BEST PRACTICES WHITE PAPER

Value-Driven Delivery

Erik Simmons, Senior Fellow, Construx Software

Version 1.1, July 2018

Value-Driven Delivery is the most compelling way to structure and sequence work when creating solutions. By delivering value to stakeholders in short, regular intervals, in the order the stakeholders prefer, teams learn about the problem at hand and how the solution elements perform. They also enable potential business results from early use of parts of the system.

Construx®
SOFTWARE DEVELOPMENT BEST PRACTICES

Contents

Introduction

When a team sets out to create a solution to address a problem or opportunity, it can proceed in many ways. For example:

What is easy: The team starts out with what they perceive to be easiest or most familiar. This strategy can create the feeling of a good start and rapid progress and can establish trust and credibility for the team. But schedule and resource commitments can be broken when difficult or unfamiliar things appear later in the project—and no one likes late surprises.

What is risky: The team begins with the riskiest elements so that unpleasant surprises are exposed early. This permits re-estimation and adjusted commitments with greater confidence because "bad news" should be discovered early if it exists.

The largest assumptions: The team addresses the largest business and design assumptions to test whether they are true. This allows for early course corrections and avoidance of dead-end development options.

The user interface: The team starts with the user interface and works downward into the system's implementation based on the chosen architecture. This allows for early feedback on the user interface and its functions.

The team has many additional choices beyond these examples, but the most compelling choice for how the team sequences work is to be ***value-driven***.

What is value?

Value is a concept we encounter frequently in daily life. Value is defined in two main ways:

1. The importance, worth, merit, or usefulness of something. For example, a device can have valuable features.
2. A principle or a behavior standard. For example, honesty and generosity are valued in people.

This definition shows that value has both an *economic* and a *philosophical* aspect. In the economic aspect, value is measured using some equivalence to a currency, whether that currency is money, time, or even personal reputation. Some people value getting a new product the day it is released, even if the price would be lower later on and they must stand in line for several hours in a cold rain to get it.

The philosophical aspect of value is not based on some currency but rather on what is right or what is good.

It is useful to distinguish *value-in-exchange* from *value-in-use*. This distinction, which is also known as the *paradox of value*, was popularized by Adam Smith in *The Wealth of Nations* (1776). Value-in-exchange refers to what you pay (in some sense) for something when you acquire it. Value-in-use comes later, at the time when you use or consume the object.

At the time of purchase, we are placed in a situation where we must estimate the item's value-in-use. This can be difficult, especially if the amount of time between purchase and use will be large. Sometimes disappointment occurs even when we fully realize the item's potential value-in-use because we inaccurately predicted how much satisfaction we'd feel from owning something or achieving a goal. Psychologists call this *miswanting*. For example, people who make New Year's resolutions for weight loss often purchase exercise equipment with the expectation of value-in-use, only to find the equipment sitting unused within a few months.

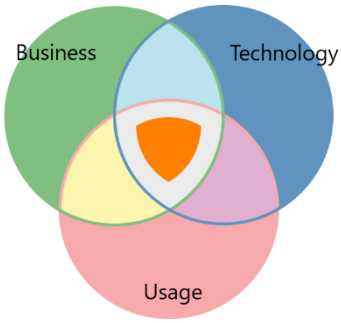

Figure 1

Another way to frame and think about the paradox of value comes from the Three-Circle Model, shown in Figure 1. Each of the three circles represents a unique viewpoint on a solution.

The **Business** circle represents the economic viewpoint. A solution must be *marketable*, *profitable*, and *affordable*.

The **Usage** circle represents the conceptual viewpoint. A solution must be *desirable*, *usable*, and *useful*.

The **Technology** circle represents the implementation viewpoint. A solution must be *manufacturable*, *functional*, and *consumable* (by the industry and associated ecosystems).

Within the Three-Circle Model, value is the overlap between the Business and Usage circles, as shown in Figure 2.

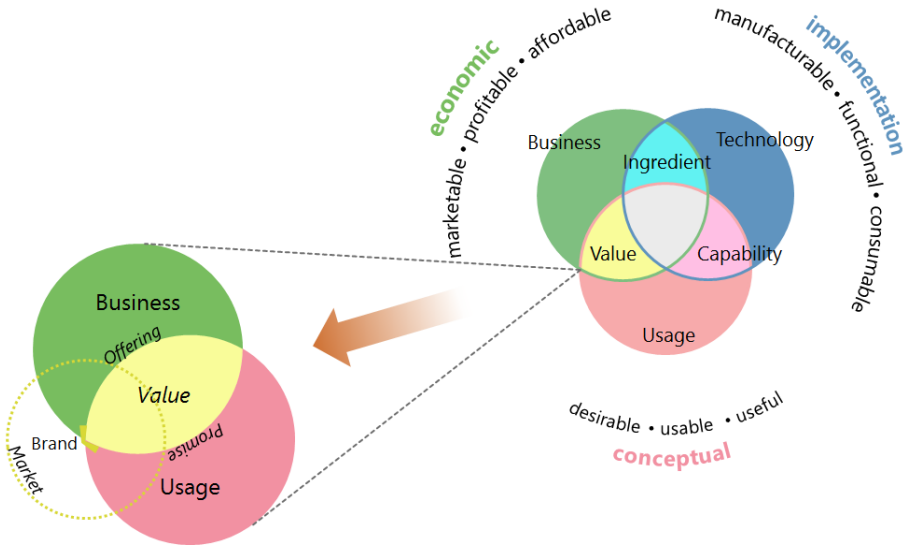

Figure 2

Value unites the economic and conceptual perspectives, tying investment to user experience. Users see a solution as an offering made by the business, and the business views the solution as a promise made to users. Thus, Value is the area between offering and promise. The offering and promise meet in the

solution's brand, which is the focus of the market cycle. By uniting the Business and Usage perspectives, Value relates value-in-exchange to value-in-use.

