OCTOBER

CHITKARA UNIVERSITY

19 | 19·10·2016 WEDNESDAY Tourist - national to international

Agile

Agile SDLC model is a combination of iterative & incremental process models with focus on process adaptability & customer satisfaction by rapid delivery of working software product.

20 | 20·10·2016 THURSDAY Agile methods break the product into small incremental builds. These builds are provided in iteration. ~~each iteration~~. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional team working simultaneously on various area like planning, requirement analysis, design, coding, unit testing & acceptance testing.

October 2016

Su	Mo	Tu	We	Th	Fr	Sa
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

Life is not measured by the breaths you take, but by its breathtaking moments.

21 · 10 · 2016 FRIDAY 21

At the end of the iteration working product is displayed to the customer & important stakeholder.

Agile model believes that every project need to be handled differently & the existing method need to be tailored to best suit the project requirement. In agile the task are divided to time boxes (small time frames) to deliver specific features for a release.

22 · 10 · 2016 SATURDAY 22

Iterative approach is taken & working software build is delivered after each iteration.

SUNDAY 23

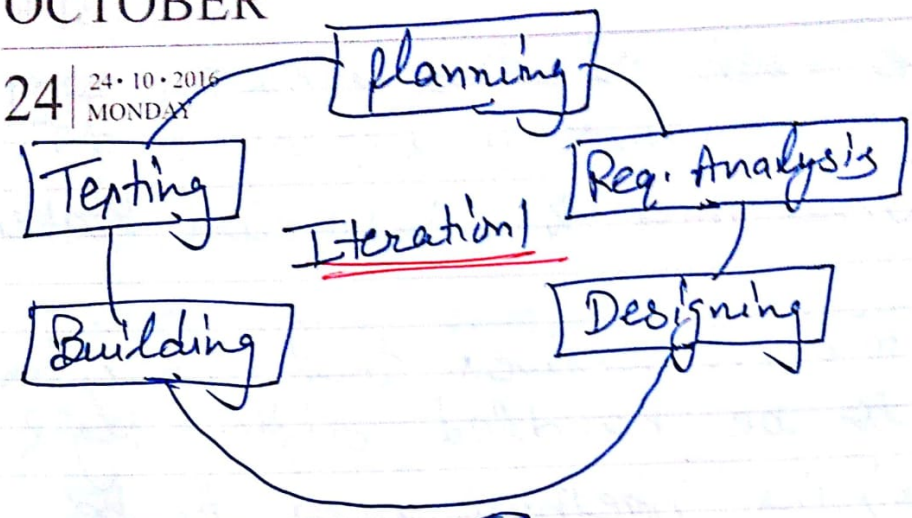
November 2016

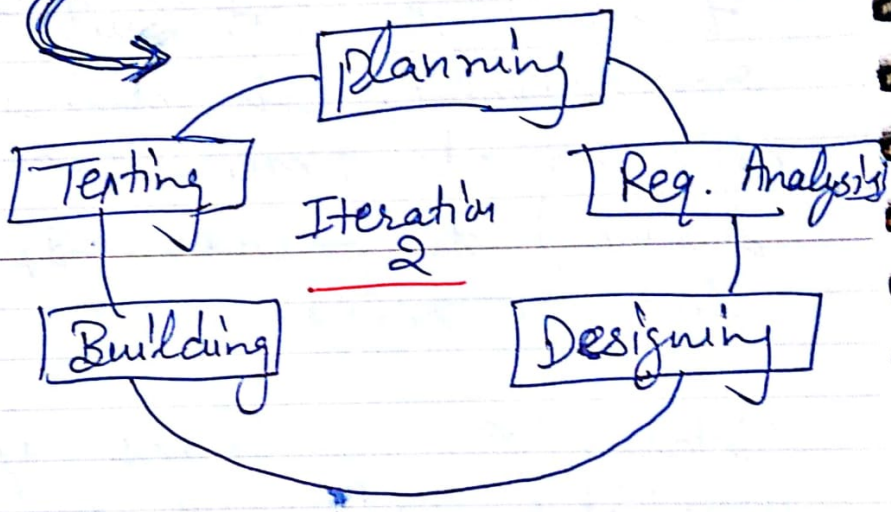
Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

