

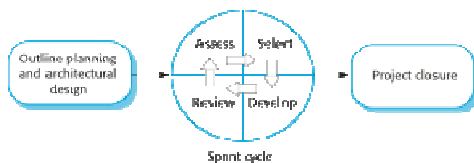
Agile project management

- The principal **responsibility** of software project managers is to manage the project so that the software is **delivered on time** and within the **planned budget** for the project.
- The **standard approach** to project management is **plan-driven**. Managers draw up a plan for the project showing **what should be delivered**, **when** it should be delivered and **who** will work on the development of the project deliverables.
- Agile project management requires a different approach, which is adapted to incremental development and the particular strengths of agile methods.

Scrum

- The Scrum approach is a general agile method but its focus is on managing **iterative development** rather than specific agile practices.
- There are three phases in Scrum.
 - The initial phase is an **outline planning phase** where you establish the general objectives for the project and design the software architecture.
 - This is followed by a series of **sprint cycles**, where each cycle develops an increment of the system.
 - The project **closure phase wraps up the project**, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

The Scrum process



The Sprint cycle

- Sprints are **fixed length**, normally 2–4 weeks. They correspond to the development of a release of the system in XP.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.
- **The selection phase involves all of the project team who work with the customer** to select the features and functionality to be developed during the sprint.

The Sprint cycle

- Once these are agreed, the team organize themselves to **develop** the software. **During this stage** the team is **isolated** from the customer and the organization, with all communications channelled through the so-called **'Scrum master'**.
- The role of the **Scrum master** is to protect the development team from **external distractions**.
- At the **end of the sprint**, the work done is **reviewed** and **presented** to **stakeholders**. The **next sprint** cycle then begins.

Teamwork in Scrum

- The **'Scrum master'** is a facilitator who arranges daily meetings, tracks work to be done, records decisions, measures progress and communicates with customers and management outside of the team.
- The **whole team attends short daily meetings** where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.
 - This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

Scrum benefits

- The product is broken down into a set of **manageable and understandable chunks**.
- **Unstable requirements** do not hold up progress.
- The **whole team have visibility** of everything and consequently team communication is improved.
- **Customers see on-time delivery** of increments and gain feedback on how the product works.
- **Trust between customers and developers is established** and a positive culture is created in which everyone expects the project to succeed.

Scaling agile methods

- Agile methods have proved to be successful for **small and medium sized** projects that can be developed by a small **co-located team**.
- The need for faster delivery of software (suits the customer needs)
- It is sometimes argued that the success of these methods comes because of **improved communications** which is possible when everyone is working together.
- Scaling up agile methods involves changing these to cope with larger, longer projects where there are multiple development teams, perhaps working in different locations.

Large systems development

- Large systems are usually collections of **separate, communicating systems**, where separate teams develop each system. Frequently, these teams are working in different places, sometimes in different time zones.
- Large systems are '**brownfield systems**', that is they include and **interact** with a number of **existing systems**. Many of the system **requirements are concerned with this interaction** and so don't really lend themselves to **flexibility and incremental** development.

Large systems development

- Where several systems are **integrated** to create a system, a significant **fraction of the development** is concerned with system **configuration** rather than original **code development**.
- Large systems and their development processes are often constrained by **external rules and regulations** limiting the way that they can be developed.

Large system development

- Large systems have a **long procurement and development time**. It is difficult to maintain **coherent teams** who know about the system over that period as, unavoidably, people move on to other jobs and projects.
- Large systems usually have a **diverse set of stakeholders**. It is practically impossible to involve all of these different stakeholders in the development process.

Scaling out and scaling up

- 'Scaling up' is concerned with using agile methods for developing large software systems that **cannot be developed by a small team**.
- 'Scaling out' is concerned with how agile methods can be **introduced across a large** organization with many years of software development experience.
- When scaling agile methods it is essential to maintain agile fundamentals
 - Flexible planning, frequent system releases, continuous integration, test-driven development and good team communications.

Scaling up to large systems

- For large systems development, it is not possible to **focus only on the code** of the system. You need to do more **up-front design** (requirements fixing bugs) and **system documentation**
- **Cross-team communication** mechanisms have to be designed and used. This should involve regular phone and video conferences between team members and frequent, short electronic meetings where teams update each other on progress.
- **Continuous integration**, where the whole system is built every time any developer checks in a change, is **practically impossible**. **However, it is essential** to maintain frequent system builds and regular releases of the system.

Chapter 3 Agile software development

13

Scaling out to large companies Difficulties

- Project managers who do not have experience of agile methods may be **unwilling to accept** the risk of a **new approach**.
- Large organizations often have **quality procedures** and **standards** that all projects are expected to follow and, because of their **bureaucratic nature**, these are likely to be incompatible with agile methods.

Chapter 3 Agile software development

14

Scaling out to large companies Difficulties

- Agile methods seem to work best when team members have a **relatively high skill** level. **However**, within large organizations, there are likely to be a **wide range of skills and abilities**.
- There may be **cultural resistance** to agile methods, especially in those organizations that have a long history of using conventional systems engineering processes.

Chapter 3 Agile software development

15

Key points

- A particular strength of extreme programming is the development of automated tests before a program feature is created. All tests must successfully execute when an increment is integrated into a system.
- The Scrum method is an agile method that provides a project management framework. It is centred round a set of sprints, which are fixed time periods when a system increment is developed.
- Scaling agile methods for large systems is difficult. Large systems need up-front design and some documentation.

Chapter 3 Agile software development

16