The Three-Circle Model contains many other insights, but one particularly worth mentioning in the value-driven context is that there must be coherence among the solution's Value, Capabilities, and Ingredients. The solution's Capabilities must be those that are necessary and sufficient to generate its Value, and the Ingredients must in turn be necessary and sufficient to enable those Capabilities. A focus on value prevents scope creep and gold-plating, while assuring that no value is left unrealized.

Value-Driven Delivery

Value-Driven Delivery is achieved using the answers to these three simple, but not at all easy, questions:

1. Who are your stakeholders?
2. What do they value?
3. What are you doing in the next two weeks or less to provide value to them?

Who are your stakeholders?

Teams often have more stakeholders than it first appears. A *stakeholder* is an individual or organization with some *material stake* in the outcome of the program or project. It is not uncommon for a team to have one or two dozen stakeholders, including end users, peer teams, component teams, management, quality, support, legal, etc.

The list of stakeholders must be kept current. Maintaining an up-to-date list of stakeholders is crucial to the practice of Value-Driven Delivery because value is defined by stakeholders. It is common for stakeholders to change their minds on what is valuable during projects, so it's important to keep in touch with them. Also, stakeholders can appear or disappear over time, and the team must be aware of all such events so that it can respond appropriately. When a project has a large number of stakeholders, the team must consciously prioritize among them to ensure the best possible allocation of development resources. The final section of this white paper, "Supporting models, concepts, and techniques," contains an introduction to several models and concepts that can help you prioritize stakeholder groups.

What do they value?

With an accurate list of stakeholders, the team must develop an understanding of what the stakeholders value. Those familiar with Lean Startup principles and practices will find Customer Development a good place to start. But there are

many other stakeholders beyond customers, so Customer Development is not enough by itself.

One challenge is that when asked what they value, many stakeholders will quite naturally reply in terms of features, capabilities, or functions they would like to see. Few will reply with their underlying values. In most such cases, it is good to ask "why?" to get beneath their constrained implementation requests to understand the values at work. Many, perhaps most, stakeholders are not deep domain experts, so they won't be able to envision the ramifications of a particular implementation request. Also, stakeholders rarely understand the detailed capabilities of the tools the team is using to construct the solution. So, the suggested feature, capability, or function is unlikely to be the best idea from many perspectives, such as cost effectiveness, maintainability, scalability, performance, or security.

Getting beneath a stakeholder's constrained implementation ideas to his or her underlying values can be tricky. But there are good techniques for doing so, especially within some new Requirements Engineering practices.

A focus on stakeholder value broadens the range of work a team can undertake. For example, many years ago a team that I led was engaged to create a new membership application for a trade union. The union's current membership system had several problems and shortcomings, including very poor report performance. It took hours to determine what needed to be known in minutes or seconds, and some summary reports took days to complete, if they completed at all. The new system was of a size that would take a year or more to finish. In the traditional model, the team would collect requirements for the first month or two and then leave to design, construct, and test the system. The end users would receive value only when the system was rolled out all at once a year or more later, and the team's only product would have been the delivered software system.

But this effort was managed using a Value-Driven Delivery approach. The team was keenly aware that its product was not a delivered system, or software code, or completed tests, or documentation pages—it was stakeholder value. The most valuable thing the team could do first was to bring in a database specialist to improve report performance in the current system by fixing some indexing problems and rewriting a few queries and stored procedures. This work did not contribute to the new system at all, but the users would have suffered with the existing system's issues for another 12 months otherwise. This simple, quick investment generated significant stakeholder value within the first week of the project. Besides the obvious business results, the development team also gained the trust and respect of all the stakeholders, which paid dividends through the project and led to many more incremental value deliveries before the full system was finally in place many months later.

What are you doing in the next two weeks or less?

Value-Driven Delivery is based on early, frequent stakeholder value deliveries. The two-week duration stated here is a rule of thumb, and you might find that another duration fits your environment. However, shorter is nearly always better, so push for the shortest feasible iteration length.

Using Value-Driven Delivery can mean that the team cannot necessarily do what is easiest, most convenient, or least expensive first. However, the benefits of early, regular value delivery vastly outweigh the costs the team incurs to behave that way. The benefits of early, frequent stakeholder value deliveries include:

1. Early learning about the problem at hand and the environment that surrounds it.
2. Early stakeholder feedback on delivered parts of the solution.
3. Information on how the solution elements perform, and the chance to improve them in subsequent delivery cycles.
4. Potential business results from early use of parts of the solution.

Of course, there are many elements that a team might build as part of solution development that require longer than two weeks (or one cycle) to fully complete. This work can proceed based on stakeholder priority, but the challenge remains to deliver something of value to some stakeholder every two weeks or less. In every case, some part or aspect of a larger solution element can be delivered within a two-week time frame at most.

Ideally, value deliveries will be sequenced so that the most valuable things get delivered first. The challenge is to minimize the time required for the team to get net-positive return on investment (ROI) given the program's accruing costs.

With traditional sequential work, all the value is delivered at the end of the program via a big-bang rollout of the entire system. Cost accrues throughout the project, so there's a danger the team won't get to positive ROI before time or money runs out (Figure 3). In most environments, working with a traditional sequential approach is too risky.