CHITKARA UNIVERSITY

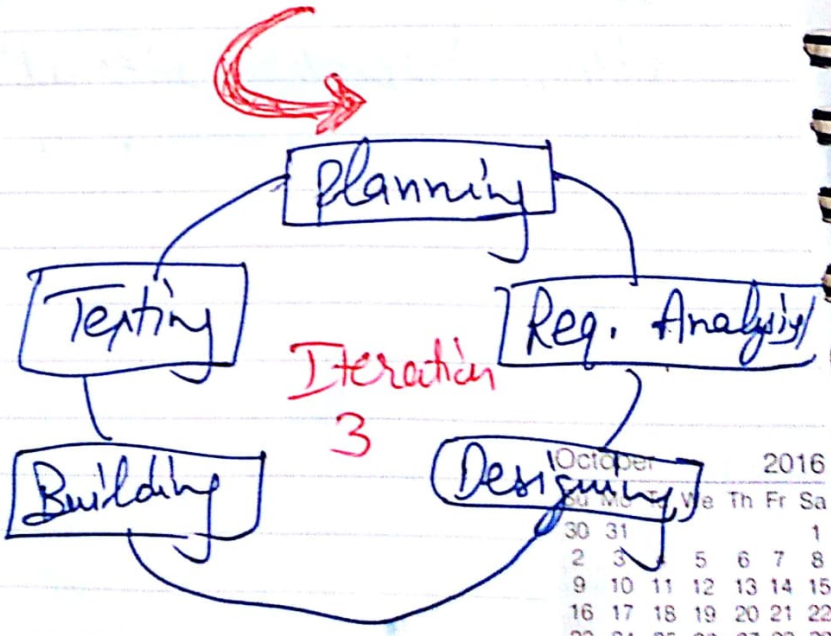
OCTOBER

24 | 24 • 10 • 2016 MONDAY





25 | 25 • 10 • 2016 TUESDAY



October 2016

Su	Mo	Tu	We	Th	Fr	Sa
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

Agile principle

Individuals & interaction!- In agile development, self organization & motivation are important, as are interaction like co-location & pair programming. (over process & tools)

Working Software!- Demo working software is considered the best means of communication with the customer to understand their requirement instead of just depending on documentation. (over comprehensive documentation)

Customer Collaboration!- As the requirements can not be gathered completely in the beginning of the project due to various factor, continuous customer interaction is very important to get proper product requirements. (over contract negotiation)

CHITKARA UNIVERSITY

28 | 28·10·2016 FRIDAY

Responding to change :- agile development is focused on quick response to change & continuous development.

over following a plan.

- There are many agile processes SCRUM, crystal, Behavior-Driven Development (BDD), Test Driven Development (TDD), feature-Driven Development (FDD), Adaptive Software Development (ADP), Extreme programming and many more.

29 | 29·10·2016 SATURDAY

Extreme programming :-

planning :- 1) ~~User stories~~ the planning activity begin with the creation of set of stories (user stories) that describe required features & functionality for software to be build.

30 SUNDAY

October 2016

Su	Mo	Tu	We	Th	Fr	Sa
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

CHITKARA UNIVERSITY

OCT/NOV

Index Card! — Each story is written by customer & is placed on an index card.

↳ Assign priority / number.

↳ Members of the XP team then assess each story & assign a cost (measured in development weeks)

If stories require more than three development weeks the customer is asked to split the story into smaller stories & assignment of value & cost occur again

31 MONDAY

01 · 11 · 2016 TUESDAY 01

Three way of ordering stories

1) all stories will be implemented immediately (with in a few weeks)
2) the stories with highest value will be moved up in the schedule & implemented first
3) The stories with highest risk will be implemented first.

