D-Cube (Decisive Dynamic Development)

Transparent · Feedback · Motivate · Reduce Waste

Chandra R. Munagavalasa, PSM I, SSGB, PMI-ACP, PMP

Abstract

Many popular agile approaches that exist today have strict rules and/or requirements. D-cube is a very simple and highly flexible agile approach that can be tailored to fit to your needs.

Whether you empower the team or have a top down structure; whether you use story points to estimate the work or use a three-point estimate; whether the team is collocated or distributed; or whether you have a daily scrum or a status meeting, D-cube can be applied to any kind of software development project.

D-cube primarily emphasizes the need for motivated teams, transparency, feedback loops, and the reduction of waste.

Introduction

In this competitive world of software development, successful project deliveries are vital for the survival of any software development organization. Project failures not only result in financial loss, but also destroy customers' goodwill.

What defines a successful software project? Most of the time, the answer is one or more of the following:

1. Conforming to the requirements;
2. Delivering the software on or before schedule;
3. Completing the project within the allotted budget; and
4. High business value and return on investment (ROI).

Let me step away from software development for a moment.

I wanted to get a coffee from the coffee shop in the next 10 minutes and the coffee shop is located a mile away. I went to my car, only to find that it had a flat tire. So, I ran to the coffee shop as fast as I could to cover the mile in 10 minutes and bought the coffee.

I got the coffee in the allocated time frame, and cheaper than I had initially projected since I saved money on gas. The quality of the coffee is as I expected, and so it conformed to my requirements. Does this constitute a successful task? Can this approach be applied again if I want to get another coffee?

Coming back to our software development project, let me ask you few questions.

- Did you ever come across project teams having a daily status meeting with either their team lead or reporting manager taking notes and calling it a "daily scrum?"

- Did you ever come across team members answering more than the three scrum questions in the scrum meeting?
- Did you ever notice organizations saying they are “going lean” when they actually cut costs?
- How many times have you seen an “information radiator” showing progress on fancy, complex charts that do not actually show the necessary data?
- How many times have you seen organizations “listen” to customer feedback through surveys, voice of customer, and other means, only to view the data, but not really “understand” it, or worse yet, “act” on the feedback?

Many development team members and managers like to use the words scrum, lean, kaizen, servant-leader, etc. even though, in reality, they do quite the opposite of the actual definition of these words. Why? Maybe because of the pressure from upper management to follow a certain process, or maybe they lack knowledge and understanding, or maybe simply because these terms just sound good.

Several different agile approaches can be applied to software project development—Agile Modeling, Crystal Clear, Dynamic Systems Development Method (DSDM), Extreme Programming (XP), Scrum, Feature Driven Development (FDD), Kanban, Scrum ban, etc. You may argue that one approach is better than the other, or that a combination of methodologies is better than one, or maybe that a tailored agile process is better. A few of you may even argue that the traditional waterfall approach works better for your projects. If the current process works well for you, why change? Well, just because a particular process worked for you, it does not mean that it is the best process.

Whether you empower the team or have a top-down structure; whether you use story points to estimate the work or use a three-point estimate; whether the team is co-located or distributed; whether you do a daily scrum or a status meeting; I am introducing a new agile development approach that can be applied to any software development project, called D-cube (Decisive Dynamic Development).

You may be thinking, why do we need yet another agile approach when there are numerous other variations that already exist in the market? Many popular agile approaches that exist today have strict rules and/or requirements. D-cube is very simple and highly flexible, and can be tailored to fit to your needs. D-cube primarily emphasizes the need for motivated teams, transparency, stakeholder feedback, and the reduction of waste.

As in any agile development process, D-cube process steps are similar.

1. Gather requirements.
2. Prioritize requirements.
3. Select items to be developed for the iteration.
4. Build items.
5. Release incremental build to stakeholders.
6. Get stakeholders feedback and include it in the requirements.
7. Conduct a retrospective.
8. Go to step 2.

D-cube emphasizes that the process steps should be:

- Continuously motivate and train the project team.
- Be very transparent.
- Obtain continuous feedback from stakeholders.
- Continuously reduce waste when possible.

D-cube is an iterative and incremental software development approach. Motivated teams build the software, deliver the incremental builds to the stakeholders, and receive stakeholder feedback. All the valid feedback is then incorporated in the software product. D-cube continuously reduces waste while being as transparent as possible. The project team is continuously motivated and trained on the technology, as well as on the business requirements.

