Assignment for Lecture 3: Agile Software Development

LAI Hui Shan
m5281022

1. Describe the values advocated by the Agile Manifesto.

Ans:

Individuals and interactions > processes and tools
Working software > comprehensive documentation
Customer collaboration > contract negotiation
Responding to change > following a plan

2. Explain how the principles underlying agile methods lead to the accelerated development and deployment of software.

Ans:

The principles of agile methods:

Principle	Description
Customer involvemen	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

3. When would you recommend against the use of an agile method for developing a software system?

Ans:

Agile methods may not be suitable for large systems, projects with strict regulatory requirements, long-lifetime systems needing extensive documentation, and distributed or outsourced teams. In these cases, a plan-driven or hybrid approach may work better.

4. Describe the methodology of test-first development in XP.

Ans:

In Test-First Development, tests are written before the code, which clarifies the requirements to be implemented. Tests are written as programs, not data, so they can be executed automatically. This allows running tests as the code is being written and discovering problems during development.

5. Explain why test-first development helps the programmer to develop a better understanding of the system requirements.

Ans:

In Test-First Development, writing tests before the code clarifies the requirements that need to be implemented. By writing tests as executable programs, developers can validate requirements continuously as they code, helping them discover issues early and ensuring requirements are correctly met during development.

6. Suggest reasons why the productivity rate of programmers working as a pair might be more than half that of two programmers working individually.

Ans:

Advantages of pair programming:

1. It supports the idea of collective ownership and responsibility for the system.
    - Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.
2. It acts as an informal review process because each line of code is looked at by at least two people.
3. It helps support refactoring, which is a process of software improvement.
    - Where pair programming and collective ownership are used, others benefit immediately from the refactoring so they are likely to support the process.

7. Explain what Continuous Integration (CI) is and how it works.

Ans:

Continuous integration:
As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass. 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.

8. Describe the difficulties and possible approaches to solving them when applying Agile to the development of a Large System.

Ans:

Large system development:

1. 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.
2. 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.
3. 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.
4. Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed.
5. 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, inevitably, people move on to other jobs and projects.
6. 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 up to large systems:

1. 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 and system documentation.
2. 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.
3. 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.