November 2016

Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5
7	8	9	10	11	12
14	15	16	17	18	19
21	22	23	24	25	26
28	29	30			

All that we are is the result of our thoughts.

02 | 02·11·2016 WEDNESDAY

Computer project velocity :-

project velocity is the number of customer stories implemented during the first release.

It is used to :-

1) help estimate delivery dates & schedule for subsequent releases.

2) Determine whether an over-commitment has been ~~done~~ made for all stories across the entire development project.

03 | 03·11·2016 THURSDAY

Design :- Simple design.

XP CRC cards :- XP uses CRC cards (class-responsibility-collaborators) which is a collection of standard Index cards that represent classes.

CHITKARA UNIVERSITY

NOVEMBER

04 FRIDAY

Spike Solution :- If a difficult design problem is encountered as part of the design of a story, XP recommends the immediate creation of an operational prototype of that portion of the design. called spike solution.

~~Refactoring :-~~

05 SATURDAY

Coding :- 1) pair programming
2) continous integration - to avoid compatibility & interfacing problems & provide smoke Testing environment that help to uncover error early.
3) Refactoring :- on next page.

SUNDAY 06

4) **Testing :-** 1) Unit Testing
2) Regression Testing - whenever code is modified (Refactoring)
3) Acceptance Testing - Customer Testing

December 2016

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Everything that is done in the world is done by hope.

