

Chapter 7. Lean-Agile Release Planning

If anything is certain, it is that change is certain. The world we are planning for today will not exist in this form tomorrow. —Philip Crosby

In preparing for battle I have always found that plans are useless, but planning is indispensable. —Dwight D. Eisenhower

In This Chapter

A major reason enterprises transition to Lean-Agile software development is the need to plan releases predictably and accurately. Release planning is the process of transforming a product vision into a product backlog. The release plan is the visible and estimated product backlog itself, overlaid with the measured velocity of the delivery organization. It provides visual controls and a roadmap with predictable release points.

Lean-Agile says the release plan must be driven by the needs of the business. We prioritize to maximize value to the business. We sometimes call this approach “business-driven software development.”

To understand how to do this, we must understand some fundamental concepts about process. Therefore, the chapter begins with a conversation about the issues that underlie process—predictability, level of definition, and requirements for feedback.

Takeaways

Key insights to take away from this chapter include:

- Release planning is a continual activity of the Lean-Agile enterprise. It is the transformation of the product vision or business case into a prioritized and estimated list of features.
- Once a feature list is created, its completion plan is determined by the velocity of the teams involved.
- Effective release planning requires a delivery organization that is skilled in predictable estimation, a skill readily gained by leveraging short-cycle iterations in order to get rapid feedback.
- Effective release planning emphasizes rapid return by focusing on discovering and manifesting minimum marketable features.

Issues that Affect Planning

One of the most frequent questions we get is, “How can you predict what is going to happen if you are working with an Agile process?” We believe that this question comes from a misunderstanding of some key issues that underlie process.¹

We think of processes as having the following:

- A degree-of-process definition; that is, to what extent the process has been defined
- A degree of predictability, or the randomness of its output
- A degree of feedback, or the amount of feedback that the process uses

While these issues often are inter-related, it is best to consider them as separate issues.

Evaluating Processes

Degree of Process

Let’s first clean up the terminology: We can view the output of a process as deterministic or non-deterministic (stochastic). In a deterministic process, the outputs are 100 percent determined by the inputs; in a stochastic one, the output is a random variable—it has different values that occur with different probabilities.

Fully-determined systems do not exist, except in academia and thought experiments. Virtually all real-world manufacturing and development systems have stochastic outputs. That is, they are partially determined.

It is useful to distinguish between a process that is fully determined versus one in which its *output* is fully determined. Although many people tend to assume that a defined process produces a deterministic output, this is not always true—a precisely defined process can still produce a random output. For example, the process for obtaining and summing the results of fair coin flips may be precisely defined; its output is a random variable.

Well-defined systems can produce outputs that range on a continuum from deterministic to purely stochastic. Just as we can structure a financial portfolio to change the variance in its future value—by ranging from all cash to all equity—we can make design choices that affect the amount of variance in a system’s output.

Degree of Predictability

Thinking of system output as a random variable may be more useful than labeling it as either unpredictable or predictable. We could think of it as completely unpredictable, macroscopically predictable, or microscopically predictable. It is unclear if anything falls into the first category—even a random number generator will produce uniformly distributed random numbers. It is the

¹ Special thanks to Don Reinertsen for an e-mail laying out many of these ideas. Used with permission, any inaccuracies should be considered ours.

zones of what we would call “macroscopic” and “microscopic” predictability that is most interesting.

We can make this distinction using the coin-tossing analogy. When we toss a fair coin 1,000 times, we cannot predict whether the outcome of the next coin toss will be a head or tail—we would call these individual outcomes “microscopically unpredictable.” There may be other microscopic outcomes that are fully determined since we have a fully defined process. For example, we could define this process such that there is a zero percent chance that the coin will land on its edge and remain upright. (If coin lands on its edge, then re-toss the coin.)

Even when the outcome of an individual trial is “microscopically unpredictable,” it is still a random variable. As such, it may have “macroscopic” or bulk properties that are highly predictable. For example, we can forecast the mean number of heads and its variance with great precision. Thus, just because the output of a process is stochastic, and described by a random variable, does not mean that it is “unpredictable.” This is important because the derived random variables describing the “bulk properties” of a system are typically the most practical way to control a stochastic process. That is, even though a process may be unpredictable on its own, it can still be controlled with feedback.