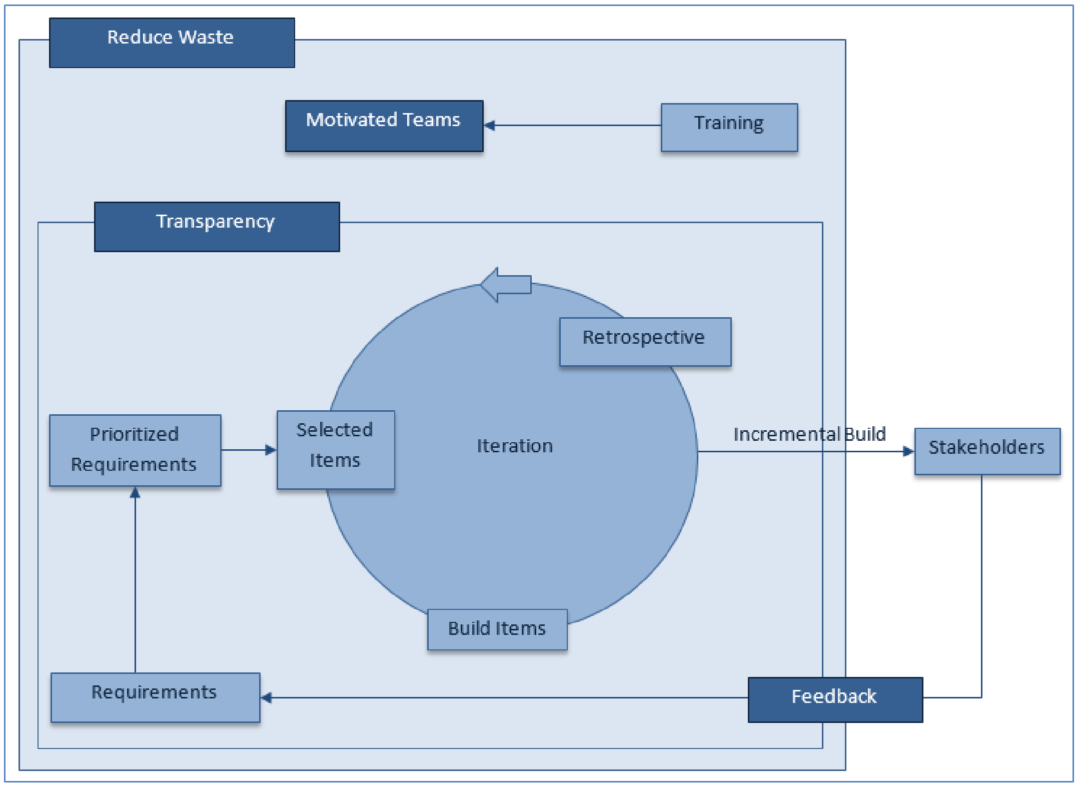

Figure 1: D-cube process steps.

You pick the rules. You pick the roles. And you follow the process steps with an emphasis on motivation, transparency, feedback, and the elimination of waste. This is D-cube.

1. Motivate Your Team

A motivated team is a very crucial to the successful completion of any software project. D-cube extremely emphasizes the importance of the motivated team.

Motivation is a highly-complex phenomenon and goes way beyond the basic payment of salary. There is no “one-solution-fits-all” in motivating individuals and teams. People react differently to motivating agents and the motivation may be based on the needs of the individual. Project managers need to research extensively and fully understand the motivating agents of the individuals on their team, as well as the team as a whole, and should continuously strive to motivate the project team and its individual members.

High motivation leads to high morale and will result in a high-quality end product. Motivated employees also tend to give their best and stay committed and loyal to the organization.

1.1 Motivation-Hygiene Theory

Frederick Herzberg's two-factor theory states that in the work place, there are certain factors—Herzberg called them motivators—that cause job satisfaction; while a completely different set of factors—called hygiene factors—cause dissatisfaction. According to this two-factor theory, job satisfaction and dissatisfaction are completely independent of each other.

Organizations should not work to improve the hygiene factors to motivate the employees. Improving hygiene factors will only result in a decrease in dissatisfaction of the employees. To motivate, the organizations should focus on the motivating factors, like achievement, recognition, growth, etc.

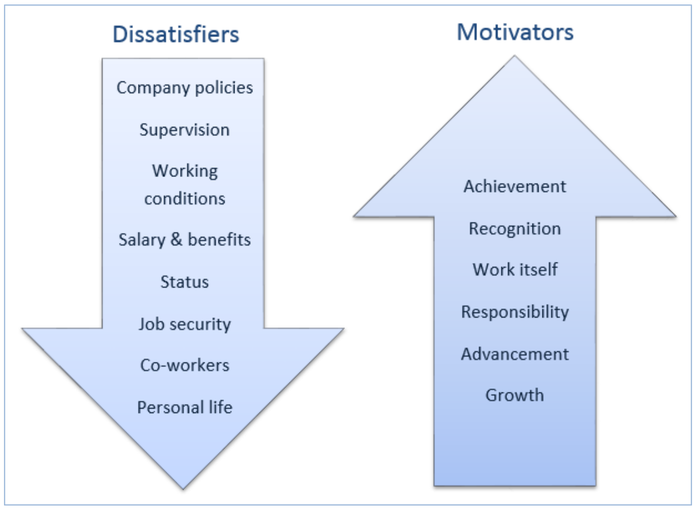

Figure 2: Herzberg's two-factor theory.