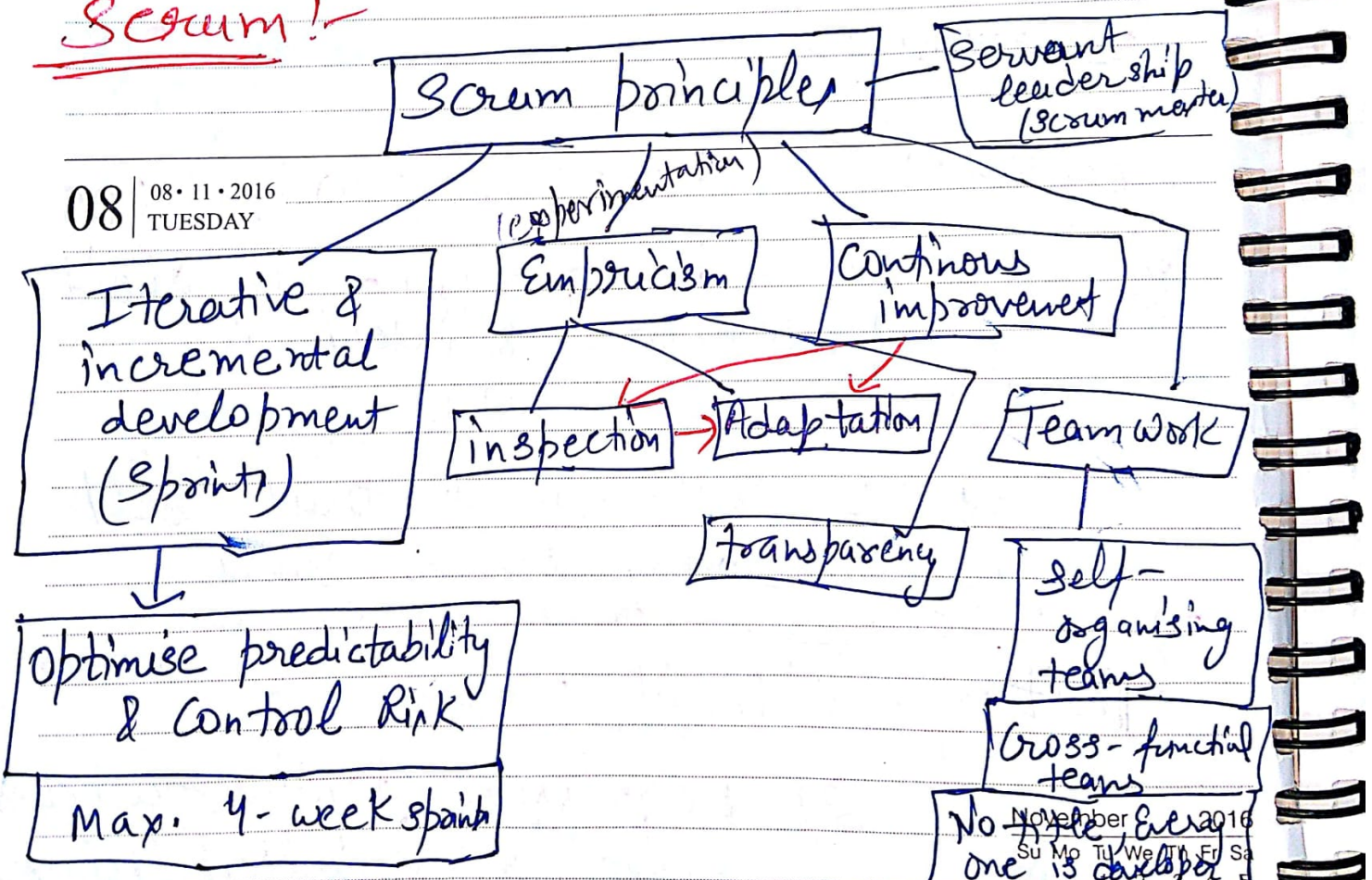
CHITKARA UNIVERSITY

NOVEMBER

07 | 07 · 11 · 2016 MONDAY

Refactoring!- Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves the internal structure. It minimizes the chances of introducing bugs.

Scrum!-



08 | 08 · 11 · 2016 TUESDAY

November 2016

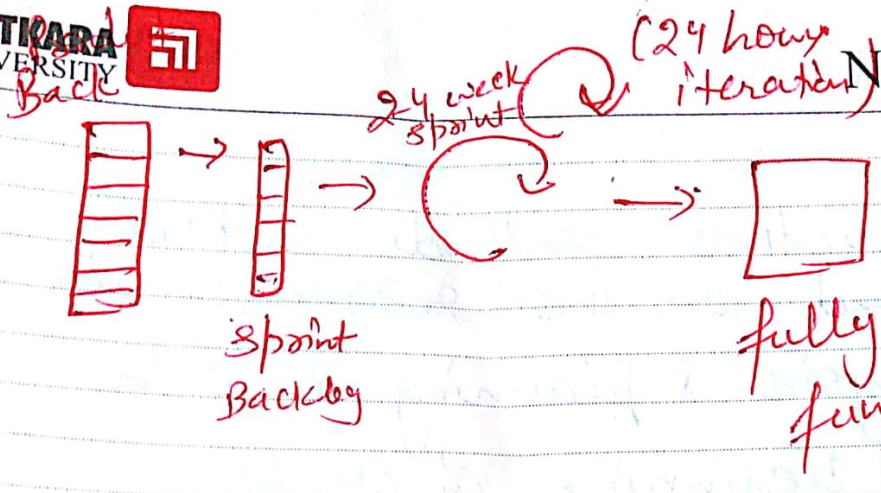
Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

You can't change the direction of the wind but can adjust the sail.

CHITKARA UNIVERSITY

NOVEMBER

14 · 11 · 2016 MONDAY | 14



Agile Vs Traditional SDLC Models

Agile is based on the adaptive software development methods where as the traditional SDLC models like waterfall model is based on predictive approach.

15 · 11 · 2016 TUESDAY | 15

In traditional SDLC models usually work with detailed planning & have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle.

December 2016

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17

16 | 16 · 11 · 2016 WEDNESDAY

↳ Predictive methods entirely depend on the requirements analysis & planning done in the beginning of cycle.

↳ Any changes to be incorporated go through a strict change control management & prioritization.

17 | 17 · 11 · 2016 THURSDAY

Agile User adaptive approach where there is no detailed planning & there is clarity on future task only in respect of what features need to be developed.

↳ There is ~~dont~~ feature driven development & the team adapts to the changing product requirements dynamically.