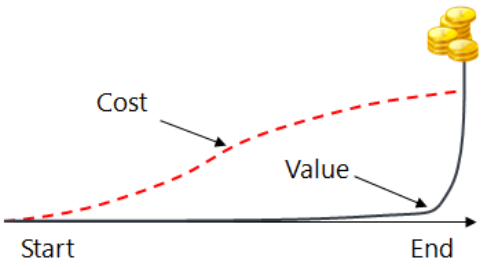

Figure 3

Adopting an incremental approach helps. The team gets value into the system earlier, so there is a good chance of keeping ROI at least close to the project's cost accrual over time (Figure 4). This is a major risk-reduction win. But things could still be better.

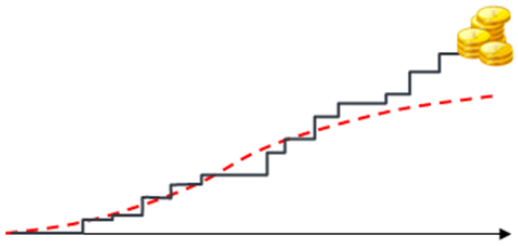

Figure 4

If the team can sequence its deliveries so that the most valuable things get done first, it can create positive ROI very early, possibly even from the first delivery cycle (Figure 5).

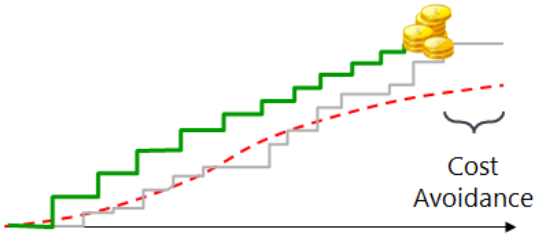

Figure 5

This is a huge benefit of a value-driven approach, which can lead to significant cost avoidance because work is not performed based on inertia or on some dated and bloated scope definition.

The value-driven approach provides still one more benefit. Because stakeholders have parts of the solution quite early, the lessons and insights that come from their use often expose new, unanticipated sources of value. So, the diminishing returns depicted in Figure 5 are often postponed, and the value delivery continues at substantial levels for much longer. This results in higher total value delivered by the team (Figure 6).

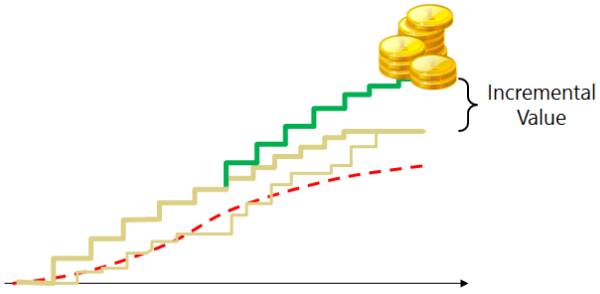

Figure 6

Effects on estimation

There is often a battle of wills at the beginning of a project based on the scope, resources, schedule, and quality. You have probably experienced some form of the argument where an executive or manager says, "The system must be finished in 12 months," to which you may have replied, "But our estimates show it can't be done in less than 18 months." The argument continues and you might be pressured to accept your boss's goal for completion date as your

estimate. Happily, this problem frequently vanishes when you use Value-Driven Delivery. When stakeholder value deliveries begin very early, and then continue every couple of weeks, the discussion changes from “How soon will you be done?” to “How long can you keep this up?” That’s a very different environment, one that emphasizes abundance and empowerment rather than scarcity and control. In the union membership application example cited earlier (in the “What do they value?” section), the project continued for nearly two years, not the nominal 12 months it would have taken to merely replace the membership application. The extension was not because of schedule slips, but because the stakeholders kept asking for more of the value the team was delivering with reliable regularity.

This example illustrates how a value-driven project really has no end date in the traditional sense. Value-driven projects end when all the things we could still do are not worth the cost of doing them. The concept that a project has no formal end date can be a challenge to traditional project and program management thinking. But by stopping work on a given project with low remaining value, those resources can be allocated to other, higher-value projects. In the end, an organization must maximize stakeholder value delivery while controlling costs, so avoiding low-value work is an excellent heuristic. Value-Driven Delivery is a great approach for today’s increasingly complex environment.

Structuring and conducting value-driven work

What is the best approach to structuring value-driven work? Let's examine three alternatives:

1. An *iterative* approach involves making several passes rather than attempting to complete work in a single pass.
2. An *incremental* approach involves working on parts of the overall solution in sequence, adding them to the completed prior work.
3. An *evolutionary* approach is both iterative and incremental and uses learning to drive future work.

Due to the evolving nature of stakeholder value, **an evolutionary approach is crucial to success** on value-driven projects. Iterative or incremental processes *can* use learning to drive future work, but they are not required to do so and, in practice, teams using these approaches often neglect or even ignore learning because of time or resource constraints. In an evolutionary approach, learning *must* drive future work—it is a central feature of evolutionary work.

Evolutionary Delivery

One excellent, long-established way of working in an evolutionary way is called Evolutionary Delivery (Evo). Evo was created by Tom Gilb, who is often called "the Grandfather of Agile," and it was described in a book called *Principles of Software Engineering Management* (Addison-Wesley, 1988).[1]

Evo includes two main parts. The first is called the *head* (also called the Strategic Management Cycle), which serves as a sensing and coordination mechanism for the work. The head focuses on top-level objectives, on global architecture, and on maintaining a coarse-grained evolutionary plan. The head is where the team keeps track of stakeholders and what they value.

The head controls the second part of Evo, the *body* (also called the Result Cycle). The body is a value-delivery engine, working in steps that are typically two weeks or shorter to deliver stakeholder value. A step is defined, constructed, and delivered, and—importantly—the results are then validated against the step's original quantified success criteria. These results and the associated learning feed the next cycle and provide feedback to the head about how well the step worked. The basic Evo process is shown in Figure 7.

[1] See Gilb's *Competitive Engineering* (Elsevier, 2005) for a more recent description of Evolutionary Delivery.