Herzberg observed various combinations of motivation and hygiene. The four combinations of this two-factor theory are:

1. High hygiene and high motivation: This is the ideal scenario where the employees are highly motivated with very few complaints.
2. High hygiene and low motivation: Employees do not complain much, but at the same time, are not highly motivated. Example: High salary, but no growth.
3. Low hygiene and high motivation: Employees are highly motivated, but they will have several complaints. Example: exciting and challenging work, but poor working conditions.
4. Low hygiene and low motivation: This is the worst possible scenario. Employee's motivation is at the lowest point and will have several complaints.

1.2 Acquired Needs Theory

David McClelland identified three motivators that he believed all individuals have: the need for achievement, the need for affiliation, and the need for power.

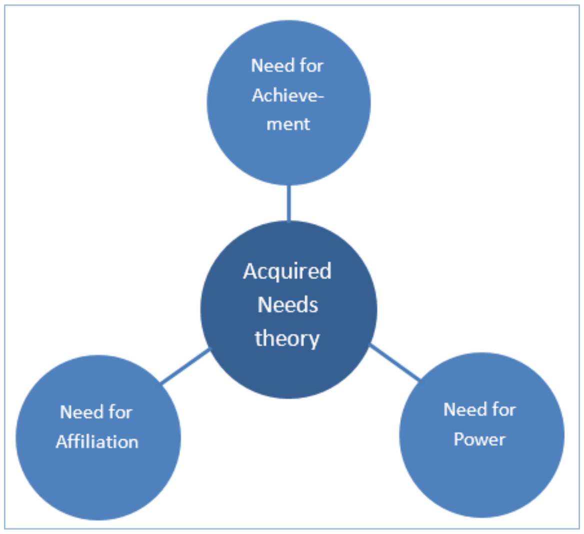

Figure 3: McClelland's acquired needs theory.

1. Need for Achievement: Achievement-based individuals seek to excel and, thus, tend to avoid both high- and low-risk situations. The individuals belonging to this group prefer to work on tasks of moderate difficulty and prefer work in which the results are based on their effort. They like to receive regular feedback on their progress and achievements.

2. Need for Affiliation: The individuals belonging to this group like to maintain social relationships, enjoy being a part of the group, and have a desire to feel loved and accepted. These individuals are more interested in collaboration over competition and they do not like high-risk and high-uncertainty situations.

3. Need for Power: The individuals belonging to this group enjoy work and place a high value on discipline. These individuals enjoys status recognition, competition, winning arguments, and influencing others.

1.3 Equity Theory

John Stacey Adams' Equity Theory states that an individual's motivation is based on what individuals consider to be fair when compared with others. As per equity theory, the employees seek to maintain a fair balance between the inputs they give and the outcomes they receive against the perceived inputs and outcomes of others. The equity theory can be described using the below equation:

$$\frac{individual's\ outcomes}{individual's\ inputs} = \frac{relational\ partner's\ outcomes}{relational\ partner's\ inputs}$$

The weightage on the input and output parameters and value of the outcome are solely determined by the recipient, and so, no single outcome can be measured objectively. One employee may give more value to flexible schedule over income, and another employee may value it the other way around.

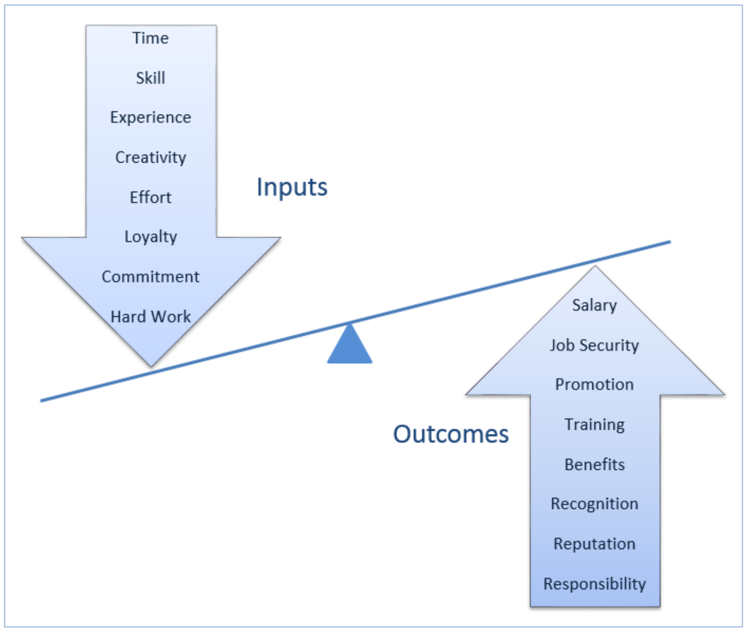

Figure 4: John Stacey Adams' equity theory.