18-11-2016 FRIDAY | 18

↳ The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

↳ customer interaction is the backbone of Agile methodology & open communication with minimum documentation are the typical features of Agile development environment.

19-11-2016 SATURDAY | 19

↳ The Agile teams work in close collaboration with each other & are most often located in the in same geographical location.

SUNDAY 20

December 2016

Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

NOVEMBER

CHITKARA UNIVERSITY

25 | 25 · 11 · 2016 FRIDAY

Agile model Pros & Cons

Pros

↳ It is very realistic approach to Software development

↳ promotes teamwork & cross training.

↳ functionality can be developed rapidly & demonstrated

Cons

↳ Not suitable for handling complex dependencies.

↳ More risk of sustainability, maintainability & extensibility.

↳ An overall plan, an agile leader & agile PM practice is a must without which it will not work.

26 | 26 · 11 · 2016 SATURDAY

27 SUNDAY

Resource Requirement are minimum

November 2016

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

When things get rough, remember it's the rubbing that brings out the shine.

↳ Suitable for fixed or changing requisets

↳ Delivery early partial working solution.

↳ Good model for environments that change steadily.

↳ Minimum rules, documentation easily employed.

↳ Enables concurrent development & delivery within an overall planned context.

↳ little or no planning required

↳ Easy to manage

28 11 2016 MONDAY 28

① Strict delivery management dictates the scope, functionality to be delivered, & adjustment to meet the requirement

② Depends heavily on customer interaction, so if customer is not clear, teams can be driven in the wrong direction.

29 11 2016 TUESDAY 29

③ There is very high individual dependency, since there is minimum documentation generated.

NOV/DEC

CHITKARA UNIVERSITY

30 | 30 • 11 • 2016 WEDNESDAY

(2) Transfer of technology to new team members may be quite challenging due to lack of Documentat

01 | 01 • 12 • 2016 THURSDAY

Jamie: Huh? You mean that marketing will work on the project team with us?

Doug (nodding): They're a stakeholder, aren't they?

Jamie: Jeez . . . they'll be requesting changes every five minutes.

Vinod: Not necessarily. My friend said that there are ways to "embrace" changes during an XP project.

Doug: So you guys think we should use XP?

Jamie: It's definitely worth considering.

Doug I agree. And even if we choose an incremental model as our approach, there's no reason why we can't incorporate much of what XP has to offer.

Vinod: Doug, before you said "some good, some bad." What was the "bad"?

Doug: [illegible] analysis and design . . . sort of says that writing code is where the action is.

(The team members look at one another and smile.)

Doug: So you agree with the XP approach?

Jamie (speaking for both): Writing code is what we do, Boss!

Doug (laughing): True, but I'd like to see you spend a little less time coding and then re-coding and a little more time analyzing what has to be done and designing a solution that works.

Vinod: Maybe we can have it both ways, agility with a little discipline.

Doug: I think we can, Vinod. In fact, I'm sure of it.

4.3.2 Adaptive Software Development (ASD)

Adaptive Software Development (ASD) has been proposed by Jim Highsmith [HIG00] as a technique for building complex software and systems. The philosophical underpinnings of ASD focus on human collaboration and team self-organization. Highsmith [HIG98] discusses this when he writes:

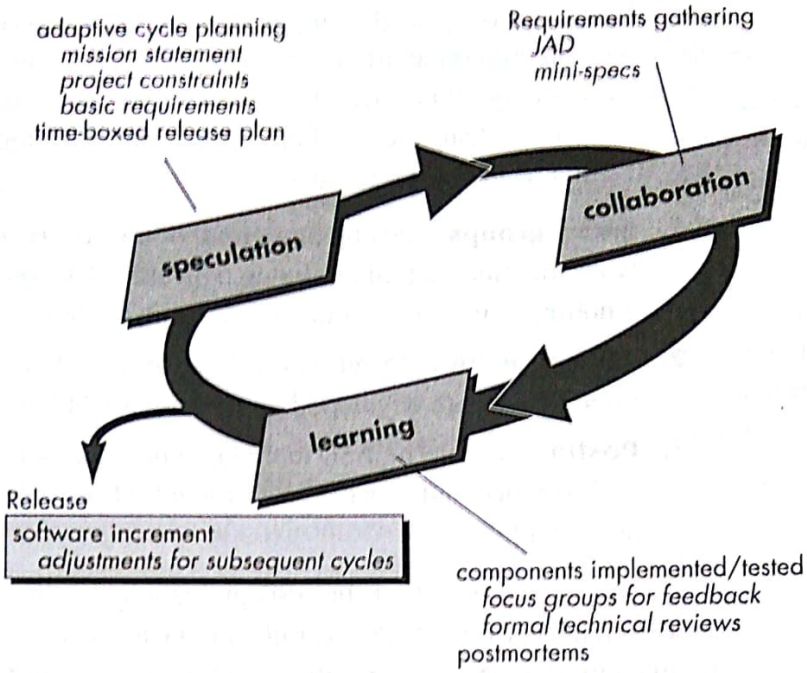
> Self-organization is a property of complex adaptive systems similar to a collective "aha," that moment of creative energy when the solution to some nagging problem emerges. Self-organization arises when individual, independent agents (cells in a body, species in an ecosystem, developers in a feature team) cooperate [collaborate] to create emergent outcomes. An emergent outcome is a property beyond the capability of any individual agent. For example, individual neurons in the brain do not possess consciousness, but collectively the property of consciousness emerges. We tend to view this phenomena of collective emergence as accidental, or at least unruly and undependable. The study of self-organization is proving that view to be wrong.

ebRef
resources for
n be found at
.adaptivesd.

Highsmith argues that an agile, adaptive development approach based on collaboration is "as much a source of *order* in our complex interactions as discipline and engineering." He defines an ASD "life cycle" (Figure 4.2) that incorporates three phases: speculation, collaboration, and learning.

Speculation. During *speculation*, the project is initiated and *adaptive cycle planning* is conducted. Adaptive cycle planning uses project initiation information—the customer's mission statement, project constraints (e.g., delivery dates or user descrip-

FIGURE 4.2 Adaptive software development



What are the characteristics of ASD adaptive cycles?

tions), and basic requirements—to define the set of release cycles (software increments) that will be required for the project.[9]

Collaboration. Motivated people work together in a way that multiplies their talent and creative output beyond their absolute numbers. This collaborative approach is a recurring theme in all agile methods. But collaboration is not easy. It is not simply communication, although communication is a part of it. It is not only a matter of teamwork, although a "jelled" team (Chapter 21) is essential for real collaboration to occur. It is not a rejection of individualism, because individual creativity plays an important role in collaborative thinking. It is, above all, a matter of trust. People working together must trust one another to (1) criticize without animosity; (2) assist without resentment; (3) work as hard or harder as they do; (4) have the skill set to contribute to the work at hand; and (5) communicate problems or concerns in a way that leads to effective action.

ADVICE

Effective collaboration with your customer will only occur if you jettison any "us and them" attitudes.

> "I like to listen. I have learned a great deal from listening carefully. Most people never listen."
>
> **Ernest Hemingway**

9 Note that the adaptive cycle plan can and probably will be adapted to changing project and business conditions.

Learning. As members of an ASD team begin to develop the components that are part of an adaptive cycle, the emphasis is on learning as much as it is on progress toward a completed cycle. In fact, Highsmith [HIG00] argues that software developers often overestimate their own understanding (of the technology, the process, and the project) and that learning will help them to improve their level of real understanding. ASD teams learn in three ways:

1. **Focus groups.** The customer and/or end-users provide feedback on software increments that are being delivered. This provides a direct indication of whether or not the product is satisfying business needs.
2. **Formal technical reviews.** ASD team members review the software components that are developed, improving quality and learning as they proceed.
3. **Postmortems.** The ASD team becomes introspective, addressing its own performance and process (with the intent of learning and then improving its approach).