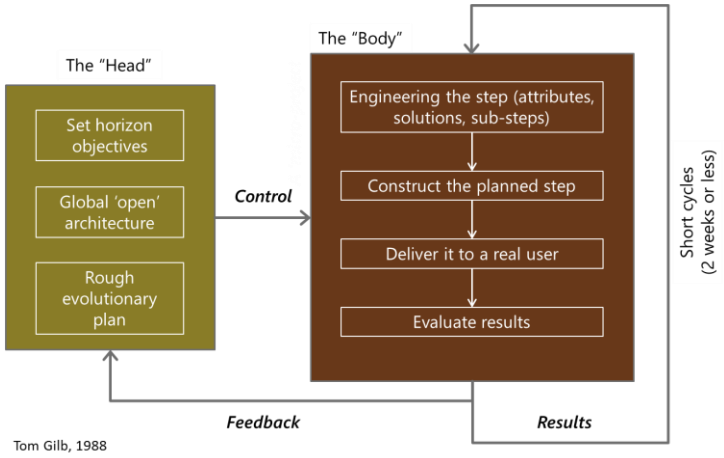

Figure 7

A central facet of Evo is that stakeholders assist with selecting the next step each time, so their values and priorities are considered directly. This regular stakeholder input results in the most valuable steps possible being taken first, which was the optimal approach described earlier. It is not just a matter of being iterative and incremental. Evo and the learning it generates enables teams to generate positive ROI very early and then stay there.

People often object to the Evo approach initially, claiming that their work cannot be broken down into such small pieces. But experience shows that this is possible for any nontrivial project, from a small software system to an aircraft carrier. All that is required is to reframe the problem and the solution in terms of stakeholders and their values. With that primary view on the system, the candidate Evo steps are usually easy to see. If, however, we look at the system only as a complex, impenetrable whole (or a heap of source code), the steps remain invisible.

What about Scrum?

Scrum can function as a fine Evo body for result delivery, but—at least in common implementations—it lacks the value-driven focus of Evo. Even if your team is practicing Scrum, you can find important additional performance gains by adopting Evo as a "value engine" on top of Scrum's "execution engine." Ryan Shriver has written eloquently and powerfully on this topic.[2] According to Shriver, adding Evo to Scrum helps transform focus from execution to value:

[2] See https://ryanshriver.files.wordpress.com/2013/01/valuedelivery_sg2010.pdf.

Execution Focus	Value-Driven Focus
Feature Building	Value Delivery
Focus on Means	Focus on Ends
Planning by Features	Planning by Value
Maximizing Story Velocity	Maximizing Value

Specifying stakeholder value

Once you have identified something that a stakeholder values, it's important to document it in a quantified, verifiable way. Otherwise, how can you know whether you have delivered it? An excellent technique for doing this is called *Planguage*. The name is a contraction of planning and language. Planguage is a keyword-driven natural language. It can be learned in just a few hours, and the results can be read and understood by a broad range of people. Like Evo, Planguage was developed by Tom Gilb as another part of his broader methodology *Competitive Engineering*.

Values expressed using Planguage will contain both a Scale and a Meter. The Scale describes the scale of measure, while the Meter describes the process or device used to establish location on the Scale. For example, when driving along a highway in a car in the USA, the Scale for velocity is miles per hour and the Meter is the speedometer. Specifying values is often much more nuanced than this simple example, but this example serves as a helpful anchor when working with more challenging material.

For example, suppose a stakeholder tells you that an order entry system is too difficult to learn and she wants it to be easier. You could, after more discussion with her, come to specify her value in the form of a Planguage specification similar to this:

Ambition: Make the system easy to learn.

Rationale: Learnability issues are among the top 3 complaints from users.

Priority Reason: Upcoming hiring makes learnability for order entry critical.

Scale: Average time required for a new user to complete a 1-item order using only the online help system for assistance.

Meter: Measurements obtained on 25 new users during user interface testing.

Source: Anna Coleman.

Minimum: No more than 7 minutes.

Target: No more than 5 minutes.

Outstanding: No more than 3 minutes.

Past: 11 minutes <-- Recent site statistics.

New user: Someone who has never used our order entry system before but is familiar with browser-based applications.

Documented in this format, the stakeholder's value is quantified and defined in such a way that everyone can see how success will be judged. Anyone who disagrees with or fails to understand the definition, targets, or measurement method can raise questions or concerns explicitly.

A range of success is captured in the Minimum, Target, and Outstanding achievement levels, allowing for flexibility and tradeoffs among competing demands during implementation. The Minimum provides the just-acceptable performance level, dividing success from some form of failure. The Target value is what we would like to achieve (often some safety margin above Minimum). The Outstanding value is something that could feasibly be achieved if everything went perfectly. In today's complex and rapidly changing environments, this *flexible clarity* is essential for teams.

If certain stakeholder values are critical to the success or failure of a project, they can be placed into a table called a Landing Zone. A Landing Zone describes a "region" of success for a project or product in no more than one to two dozen rows. Landing Zones provide many benefits, including improved and accelerated decision making and an explicit definition of success.

Detail level and timing issues

Because of the quickly evolving nature of stakeholder value, it should come as no surprise that a stakeholder value specification cannot be accomplished in a single, exhaustive pass at the beginning of a project. It is infeasible to expect everyone to know all the things that will become important as the solution evolves. Changing stakeholder values and a changing stakeholder population all but guarantee the need to evolve the specification over time.

If everything cannot be written up front, how much detail is enough? When must it be written? There is no fixed answer for all projects, but we can apply useful heuristics to answer the questions in any given instance.

First, what does it mean for a specification to be "complete"? One answer would be when it contains every possible statement and detail. But this is both impossible and unwise. Even if you could write down every detail, you shouldn't have to. Your team has experience and domain expertise, as do various stakeholder groups.