Degree of Feedback

The degree of feedback needed is another variable we should add to our duo of degree-of-predictability and degree-of-process definitions. In the software-development world, feedback is probably essential; in other areas it may not be. But for us, feedback is likely the most cost-effective way to achieve our goal—but deciding how and when to use it is really an economic issue.

Evaluating Processes

It is important not to confuse process definition with the level of determinism or the amount of feedback required to keep things on track. The key to this section is to understand that although we may not be able to predict microscopically the result of each story, we should be able to predict macroscopically the timing and the cost of the business capabilities encompassed in our features.

Transparent and Continuous Planning

Lean-Agile release planning is a continuous activity that the entire organization can observe. This makes it possible for anyone to contribute to discussions about the value of items in the plan and the effort required to produce them. Release plans enable delivery in small, end-to-end slices. This enables validation in a regular, predictable rhythm that is defined by the iteration length. As we described in chapter 4, Lean Portfolio Management, we want the product portfolio to serve as the transparent focal point for the business to sequence releases of minimal marketable features.

In all but the simplest cases, a feature requires several iterations before it is ready to be released to the customer. Reasons for this include

- The feature is too big to finish in one iteration.
- Multiple features may need to be released together in one package.
- The customer can only “consume,” or put to use, features at a certain pace or at a certain time of year.
- Marketing, training, support, and packaging for an otherwise completed feature will not be ready after a single iteration.

Release planning must account for all of these when developing the release schedule.

We think of release planning as continuously decomposing a product vision while focusing on those features of greater priority (value) to the business. This decomposition uses just-in-time methods to prevent wasted effort on lower-priority or unneeded features. That is, we expand on features just as much as we need to according to our expectations of when we will build them (whose order is determined by the value they provide to the customer). This plan enables the team to look ahead responsibly so that large-effort activities can be broken down in small enough segments (right-sized work) and balanced against higher priority items that come up. A good release plan provides a clear visual control and obviates the need to look too far ahead and work too far in advance on future, larger features. The continuous activity model is shown in Figure 7.1.

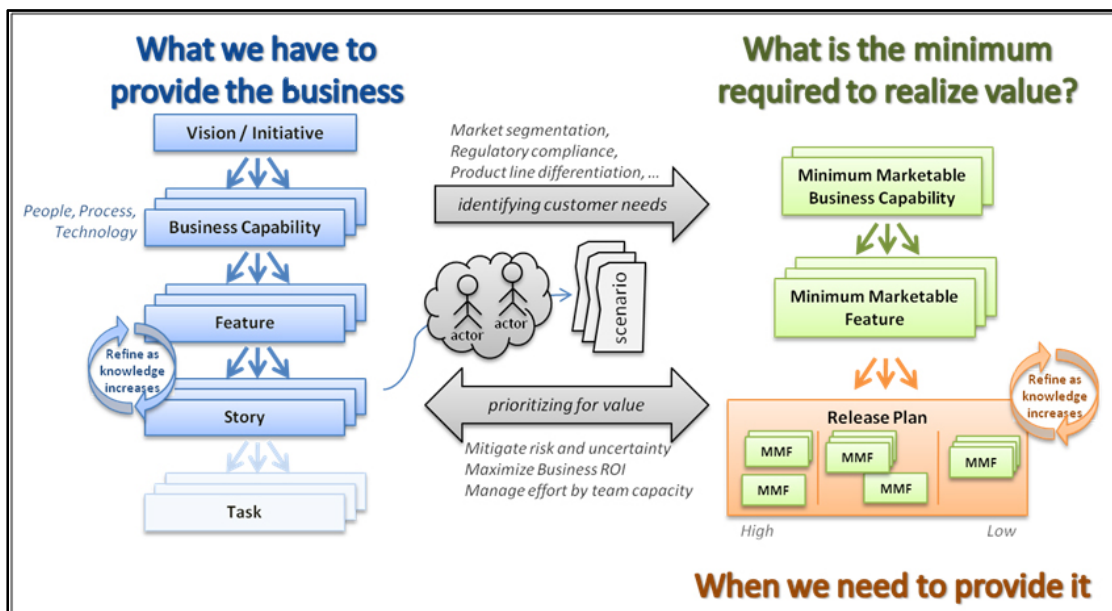


Figure 7.1. The continuous activities involved in release planning

Release planning starts with a vision provided by the Product Champion, who can make decisions regarding value priority for both the customer and the business. We typically look to the organization that creates project charters to find ideal candidates for this role. The vision

should be reviewed and understood by the delivery team and should be revisited as market conditions change priorities. The vision should be visible (for example, with posters on walls) and re-reviewed as part of every iteration's planning session.