Equity theory consists of four propositions:

1. Individuals will pursue to maximize their outcomes.
2. Groups can maximize collective rewards by developing accepted systems for equitably apportioning rewards and costs among members. Systems of equity will evolve within groups, and members will attempt to induce other members to accept and adhere to these systems.
3. When individuals find themselves in inequitable relationships, they will get distressed. The more inequitable the relationship, the more stress individuals feel.
4. Individuals who perceive that they are in an inequitable relationship will attempt to eliminate their distress by restoring equity.

1.4 Reinforcement Theory

Behaviorist B.F. Skinner proposed the Reinforcement Theory, which states that an individual's behavior can be altered by controlling the consequences of the behavior. Rewards are used to reinforce the desired behavior and punishments are used to prevent unwanted behavior. This behavior theory is also known as Operant Conditioning.

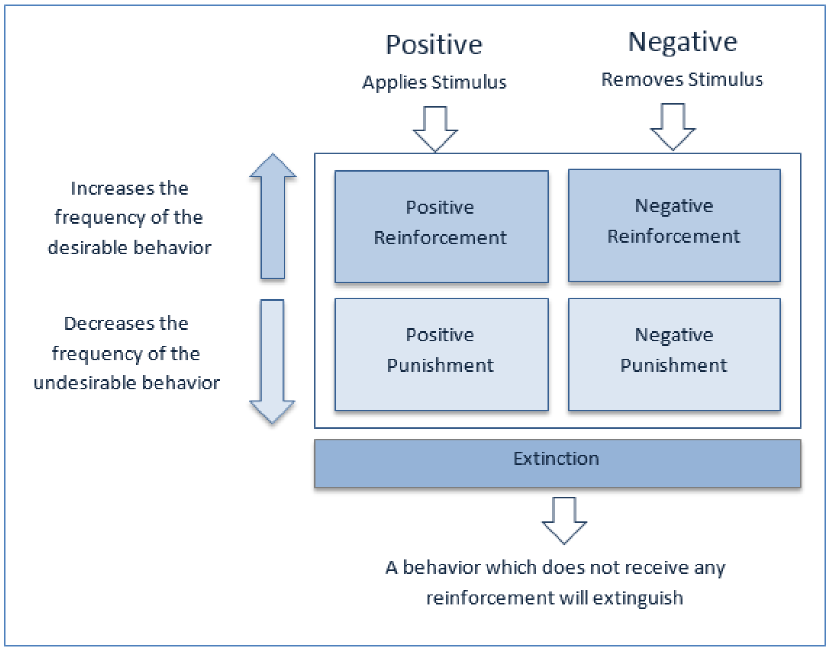
Positive
Applies Stimulus
Negative
Removes Stimulus
Increases the frequency of the desirable behavior
Decreases the frequency of the undesirable behavior
Positive Reinforcement
Negative Reinforcement
Positive Punishment
Negative Punishment
Extinction
A behavior which does not receive any reinforcement will extinguish

Figure 5: Skinner's reinforcement theory.

There are four primary approaches to reinforcement theory:

1. Positive reinforcement: Positive reinforcement is rewarding an individual when he or she has performed a desired behavior. Giving any reward may not result in the desired behavior; instead, the reward must stimulate the individual to produce the desired behavior.

2. Negative reinforcement: Negative reinforcement is rewarding an individual by removing an undesirable consequence when he or she has performed a desired behavior. Negative reinforcement also uses a reward system. An individual is rewarded for his/her desired behavior by removing what the individual does not like. Note that the "negative" in this context does not mean "bad," but it is the removal of something an individual does not like.

3. Extinction: Extinction involves the absence of the reinforcements. This happens until the unwanted behavior gradually goes away or has reached the desired level. Note that the extinction is not permanent and that the unwanted behavior may return after the extinction process is completed.

4. Punishment: Punishment is removing a positive consequence when an undesirable behavior occurs. Reinforcement theory provides two ways to eliminate undesirable behaviors:

    1. Positive punishment: This is the type of punishment most people are familiar with. Positive punishment is giving individuals what they "do not like" when they have performed the undesired behavior.

2. Negative punishment: This involves removing a positive consequence when an undesirable behavior occurs. In other words, removing what the individuals “like” when they have performed the undesired behavior.