It is important to note that the ASD philosophy has merit regardless of the process model that is used. ASD's overall emphasis on the dynamics of self-organizing teams, interpersonal collaboration, and individual and team learning yield software project teams that have a much higher likelihood of success.

4.3.3 Dynamic Systems Development Method (DSDM)

WebRef
Useful resources for DSDM can be found at **www.dsdm.org.**

The *Dynamic Systems Development Method* (DSDM) [STA97] is an agile software development approach that "provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment" [CCS02]. Similar in some respects the RAD process discussed in Chapter 3, DSDM suggests a philosophy that is borrowed from a modified version of the Pareto principle. In this case, 80 percent of an application can be delivered in 20 percent of the time it would take to deliver the complete (100 percent) application.

Like XP and ASD, DSDM suggests an iterative software process. However, the DSDM approach to each iteration follows the 80 percent rule. That is, only enough work is required for each increment to facilitate movement to the next increment. The remaining detail can be completed later when more business requirements are known or changes have been requested and accommodated.

The DSDM Consortium (www.dsdm.org) is a worldwide group of member companies that collectively take on the role of "keeper" of the method. The consortium has defined an agile process model, called the *DSDM life cycle*. The DSDM life cycle defines three different iterative cycles, preceded by two additional life cycle activities:

WebRef
A useful overview of DSDM can be found at **www.cs3inc.com/DSDM.htm.**

Feasibility study—establishes the basic business requirements and constraints associated with the application to be built and then assesses whether the application is a viable candidate for the DSDM process.

must be considered.

4.1 What Is Agility?

Just what is agility in the context of software engineering work? Ivar Jacobson [JAC02] provides a useful discussion:

> *Agility* has become today's buzzword when describing a modern software process. Everyone is agile. An agile team is a nimble team able to appropriately respond to changes. Change is what software development is very much about. Changes in the software being built, changes to the team members, changes because of new technology, changes of all kinds that may have an impact on the product they build or the project that creates the product. Support for changes should be built-in everything we do in software, something we embrace because it is the heart and soul of software. An agile team recognizes that software is developed by individuals working in teams and that the skills of these people, their ability to collaborate is at the core for the success of the project.

In Jacobson's view, the pervasiveness of change is the primary driver for agility. Software engineers must be quick on their feet if they are to accommodate the rapid changes that Jacobson describes.

> Agility is dynamic, content specific, aggressively change embracing, and growth oriented.
>
> **Steven Goldman et al.**

But agility is more than an effective response to change. It also encompasses the philosophy espoused in the manifesto noted at the beginning of this chapter. It encourages team structures and attitudes that make communication (among team members, between technologists and business people, between software engineers and their managers) more facile. It emphasizes rapid delivery of operational software and de-emphasizes the importance of intermediate work products (not always a good thing); it adopts the customer as a part of the development team and works to eliminate the "us and them" attitude that continues to pervade many software projects; it recognizes that planning in an uncertain world has its limits and that a project plan must be flexible.

VICE

ke the f assuming y gives you hack out A process is and discipline l.

The Agile Alliance [AGI03] defines 12 principles for those who want to achieve agility:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Agility can be applied to any software process. However, to accomplish this, it is essential that the process be designed in a way that allows the project team to adapt tasks and to streamline them, conduct planning in a way that understands the fluidity of an agile development approach, eliminate all but the most essential work products and keep them lean, and emphasize an incremental delivery strategy that gets working software to the customer as rapidly as feasible for the product type and operational environment.

4.2 What Is an Agile Process?

Any *agile software process* is characterized in a manner that addresses three key assumptions [FOW02] about the majority of software projects:

1. It is difficult to predict in advance which software requirements will persist and which will change. It is equally difficult to predict how customer priorities will change as a project proceeds.
2. For many types of software, design and construction are interleaved. That is, both activities should be performed in tandem so that design models are proven as they are created. It is difficult to predict how much design is necessary before construction is used to prove the design.
3. Analysis, design, construction, and testing are not as predictable (from a planning point of view) as we might like.

much that can be gained by
ing to be gained by denigrating either approach.