A more pragmatic, effective definition of specification completeness is this:

> *A specification is complete when it contains the details necessary for those who read it to do their work at an acceptable risk level*.

Using this definition, a specification can be complete on the first day of the project, the second, and so on until the last day, even though the specification might be significantly larger by the project's end than it was at the beginning.

> **Heuristic: Capture stakeholder values at just enough detail to enable current work at an acceptable risk level.**

Additional details can be added as needed over time. However, avoid the trap of specifying a lot of detail in areas that are well-known and understood. It may feel like great productivity to generate a hundred pages or more of specification in the first week of the project. If it is filled with precedented details, this specification does not reduce project risk levels. It merely documents what everyone already knows, making the specification's value mainly ceremonial. To avoid this trap, use the following heuristic:

> **Heuristic: Focus specifications on new, risky, and complex areas of stakeholder value.**

This heuristic will generate more detail when it is most useful, not when it is easiest to document. One consideration is that the valuable content is likely to change with time. For example, team members gain domain expertise with time, so some information can be abstracted. Therefore, the template or data structure that comprises the specification must be kept current:

> **Heuristic: Regularly update the specification template or data structure to reflect the *current* needs of those who use it to guide their work.**

Problems can occur in either direction. Some content may be "inertial," appearing because it appeared in previous specifications rather than because someone derives value from it currently. Also, specification readers might be lacking information they need to keep risk at an acceptable level. Regular re-evaluation of the specification content and detail emphasis can prevent waste and manage risk. When did you last ask the people who read your specifications what they value? What fraction of your specifications are not read by anyone?

Supporting models, concepts, and techniques

Various models and techniques are helpful when used with Value-Driven Delivery. Five models and techniques are described here, but you might find other models and techniques useful in your value-driven work:

1. Agile stories
2. The Kano Model
3. Diffusion of Innovations
4. User experience proof points
5. The HEART framework

Agile stories

Anyone who has worked within an Agile environment is probably familiar with Agile stories. The most common syntax for Agile stories is

> *As a* [role name]*, I want* [feature or function] *so that* [**goal or value proposition**].

Stakeholder values can be discovered when using Agile stories, since they explicitly contain a role name and the value that some stakeholder perceives or anticipates when asking for some feature or function. Stories also contain Acceptance Criteria, which can contain additional details and insights into the stakeholder's values. Here is an example story:

> *As a digital photographer, I want to be able to store various settings so that I can recall them quickly and not miss an important picture.*
>
> *Acceptance criteria:*
>
> 1. *The camera holds up to five sets of user settings.*
> 2. *Settings include white balance, autofocus, flash, ISO rating, file format, and auto-exposure.*
> 3. *Setting sets can be created, read, updated, and deleted.*
> 4. *A setting set can be recalled with no more than two taps.*
> 5. *Each setting can be given a user-defined name of up to 32 characters.*

Good acceptance criteria are clear, concise, written in the language the end user understands, and verifiable. From a value-driven perspective, story verifiability must include the *goal or value proposition* level, not just the *feature or function* level. **Don't assume correct implementation of the feature or function alone guarantees the satisfaction of the goal or value proposition asserted in the story.** In the example story, there is no guarantee that meeting only the stated acceptance criteria to the letter will in fact deliver the necessary value to the camera user. As we mentioned earlier, many

stakeholders are not domain experts and do not fully understand the tools used to create a solution. Their requested features, capabilities, and functions might not be the best way to achieve the stakeholders' underlying values.

An Agile story is best thought of as a promise to hold a future conversation. The story's existence indicates that the feature, capability, or function is important and valued by the stakeholder, and it provides a bit of the context. But it is likely that additional details will be needed prior to delivery. In most cases, these can come from a conversation just prior to starting implementation. At that time, the conversation alone might suffice, but often the additional details can and should be captured in Planguage statements or system models such as those within UML or SysML. It is not uncommon for the story to evolve significantly as discussions elaborate on the stakeholders' values.

The Kano Model

Developed in the 1980s by Noriaki Kano, the Kano Model[3] is an effective model for locating, classifying, and analyzing stakeholder values. The model maps stakeholder satisfaction against various types of solution qualities. Because the original paper has been translated in various ways into English, many versions of the terms used for the model's quality categories exist. One easy-to-understand set of these terms is shown in Figure 8.

In the Kano Model, the vertical dimension represents customer satisfaction, as indicated by the unhappy and happy faces in the figure. The horizontal dimension represents the degree to which various solution qualities are present or instantiated. For the purposes of this article, we'll discuss the model in terms of stakeholder requirements and their resulting value.

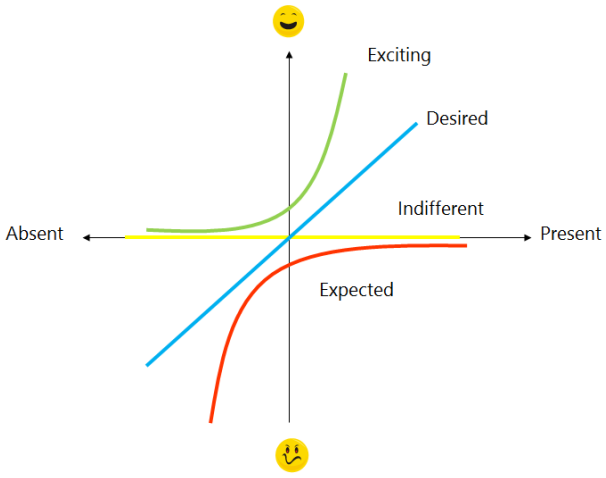

Figure 8

Expected requirements: The red line (the lower curved line) shows Expected requirements. Expected requirements do not result in stakeholder satisfaction no matter how much they are present. Their only result is to prevent