1.5 Theory X and Theory Y

Douglas McGregor formulated two contrasting theories for human behavior at work, called Theory X and Theory Y.

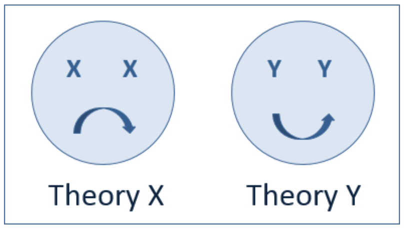

Figure 6: McGregor’s theory X and theory Y.

1. Theory X: As per this theory, theory X managers believe that the employees are naturally unmotivated and do not like their jobs. Therefore, an authoritarian management style is required and these individuals need to be closely supervised. Since employees do not like their job, they need to be persuaded, compelled, and/or punished to achieve the organization’s objectives.

2. Theory Y: Theory Y managers assume that employees are self-motivated, creative, responsible, and enjoy working. Theory Y managers believe that given proper conditions, employees will learn, accept responsibility, and exercise self-control in accomplishing their objectives.

1.6 ERG Theory

Clayton Alderfer developed ERG Theory to explain the simultaneous nature of Maslow’s hierarchy of needs. ERG theory grouped Maslow’s five hierarchy needs into three needs:

1. Existence needs: These are the needs for basic material existence, which include what Maslow considered to be physiological and safety needs.
2. Relatedness needs: These needs are for maintaining interpersonal relationships, social status, and recognition. This group includes Maslow’s social needs and the external component of esteem.
3. Growth needs: These are the needs for personal growth. This need category includes Maslow’s self-actualization needs and the intrinsic component of esteem.

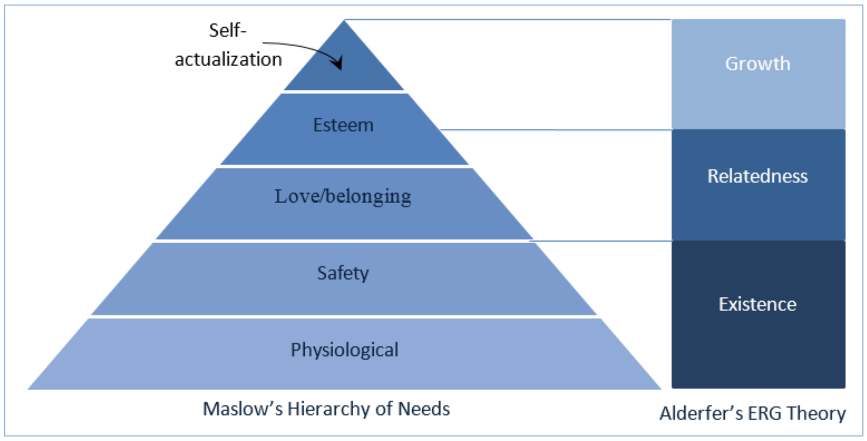

Figure 7: Maslow's hierarchy of needs and Alderfer's ERG theory.

Contrary to Maslow's hierarchy theory, ERG theory states that people can be motivated by more than one level of needs at the same time. It also states that the importance of the needs differs for each person. Some people may put growth as a higher value than relatedness at certain stages of their lives.

1.7 Other Motivational Theories

A few more theories of motivation worth knowing about are:

- Cognitive evaluation theory
- Expectancy theory (Vroom expectancy motivation theory)
- Goal settings theory

2. Transparent

Transparency is not only important to project managers and the management team, but also to every individual team member. Transparency allows everyone to see and understand where the project is and what is really happening in every iteration. The information displayed should be very simple and concise and should not be obsolete.

2.1 Information Radiators

Alistair Cockburn coined the term "Information radiators" for the highly visible ways to display information. The information can be shown as graphs, charts, or a summary of data. The information radiators are usually displayed in large traffic areas to maximize their exposure.

2.2 Kanban Boards

Kanban is very simple, yet very powerful. A Kanban board consists of a large board with cards placed in columns, with numbers at the top of each column. The numbers at the top of the columns represent the limit in work-in-progress (WIP). Limiting the amount of WIP in each process prevents overproduction and reveals any bottlenecks.

