What would we consider as we were developing a new methodology?

- Software development is predominantly a design activity
- Characteristics of individuals are the first-order influence on project activities
- Modern software development can't rely on individuals – it requires effective teams
- Customers are unlikely to know what they want in detail before they start
- Customers need to be able to change requirements without unreasonable penalties
- The process needs to be flexible
- Responsibility should be aligned with authority

The Agile Alliance

The methodologies falling under the "agile" umbrella all address these issues in slightly different ways, however the agile methodologies do have common underlying values and principles. The Agile Alliance expressed the values in the Agile Manifesto[7]:

"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."

They also expressed the principles behind the manifesto[8]:

- 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 time-scale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support

More About Extreme Programming

Extreme Programming is an agile, adaptive software develop methodology with a well-defined set of values and core practices.

- **Agile** : XP is consistent with the values and principles of the Agile Alliance
- **Adaptive** : XP's documented practices are only a starting point – XP teams adapt the process to improve their results
- **Values**: Since XP is adaptive, the details vary from one XP project to another. But the underlying values stay the same. If the values change, then the process is no longer XP
- **Core Practices** : XP specifies a set of mutually-supporting practices that encourage collaboration and reduce the cost of change XP does not have complex rules, and it does not try to specify exactly how to respond to every possible situation the team will encounter, instead, XP tries to be "barely sufficient[11]". Dee W.

Hock, founder of the Visa organization, made these points[12]:

- Simple, clear purpose and principles give rise to complex, intelligent behaviour
- Complex rules and regulations give rise to simple, stupid behaviour

XP specifies the rules – the values and the practices – and lets the team figure out the detailed
behaviour.

Extreme Programming Values and Principles

The XP values are:

- Communication
- Simplicity
- Feedback
- Courage

An XP project relies on these four values. If your organization or team doesn't truly share these values, then an XP project will fail. Of course, most of those values are motherhood-and-apple pie –it would be hard to find an organization that said that it didn't believe in them. XP tries to remove some of the vagueness from these values by describing principles that embody the values.

- Open, honest communication
- Quality work
- Rapid feedback at all le
- Embrace change
- Play to win
- Concrete experiments
- Small initial investment
- Incremental change
- Accepted responsibility
- Honest measurement
- Travel light
- Teach learning
- Local adaptation

Open, honest communication

Most software development problems can be traced back to communication failures – someone didn't talk to the right person, or they talked to the right person at the wrong time, or they didn't talk to anyone all. XP encourages communication by using practices that can't be done without communicating openly and honestly. Very few projects fail because good news was suppressed –
it's how the bad news is treated that makes the difference. This means that programmers need to be able to explain the consequences of decisions, to express their fears, and to admit when they've simply screwed up. Of course, we use practices that reduce the chances of things going wrong, but no matter what we do, on every project something **will** go wrong.

Quality work

Let's stop for a moment and consider a simple model for controlling software development projects.
This model says that although there are lots of details and variations, fundamentally there are only four things that a manager can manipulate to control a software development – cost, time, scope and quality. When a project is running successfully, most project managers will leave the control variables just as they are – it's what happens when a project starts to fail by some measure that's interesting.
A project manager might choose to correct a failing project by allowing for increased costs. The extra money might be spent on new hardware or, more often, extra staff time. They might choose to allow the project to run for more time with fewer resources (since using the same resources for longer would increase costs as well). Another way to make the project successful might be to leave the time and costs the same, but get the customer to agree to cutting some features from the requirements. And finally, the project manager might choose to leave the costs, time and scope the same, but reduce the quality of the resulting application. In this context it doesn't really matter what definition of quality the project is using, only that reducing quality reduces the required development effort.
Unfortunately, these control variables aren't independent, and the relationships between them aren't linear. For example, we've already mentioned that extending time while leaving resources the same usually also increases costs. Increasing costs late in a project may have very little effect, since making the new resources productive may require considerable effort.

Extreme Programming Practices in brief

XP is an adaptive method, so although all XP teams need to consistently embrace the values and principles, the details of the actual day-to-day operation of an XP project vary from team to team. A reasonable question then is “where do I start?”. XP provides a set of daily practices that, used together, have been demonstrated to efficiently produce high quality software. These practices are:

- Whole Team
- Planning Game
- Customer Tests
- Small Releases
- Simple Designs
- Pair Programming
- Test-Driven Development
- Design Improvement

For More info **contact@alphasynopsys.com**