[3] See https://en.wikipedia.org/wiki/Kano_model.

dissatisfaction when fully present. If missing, the lack of these requirements causes great dissatisfaction. For example, suppose I were to book a hotel room online and, after arriving at hotel and going to my room, I discover that the door will not lock, the lights do not work, and the room lacks sheets and towels. My natural reaction would be great dissatisfaction, and I would let the front desk know I needed another room. It would not be acceptable for the front desk personnel to reply to my complaint with "But Mr. Simmons, you did not ask for those things when you made your reservation." These requirements were my *expectation,* and they comprise a part of the definition of the product known as a Hotel Room. Stakeholders do not tell you their Expected requirements unless they are not met, because they expect them to be present based on the underlying definition of the solution itself. Missing Expected requirements will cause you to miss market entry, simply because your solution will not be what stakeholders expect such a product to be.

Indifferent requirements: The yellow line (the horizontal line sitting on the Absent-Present dimension line) represents Indifferent requirements. These requirements are not valued by stakeholders at the present time, but they could be in the future. Returning to the hotel room example, for many years I did not drink coffee, so I did not place any value on how many types of coffee were provided in the room, nor its quality. Based on numerous studies showing associations between coffee consumption and various health benefits, I started to drink coffee, so now that aspect of the room matters to me, especially if no convenient coffee house is nearby. Stakeholders do not tell you their Indifferent requirements because (obviously) they are indifferent to them. You must decide what subset of all Indifferent requirements is worth the time and resources to track and manage. Broader trends in technology, ethics, law, health, politics, demographics, and other domains can be guides to the relevant choices. SenseMaker® is an excellent way to detect weak signals that could be harbingers of a shift in Indifferent requirements.

Desired requirements: The blue line (the diagonal line) represents Desired requirements, which are the first chance to create positive stakeholder satisfaction because the line extends above the x-axis. In the hotel room example, my choice to begin drinking coffee caused an Indifferent requirement (slope = 0) to become a Desired requirement (slope > 0). I also desire a low price, a comfortable bed, and a quiet, spacious room. The more these requirements are met, the more satisfied I am, and vice versa. Stakeholders *do* tell you their Desired requirements, because they are not expectations and they care about them.

Exciting requirements: The green line (the upper curved line) represents Exciting requirements, which are in many ways the inverse of Expected requirements. The absence of Exciting requirements does not create dissatisfaction because stakeholders do not even conceive that the Exciting

requirements are possible. But when Exciting requirements are exposed and met, the result is significant stakeholder satisfaction. I cannot provide examples of my Exciting requirements for a hotel room—by speaking them I would be stating Desired requirements. But I can imagine they might involve a chilled bottle of champagne in my room, along with a certificate for a massage after a long flight (at no additional cost, of course). Exciting requirements are often discovered by market research, observational studies, and various techniques within the practices of innovation (including SenseMaker® use).

Note that of these four types, only Desired requirements are spoken by stakeholders. The others must be known or discovered by you.

Expected requirements will *get you into the market*. If you miss expected requirements, your solution will be rejected and will fail in the marketplace.

Desired requirements will *keep you in the market* in the presence of competition.

Exciting requirements will *make you a market leader*.

You might find it valuable to look at your existing specifications and judge whether the amount of content fits your current market position. Many teams find that legacy Expected requirements are clogging their specifications, when the emphasis should be on Desired and Exciting requirements instead. This happens because there is a natural flow from Exciting to Desired and finally to Expected. For example, high-speed wireless internet connections were at first an Exciting differentiator for hotels, but they shifted to Desired and (in most countries) are now Expected to be present and free. Once again, we see that stakeholder value changes with time, re-emphasizing the need for Value-Driven Delivery to be based an evolutionary approach (iterative, incremental, and learning-driven).

A requirement's progression from Exciting to Expected also leads to another specification heuristic:

> **Heuristic: Abstract reused requirements when possible to reduce detail level and focus readers' attention on the new, risky, and complex requirements.**

Diffusion of Innovations

Diffusion of Innovations (DoI) is a mature field of study with nearly a century of practice. DoI has been used to study diverse areas ranging from adoption of family planning practices, new antibiotics, new crop strains, and technology-based products. Among DoI's many useful elements for Value-Driven Delivery are two in particular: diffusion populations and innovation accelerators.

Diffusion populations

DoI defines five populations: innovator, early adopter, early majority, late majority, and laggard. These populations are rather idealized, but they are a good way to conceptualize the overall market and the arc of adoption within it. The populations are shown in Figure 9.

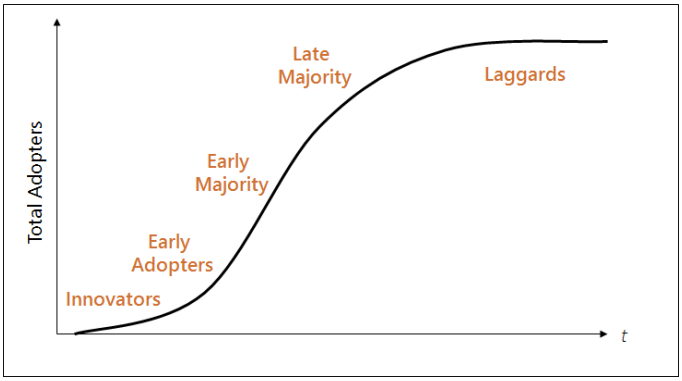

Figure 9

Innovators are the first small group to adopt. In the classic model, they represent 2.5% of the total. After that comes early adopters, who are the next 13.5%, and then the early majority, who make up 34%. It is the boundary between the early adopter and early majority that is often referred to as "the chasm." The final two populations are the late majority, who also make up 34%, and then the Laggards, who are the last 16% to adopt. Within this model, the cumulative adoption follows a familiar S-shaped curve.