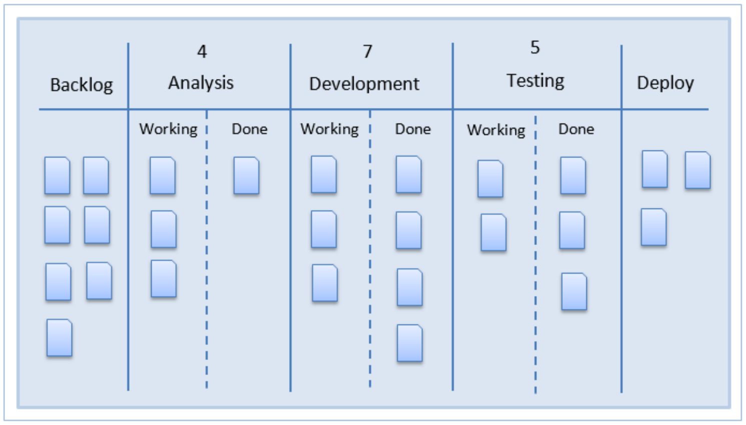

Figure 8: Kanban board example.

2.3 Burn Down Charts

Burn down charts are a visual representation of a project's progress and help to determine when the project should be complete. Burn down charts typically display information about the remaining effort versus time.

In the below example, assuming that 7 is the current iteration, the effort completed so far is 30 units and 20 units of effort are remaining.

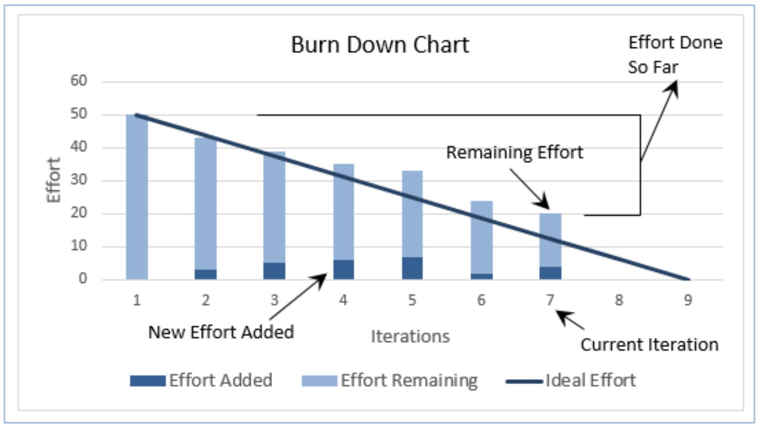

Figure 9: Burn down chart example.

2.4 Cumulative Flow Diagram (CFD)

CFD helps us to understand the work in progress (WIP), cycle times, bottlenecks, and likely completion dates.

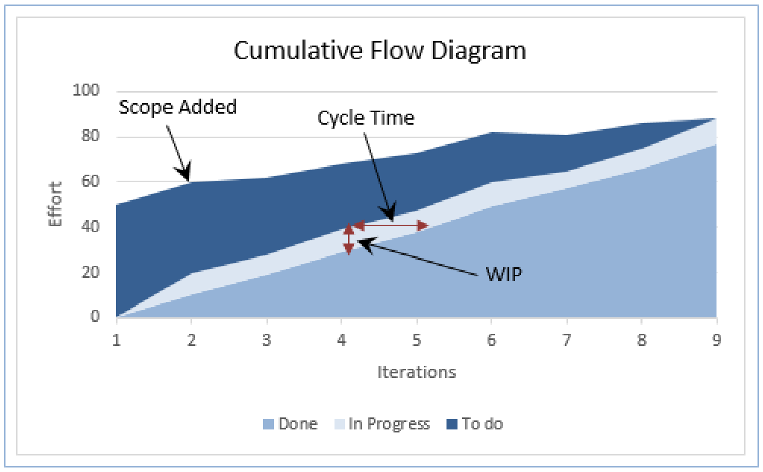

Figure 10: Cumulative flow diagram example.

Average throughput can be calculated using the Little's law formula:

$$Avg.Throughput = \frac{Avg.Work\ In\ Progress\ (WIP)}{Avg.\ Cycle\ Time}$$

Avg. WIP = Average number of items being worked on at any time
Avg. Cycle Time = Average time taken to complete a work item
Avg. Throughput = Average work items produced per unit of time

An important piece of the information that CFD provides is regarding process bottlenecks. Bottlenecks are the areas where the layers go from thin to thick. The cause of the problem will be from the process layer shown below the thick layer.

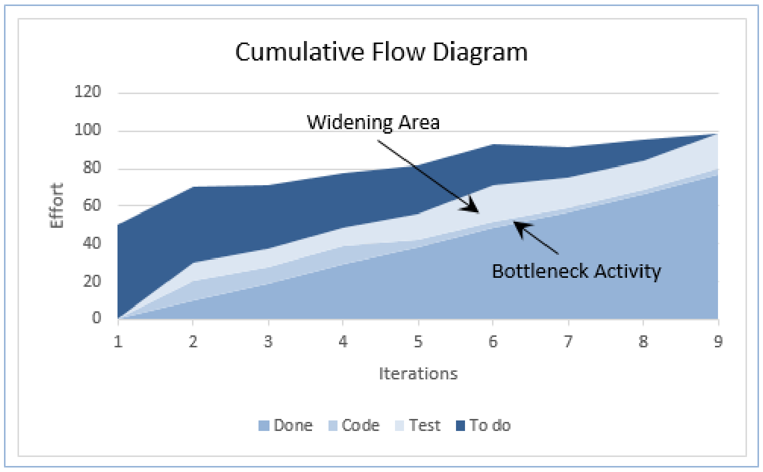

Figure 11: Bottleneck information shown in CFD.