The interested reader should see [HIG01], [HIG02a], and [DEM02] for an entertaining summary of the important technical and political issues.

4.2.2 Human Factors

Proponents of agile software development take great pains to emphasize the importance of "people factors" in successful agile development. As Cockburn and Highsmith [COC01] state, "Agile development focuses on the talents and skills of individuals, molding the process to specific people and teams." The key point in this statement is that the *process molds to the needs of the people and team*, not the other way around.[2]

> "What counts as barely sufficient for one team is either overly sufficient or insufficient for another."
>
> **Alistair Cockburn**

If members of the software team are to drive the characteristics of the process that is applied to build software, a number of key traits must exist among the people on an agile team and the team itself:

What key traits must exist among the people on an effective software team?

Competence. In an agile development (as well as conventional software engineering) context, "competence" encompasses innate talent, specific software related skills, and overall knowledge of the process that the team has chosen to apply. Skill and knowledge of process can and should be taught to all people who serve as agile team members.

Common focus. Although members of the agile team may perform different tasks and bring different skills to the project, all should be focused on one goal—to deliver a working software increment to the customer within the time promised. To achieve this goal, the team will also focus on continual adaptations (small and large) that will make the process fit the needs of the team.

Collaboration. Software engineering (regardless of process) is about assessing, analyzing, and using information that is communicated to the software team; creating information that will help the customer and others understand the work of the team; and building information (computer software and relevant databases) that provides business value for the customer. To accomplish these tasks, team members must collaborate—with one another, with the customer, and with business managers.

Decision-making ability. Any good software team (including agile teams) must be allowed the freedom to control its own destiny. This implies that the

---

2 Most successful software engineering organizations recognize this reality regardless of the process model they choose.

team is given autonomy—decision-making authority for both technical and project issues.

Fuzzy problem-solving ability. Software managers should recognize that the agile team will continually have to deal with ambiguity and will continually be buffeted by change. In some cases, the team must accept the fact that the problem they are solving today may not be the problem that needs to be solved tomorrow. However, lessons learned from any problem solving activity (including those that solve the wrong problem) may be of benefit to the team later in the project.

Mutual trust and respect. The agile team must become what DeMarco and Lister [DEM98] call a "jelled" team (see Chapter 21). A jelled team exhibits the trust and respect that are necessary to make them "so strongly knit that the whole is greater than the sum of the parts" [DEM98].

KEY POINT

A self-organizing team is in control of the work it performs. The team makes its own commitments and defines plans to achieve them.

Self-organization. In the context of agile development, *self-organization* implies three things: (1) the agile team organizes itself for the work to be done; (2) the team organizes the process to best accommodate its local environment; (3) the team organizes the work schedule to best achieve delivery of the software increment. Self-organization has a number of technical benefits, but more importantly it serves to improve collaboration and boost team morale. In essence, the team serves as its own management. Ken Schwaber [SCH02] addresses these issues when he writes: "The team selects how much work it believes it can perform within the iteration, and the team commits to the work. Nothing demotivates a team as much as someone else making commitments for it. Nothing motivates a team as much as accepting the responsibility for fulfilling commitments that it made itself."

4.3 Agile Process Models

The history of software engineering is littered with dozens of obsolete process descriptions and methodologies, modeling methods and notations, tools, and technology. Each flared in notoriety and was then eclipsed by something new and (purportedly) better. With the introduction of a wide array of agile process models—each contending for acceptance within the software development community—the agile movement is following the same historical path.[3]

> "Our profession goes through methodologies like a 14-year-old goes through clothing."
>
> **Stephen Hawrysh and Jim Ruprecht**

3 This is not a bad thing. Before one or more models or methods are accepted as a de facto standard, all must contend for the hearts and minds of software engineers. The "winners" evolve into best practice while the "losers" either disappear or merge with the winning models.