The percentages within each category are all defined in terms of a Normal distribution, so we need to take the percentages as ideal, or a default. The model for innovation adoption in Figure 9 is a nice smooth curve, but real adoption is a very messy thing, usually showing little similarity with the model's predictable path.

The characteristics of each population are different. This means that what stakeholders in each category see as valuable will be different, so diffusion populations are helpful as we work in a value-driven way. For example, teams can fail when they mismatch features, functions, and capabilities to the adopting population.

In more Agile terms, the Minimum Viable Product for each population is different, and generally adopters in later populations will require more from the product than the preceding populations. This fact should influence the order in which we sequence our value deliveries and the scope of our releases, especially in mass-market products. For example, a team that attempts to define and release a Minimum Viable Product for the late majority could be quite late to market, well behind a competitor whose Minimum Viable Product is defined carefully for the innovators or early adopters.

Simplified characteristics of each diffusion population are given in the following table. Each of the diffusion populations has distinct needs and concerns that must be met before they will adopt something new.

Innovator	Daring: Desires new functionality and ownership as an end unto itself
Early Adopter	Respected: Opinion leaders who often adopt as an influence tool
Early Majority	Cautious: Desires a competitive advantage but requires a complete, reliable solution
Late Majority	Skeptical: Adopts on the trailing edge, based on price, convenience, and peer pressure
Laggard	Traditional: Adopts only when it can't be avoided, such as when older models are no longer available

Innovators are daring, desiring new functionality and ownership so much that they define themselves in terms of owning the latest and greatest. Innovators are not afraid of tackling technical issues or of a lack of product support. When they cannot find what they want from the market, innovators sometimes hack together a partial solution to their problem rather than wait for a solution to appear. They tend to be well off economically and tolerant of risk and the unknown. The Minimum Viable Product for innovators is smaller than that of later populations. New functionality or features alone can be enough to trigger adoption. Ecosystem development, reliability, product support, and even interoperability can be secondary. Knowing this can save time and money before first release, while also providing revenue from innovator purchases to fund further development. Innovators also provide valuable knowledge about how early aspects of the solution really work.

Early adopters are (or aspire to be) respected opinion leaders, who often adopt as a way to show or increase their influence. They tend to have better social networks than innovators, who can be rather isolated. As opinion leaders, early adopters are a frequent target for marketing efforts. They serve as role models for the majority that follows. The Minimum Viable Product for early adopters must be more complete and stable than that for innovators.

The **early majority** is characterized as cautious or deliberate. They take more time deciding whether to adopt than either of the previous populations. They need even more evidence of the efficacy, completeness, and reliability of the solution. Therefore, the Minimum Viable Product for the early majority must be more complete and stable than that for early adopters.

The **late majority** is characterized as skeptical. They are motivated by things like price and convenience, and they tend to adopt after experiencing sufficient peer pressure to do so.

Laggards tend to be quite traditional and slow to abandon an existing idea or solution for something new. They are beyond skeptical—they are downright suspicious of the new thing. They will adopt only when there is no viable alternative, such as when older models are no longer available for purchase.

Keep in mind that as individuals, we can be in various diffusion populations for different products or ideas. You might be an innovator in one area and in the late majority for another. And remember, this is just a brief overview of a rich literature. See *Diffusion of Innovations* by Everett Rogers (Free Press, 2003) for additional details.

The iterative, incremental, learning-driven approach of Value-Driven Delivery is a perfect way to meet the successive requirements of the various diffusion populations. By focusing on each population's specific values in order, the value deliveries match up with the natural course of adoptions, with each population finding the value it requires in the solution at the correct moment and funding the next phase of development via their purchases.

Innovation accelerators

In addition to the innovation populations, five innovation accelerators can be quite helpful in Value-Driven Delivery: relative advantage, compatibility, simplicity, trialability, and observability. According to Rogers, these attributes explain somewhere between 50% and 90% of the variation in adoption rates.

Relative advantage	Potential improvement to the current situation if adopted
Compatibility	How well the innovation fits with the culture and past practices of the organization
Simplicity[4]	How simple the innovation is to learn and use
Trialability	The degree to which the innovation can be sampled or tried in part
Observability	How visible the innovation is to other groups or individuals

[4] Rogers uses complexity in his book, but most people invert the concept so that all five accelerators have a "more is better" nature.

Relative advantage is the potential improvement to the current situation if adopted. Examples of this include some new functionality or capability, increased efficiency, cost savings, or improved quality; greater standing in the community, or enhanced personal identity or satisfaction.

Compatibility is how well the innovation fits with the culture and past practices of the organization. For example, how well the new innovation fits with existing routines, whether it reuses existing concepts or vocabulary, whether it requires new infrastructure elements, and how well it fits with the current culture and social norms.

Simplicity is just that—how simple the innovation is to learn and use. For example, how much training is required, complexity of any user interfaces, and the innovation's fit with people's mental models of the underlying activities.

Trialability refers to the degree to which the innovation can be sampled or tried in part. If an innovation requires several weeks of training for everyone, an expensive investment in tools, and a difficult, irreversible data conversion, people will be quite reluctant to adopt the innovation. Compare this to a situation where a small team can adopt the innovation in several pieces—with little training, low cost, and no significant risk to the broader community—to evaluate the innovation's benefits and feasibility. For example, an individual can rent a car he is thinking about buying, or a dealership can loan a vehicle free of charge while a customer's car is being serviced to entice potential buyers.