Since the line gradient represents the rate of progress for an activity, a widening area is created around the activity that is progressing at a slower rate.

2.5 Product Roadmap

A product roadmap displays the product vision and its high-level goals to the stakeholders. Product roadmaps are not a place to show every last detail of the product development. In many cases, you will not know the expected delivery dates beyond the ones planned for your iteration. So, you will not have the exact date on the product roadmap; instead, for short term plans, you may have something like August 2005, and for the long-term plans, you may have something like Q2 2016. The product roadmap is a plan and not a commitment for delivery.

Note that a Gantt chart is not your product roadmap. Gantt charts are for project management and should not be used for product management.

3. Feedback Loops

Obtaining timely feedback from stakeholders plays a vital role in the successful completion of any software project. Below, are three simple steps to get the most out of customer feedback:

- Listen: If you are listening, every customer has many stories to tell.
- Understand: Simply listening, without interpreting the meaning of the customer’s story, fails to perceive the actual issues behind the customer’s story.
- Act: Act swiftly and incorporate all valid customer feedback in the software product.

In agile software development, the feedback is received from several sources— code reviews, meetings, retrospectives, testing, and many more.

3.1 Gulf of Evaluation

Many times, there will be a huge difference between the product or the feature as it is envisioned by the customer/stakeholder and as it is interpreted by the team. This difference, in computer science, is also known as the “gulf of evaluation.”

The gulf of evaluation is usually small in physical engineering projects, since the work is visible, tangible, and familiar. In contrast, for software projects, the work is invisible, intangible, and new, and therefore, often leads to a greater gulf of evaluation. Capturing customer feedback continuously will bridge the gap in this gulf and aids in building the right product. Any changes from the customer feedback should go back into the requirements and should be prioritized.

3.2 Cost Change Curve

In traditional software development, the average cost of fixing a defect rises exponentially with time. Therefore, the longer it takes to find a defect in the system, the more it costs to address the issue.

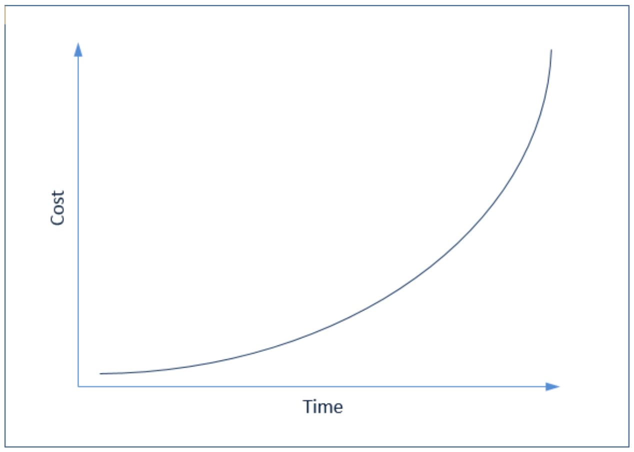

Figure 12: Cost-change curve.

With continuous feedback from the stakeholders, the cost curve can be flattened. It is very important to get the feedback from the end users. These are the people who can tell you what is right and what you have missed. The longer the feedback loops, the more time and work that will be required to fix any defects.

3.3 Test-Driven Development (TDD)

TDD is a software development process that repeats on a very short development cycle in which the developer writes the test case before writing the minimum amount of code to pass the test, and finally, refactors the code.

- Write a single unit test.
- Run the test and the test should fail since there is no written code.
- Write the minimum (“just enough”) code necessary to pass the test.
- Refactor the code.
- Repeat this process.

3.4 Continuous Integration (CI)

CI is merging all the code modified by the developers into a shared code repository several times a day. Each merge is verified by an automated build, which allows the developers to detect problems early. Since the code is merged very frequently, there is significantly less back-tracking to figure out where things went wrong.

3.5 Continuous Delivery (CD)

CD is an approach in which the valuable software is produced in short cycles so that the software can be reliably released at any time.

3.6 Other Feedback Methods

Some other feedback methods are:

- Acceptance test-driven development (ATDD)

- Behavior-driven development (BDD)

4. Reduce Waste

Any process which does not add value is waste. Simply put, waste is the opposite of value. It is impossible to eliminate waste completely. We should strive to reduce as much waste as possible. However, even though waste does not add value, in some cases, it may be useful.

