuming tasks
    - Code Generation,
    - Version Control,
    - Bug Tracking,
    - Documentation,
    - Testing,
    - Continuous Integration,
    - Project Management

Fundamental Principles of RAD

- Active User Involvement
- Small Empowered Teams
- Frequent Delivery of Working Products
- Iterative Development and Prototyping
- Computer Assisted Systems Engineering (CASE) Tools

Agile methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- One of the aims of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

11

The Agile Manifesto - I

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

The Agile Manifesto – II

- Our **highest priority** is to **satisfy** the **customer** through early and **continuous delivery** of valuable software.
- **Welcome changing** requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- **Business people and developers must work together daily** throughout the project.

The Agile Manifesto – III

- Build projects around **motivated individuals**. Give them the environment and support they need, and **trust them to get the job done**.
- The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.
- **Working software** is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

The Agile Manifesto – IV

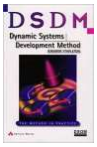
- **Continuous attention to technical excellence** and good design enhances agility.
- **Simplicity** – the art of maximizing the amount of work not done – is essential.
- The **best** architectures, requirements, and designs emerge **from self-organizing teams**.
- At regular intervals, the team **reflects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

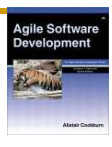
What are Agile Methods?

- **Agility** can be defined as the ability to create change and **respond to change**
- **Agile software development** is a conceptual framework for undertaking software engineering projects.
- Most agile methods attempt to **minimize risk** by developing software in **short timeboxes**, called **iterations**, which may typically last one to four weeks.
- Each iteration is like a miniature software project of its own, and includes all of the tasks necessary to release the **mini-increment** of new functionality.

Which are different Agile Methods?

- **Extreme Programming (XP)**
- **Scrum**
- Agile Modeling
- Adaptive Software Development (ASD)
- Crystal Clear and Other Crystal Methodologies
- Dynamic Systems Development Method (DSDM)
- Feature Driven Development
- **Lean software development**
- Agile Unified Process (AUP)
- **Kanban**





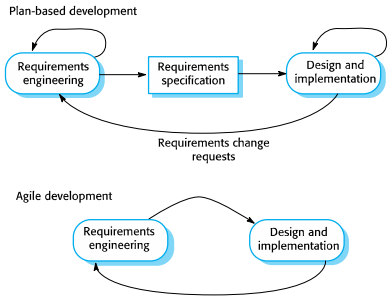
Agile Methods v/s Traditional Methods

- Agile methods **emphasize real-time communication**, preferably face-to-face, over written documents.
- Some principles underlying code production are: **keep it simple**, have one **shared metaphor** to guide system development, regularly restructure the system to improve it (**refactoring**), **continuously integrate and test**, and follow **coding standards**.
- The **customer** is **on site** and is part of the development team.

*Agile methods like XP rely on the close collaboration of activity of engaged individuals with ordinary talents and has **the ability to flexibly schedule the implementation of functionality**, responding to changing business needs.*

(Extreme Programming explained: Embrace Change By: Kent Beck with Cynthia Andres; 2nd ed.2005)

Plan-driven and agile development



19

Characteristics of Agile Methods

- **Agile software processes…**
  - Increase **responsiveness** of software teams
    - Changing requirements
    - Strong customer involvement
  - **Reduce overhead** to speed up development
  - Focus on **people**
  - Focus on **delivering** software

Characteristics of Agile Methods

- **Agile software processes…**
  - Adaptive to requirements changes . . . embracing change
    - Customers just don't know
    - Developers just don't understand
  - **Iterative**/Explorative
  - **Process itself adapts** to project . . . disciplined, yet flexible
  - Rely on **collaboration/innovation** through group interaction
  - Simplicity… "Just do it."
  - **Minimal documentation** (efficiency, low inertia)

Pros

- Can quickly determine the scope of the next release, combining business priorities and technical estimates
- The **customer decides** scope, priority, and dates from a business perspective, while technical people estimate and track progress
- **Incremental** development: Consistent with most modern development methods
- Emphasis on **responsibility** for **quality**
- **Keep it simple**, have one shared metaphor to guide system development, regularly restructure the system to improve it, continuously integrate and test
- "Agile Methods like XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software" -- Kent Beck, Extreme Programming Explained

Cons

- The whole thrust of these methodologies can be summed up with the phrase "**you'll just have to trust me**"
- **Code-centered** rather than design-centered development
- **Quality through testing**: A development process that relies heavily on testing is unlikely to produce quality products
- The **lack of an orderly design** process and the use of unstructured reviews mean that extensive and time-consuming testing would still be needed
- Agile Methods are **only briefly described**
- Reliance on **verbal** communication => low support for transition