Observability refers to how visible the innovation is to groups or individuals, in terms of whether people recognize the underlying problem or opportunity the solution addresses, and the ability to sense the benefits the innovation creates. For example, companies might struggle to achieve Agile adoption when people are not aware of the problems that the current methodology is creating. This is especially true of groups near the periphery of an Agile adoption effort, where the benefits might be less direct than for other groups.

In Value-Driven Delivery, the five accelerators can help locate, analyze, and categorize stakeholder values. They can also suggest ways to improve a solution's adoptability, showing both the solution's strengths and weaknesses.

User experience proof points

User experience (UX) has become a major element of many solutions. Especially where competition is present, and at least moderate functional equivalence exists between competing solutions, user experience plays a strong role in purchase decisions. User experience is defined as *the thoughts, attitudes, emotions, and perceptions of the user before, during, and after*

use.[5] So UX is a very broad concept. Notice that UX is not the same as the user interface or as interaction design. Don't conflate those concepts with UX, despite that being common in the industry today.

There are seven UX "proof points," reflecting moments in the cycle from someone's first awareness of a product all the way to its disposal or replacement. The cycle of seven proof points is shown in Figure 10.

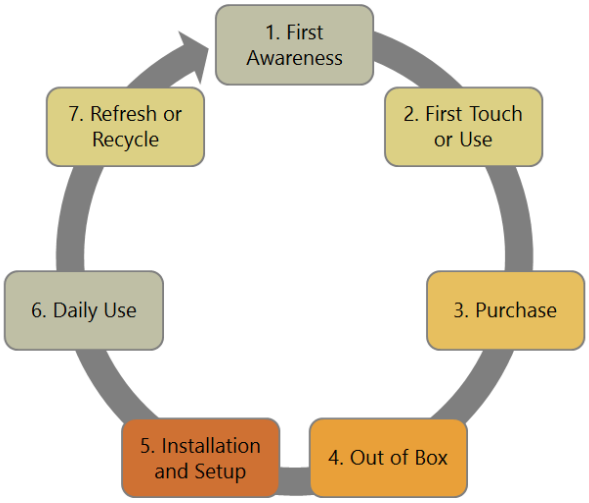

Figure 10

Each of these points is an opportunity for stakeholder value delivery, but many teams tend to focus only on daily use, with perhaps some consideration of the installation or time-of-purchase experience. Each of these seven points is a chance to differentiate the user experience of your solution from that of your competitors. Use the proof points to structure your discussions with stakeholders about what they value or to categorize your stories and requirements. The proof points also have uses beyond Value-Driven Delivery, such as a way to structure a set of use cases for your product. This is a simple model, but it is useful in many situations because the proof points span the entire solution life cycle.

The HEART framework

Originally created at Google,[6] the HEART framework for assessing user experience measures a user's **Happiness**, **Engagement**, **Adoption**, **Retention**, and **Task success**.

---

[5] This definition is very similar to other existing definitions, such as *The International Standard on Ergonomics of Human System Interaction*, ISO 9241-210.

[6] See *Measuring the User Experience on a Large Scale: User-Centered Metrics for Web Applications*, Kerry Rodden, Hilary Hutchinson, and Xin Fu:

http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/36299.pdf

Each of the dimensions is elaborated using a table of goals, signals, and metrics:

- **Goals:** High-level, conceptual results that motivate use of the system
- **Signals:** Observable indicators of goal satisfaction level
- **Metrics:** Obtainable, quantitative measures for each signal

The goal-signal-metric paradigm is similar to the classic goal-question-metric paradigm first published by Victor Basili, Caldiera Gianluigi, and H. Dieter Rombach in 1994. Both paradigms contain conceptual, operational, and quantitative levels in succession. In Value-Driven Delivery, the HEART framework provides useful categories for specifying and measuring stakeholder value delivery.

It is also possible to use the HEART categories as categories for Planguage statements. In that case, the Ambition, Scale, Meter, Minimum, Target, and Outstanding keywords can be used to specify stakeholder value within the HEART categories.

Summary

Value-Driven Delivery seeks the answers to three simple, but not at all easy, questions:

1. Who are your stakeholders?
2. What do they value?
3. What are you doing in the next two weeks or less to provide value to them?

Without a continuous, purposeful focus on value delivery, teams often end up emphasizing output without understanding how well the things they deliver satisfy their stakeholders and solve the underlying problems. Whether you are using a traditional sequential approach to development, Agile or Lean methods, or a hybrid approach, Value-Driven Delivery can create the proper focus from the executive suite to the individual contributor level and help you ensure that work is done in the order and manner that maximizes stakeholder value delivered.

Contributors

Erik Simmons, Senior Fellow
erik.simmons@construx.com

About Construx

Construx Software is the market leader in practical, research-based training and consulting that supports software professionals. Construx was founded in 1996 by Steve McConnell, respected author and thought leader on software development best practices. Steve's books *Code Complete*, *Software Estimation*, and other titles are some of the most accessible books on software development, with more than a million copies in print in 20 languages. Steve's passion for advancing the art and science of software engineering is shared by Construx's team of seasoned consultants. Their depth of knowledge and expertise have helped hundreds of companies solve their software challenges by identifying and adopting practices that have been proven to produce high quality software faster and with greater predictability.

- For more information about Construx's support for software development best practices, email us at consulting@construx.com or call us at +1(866) 296-6300.
- Sample Construx's OnDemand training for free at https://cxlearn.com.
- Review Construx's instructor-led training offerings at http://www.construx.com/seminars.

Construx®

SOFTWARE DEVELOPMENT BEST PRACTICES

Construx Software Builders, Inc.
10900 NE 8th Street, Suite 1350
Bellevue, WA 98004
U.S.A.