4.1 Manufacturing Wastes

Manufacturing wastes were first identified at Toyota by Taiichi Ohno. The seven wastes in manufacturing are:

1. Inventory: Inventory needs to be stored, which will incur storage costs and does not add value to the product.
2. Overproduction: Producing more goods than the actual demand. Overproduced goods need to be moved (which requires people and equipment) and stored. All these processes incur costs and add no value.
3. Overprocessing: Overprocessing is adding steps to the process that are not really needed. Overprocessing may require more time from the resources, more materials used, and/or wearing down the equipment, resulting in additional costs without adding any value.
4. Transport: Transport is moving materials from one place to the other. Moving materials involves cost and does not add any value to the product.
5. Waiting: Breakdown of machines, unreliable processes (waiting on the previous process to complete), or a delay in components delivery can cause waiting.
6. Motion: Unnecessary motions are the movement of the resources (either workers or machines) over what is required to perform the process. Some of the causes of motion waste are due to poorly designed layout.
7. Defects: Defects are flaws in the produced goods that impact the goods' features and functionality. Poorly produced goods require either rework or replacement, which incurs cost with no added value.

4.2 Software Development Wastes

Mary and Tom Poppendieck (2013) translated these seven manufacturing wastes into software development.

1. Partially Done Work: Any incomplete work which is not integrated, tested, or deployed is like excess inventory. Until the work is completed, it will not add any value. Furthermore, the partially done work may become obsolete long before it is completed. Therefore, the amount of partially done work should be kept to a minimum.
2. Extra Features: These are the features that are not required or do not add any value. These features not only take up development time, they also need to be documented, tested, and maintained.

3. Relearning: Relearning refers to repeating a value-added activity. Code that is not documented and/or not written well will force the developer to relearn, so as to understand the code's behavior every time the developer needs to maintain the code.
4. Handoffs: Knowledge lost when deliverables are handed off to another resource. During the handoff, the tacit knowledge will never get passed on.
5. Delays: Delays are waiting on something. Waiting for approvals and waiting on resources to become available are some common forms of delay.
6. Task Switching: This is the time wasted when a resource is assigned to multiple projects. When switching between the projects, the resource needs to re-adjust and may have to relearn the knowledge.
7. Defects: Any software bug that needs to be fixed, either small or big, is a defect in the software. Poppendieck's formula to quantify the amount of waste caused by a defect is as follows:

$$Waste = (Impact\ of\ defect) \times (Time\ defect\ lies\ undetected)$$

Conclusion

Many popular agile approaches that exist today have strict rules and/or requirements. D-cube is a very simple and highly flexible agile approach that can be tailored to fit to your needs.

D-cube primarily emphasizes the need to continuously motive and train the project team, be very transparent, get continuous feedback from stakeholders, and continuously reduce waste whenever possible.

You pick the rules. You pick the roles. And you follow the process and choose the emphasis on motivation, transparency, feedback, and the elimination of waste. This is D-cube.

References

Equity theory. (n.d.). In *Wikipedia*. Retrieved November 16, 2015 from https://en.wikipedia.org/wiki/Equity_theory

Griffiths, M. (2012). *PMI-ACP exam prep, premier edition: A course in a book for passing the PMI agile certified practitioner (PMI-ACP) exam*. Minnetonka, MN: RMC Publications.

Koskela, L. (2007). *Test driven: TDD and acceptance TDD for Java developers*. Greenwich, CT: Manning Publications.

Kua, P. (2012). *The retrospective handbook: A guide for agile teams*. Vancouver, BC: Leanpub.

Need theory. (n.d.). In *Wikipedia*. Retrieved November 16, 2015 from https://en.wikipedia.org/wiki/Need_theory

Poppendieck, M. & Poppendieck, T. (2013). *The lean mindset: Ask the right questions*. Boston, MA: Addison-Wesley Professional.

Two-factor theory. (n.d.). In *Wikipedia*. Retrieved November 16, 2015 from https://en.wikipedia.org/wiki/Two-factor_theory

About the Author

Chandra R. Munagavalasa, PSM I, SSGB, PMI-ACP, PMP currently works as manager in software development in Houston, TX. He works with several Microsoft products and enjoys developing software in C#. He is also a PMP, PMI Agile Certified Practitioner, Six Sigma Green Belt, and Professional Scrum Master-I. Mr. Munagavalasa holds a master's degree in industrial management from IIT-Madras.

Email: chandra.munagavalasa@gmail.com
LinkedIn: http://www.linkedin.com/pub/chandra-munagavalasa/2/832/920
Twitter: https://twitter.com/cmunagavalasa