**Agile Methods and Their Importance**

*-Alapan Chaudhuri, 2019111023*

**Agile Software Development** comprises various approaches to software development under which requirements and solutions evolve through the collaborative effort of self-organizing and cross functional teams and their end users. It advocates adaptive planning, evolutionary development, early delivery, and continual improvement, and it encourages rapid and flexible response to change.

Agile Software Development is based on the following 12 principles:

* + *Customer satisfaction by early and continuous delivery of valuable software.*
  + *Welcome changing requirements, even in late development.*
  + *Deliver working software frequently (weeks rather than months)*
  + *Close, daily cooperation between business people and developers*
  + *Projects are built around motivated individuals, who should be trusted*
  + *Face-to-face conversation is the best form of communication (co-location)*
  + *Working software is the primary measure of progress*
  + *Sustainable development, able to maintain a constant pace*
  + *Continuous attention to technical excellence and good design*
  + *Simplicity—the art of maximising the amount of work not done—is essential*
  + *Best architectures, requirements, and designs emerge from self-organising teams*
  + *Regularly, the team reflects on how to become more effective, and adjusts accordingly*

**Forms of Agile Software Development :**

Forms of Agile Software Development and Management Method are as follows:

* Agile Scrum Methodology
* Lean Software Development
* Kanban
* Extreme Programming (XP)
* Crystal
* Dynamic Systems Development Method (DSDM)
* Feature Driven Development (FDD)

**Extreme Programming:**

Here, we shall be studying one form Agile Software Development, namely Extreme Programming, in detail.

**Extreme programming** (**XP**) is a [software development methodology](https://en.wikipedia.org/wiki/Software_development_methodology) which is intended to improve software quality and responsiveness to changing customer requirements. It is also known as XP, and advocates frequent "releases" in short development cycles, which is intended to improve productivity and introduce checkpoints at which new customer requirements can be adopted.

The principles that form the basis of XP are based on the values just described and are intended to foster decisions in a system development project. The principles are intended to be more concrete than the values and more easily translated to guidance in a practical situation.

Extreme programming sees feedback as most useful if it is done frequently and promptly. It stresses that minimal delay between an action and its feedback is critical to learning and making changes. Unlike traditional system development methods, contact with the customer occurs in more frequent iterations. The customer has clear insight into the system that is being developed and can give feedback and steer the development as needed. With frequent feedback from the customer, a mistaken design decision made by the developer will be noticed and corrected quickly, before the developer spends much time implementing it.

Unit tests contribute to the rapid feedback principle. When writing code, running the unit test provides direct feedback as to how the system reacts to the changes made. This includes running not only the unit tests that test the developer's code but running in addition all unit tests against all the software, using an automated process that can be initiated by a single command. That way, if the developer's changes cause a failure in some other portion of the system that the developer knows little or nothing about, the automated all-unit-test suite will reveal the failure immediately, alerting the developer of the incompatibility of their change with other parts of the system, and the necessity of removing or modifying their change. Under traditional development practices, the absence of an automated, comprehensive unit-test suite meant that such a code change, assumed harmless by the developer, would have been left in place, appearing only during integration testing – or worse, only in production; and determining which code change caused the problem, among all the changes made by all the developers during the weeks or even months previous to integration testing, was a formidable task.

This is about treating every problem as if its solution were "extremely simple". Traditional system development methods say to plan for the future and to code for reusability. Extreme programming rejects these ideas.

The advocates of extreme programming say that making big changes all at once does not work. Extreme programming applies incremental changes: for example, a system might have small releases every three weeks. When many little steps are made, the customer has more control over the development process and the system that is being developed.

The principle of embracing change is about not working against changes but embracing them. For instance, if at one of the iterative meetings it appears that the customer's requirements have changed dramatically, programmers are to embrace this and plan the new requirements for the next iteration.