Target dates are determined by looking at the estimates in relation to the team's velocity. For example, if a team can deliver 40 story points in a two-week iteration and we have 200 story points to achieve, we can fairly estimate that it will take five two-week iterations to complete the work at hand. Short cycle times (one to four weeks) enable quick feedback on both the rate of completion and how well we are meeting our customers' needs. During each iteration, teams must focus on coding only the most important feature at any one time. This provides a clear picture of business value (features) juxtaposed against system constraints (technical stories) and enables high-value decisions regarding minimum releasable features.

A project charter should make a business case for new capabilities or capability enhancements. We look to these capabilities to find business features, or "features." It is important to realize that features derive from the vision and capabilities; they do not appear by aggregating lower-level requirements into larger chunks, which is sometimes suggested in the literature as the creation of "epics." Trading business value against effort in search of minimum marketable features leads to decomposing capabilities to form features or stories.

To perform Lean-Agile release planning effectively, the development organization must visually establish (and continuously improve) its ability to determine velocity (story points per iteration), as described in chapter 4, Lean Portfolio Management. The visible velocity is a powerful measure of enterprise capacity (see Figure 4.13). This approach requires that the delivery organization be skilled in the art of three-level story point estimation (feature, story, task). Here is another opportunity to emphasize the importance of short cycle time (two-week iterations): The organization is able to recalibrate the quantity associated with story points, as well as get feedback and institutional learning regarding how complex the capabilities, stories, and tasks are.

These multiple levels of continuous decomposition enable an organization to provide estimates required for creating a visible release plan predictably and fearlessly. This is especially worth noting when estimates are required at the feature level, when the least amount of information is known. Experienced Agile teams are confident in providing estimates because the precision required for large features is low, and they know that they are required to commit only when features have been broken down at least two more levels (stories and tasks), and then only commit to two-week iterations with known tasks (which should be about four hours in size). In a transition to Lean-Agile, allow three to four iterations for this skill to mature well enough to produce reliable release plans. Table 7.1 shows the various levels of requirements, their sources, and estimation units.

Table 7.1. Various levels of Top-Down Requirements Utilized in the Lean-Agile Approach.

Requirement Level	Description	Source	Units
Feature	Business solution, capability or enhancement that ultimately provides value to the business and/or its customers	Business/customer value, charter document, business case	Story Points
User Story	Describes interaction of users with the system	Feature	Story Points
Story	Any requirement that is <i>not</i> a user story (e.g. technical enabling, analysis, reminder to have conversation)	Development team, analysis work, large story decomposition	Story Points
Task	Fundamental unit of work that must be completed to make progress on a story	Development team (during iteration planning)	Hours

The rate at which teams complete features can be measured in average story points completed per iteration. This provides a velocity of production. After a few iterations this should converge to a somewhat steady rate. If it doesn't, the teams need to investigate why it hasn't yet happened. Once a reasonable velocity is established, it can be used to estimate delivery dates of the releases. Prior to this, release planning will need to rely on comparing current work to the amount of time it took to perform similar work in the past.

In practice, it is never possible to focus on only one feature at a time. Some features may require longer lead times due to dependencies and waiting to complete system-enabling work. WIP should be constrained by an overall focus on the delivery of features (as opposed to the completion of tasks). The constraint is naturally held to because the visual control would quickly expose a feature that is too large. The mature organization continuously challenges the need for large features to find the minimum scope required to deliver maximum return. Metaphorically, this means that sometimes the business value requires only a "bicycle," while the development organization is creating a "motorcycle." In organizations that exhibit enterprise Agility, visible release plans serve as catalysts for communication, where business value and technical constraints are continuously decomposed and visible along with multiple options based on effort and value. The end result is an organization that incrementally demonstrates and evaluates the value of the release, one feature at a time. A business develops true Agility when it can make real-time verification that what it has built meets the minimum scope required for the feature to deliver its intended value. This is achieved by continuously fighting the waste that comes from building too much. The resulting increase in speed of delivery now enables the organization to meet the demands of rapidly changing markets, customer needs, and business opportunities.

Depending on the release structure of the organization, dedicated release iterations may be required to actually deploy the product to the enterprise production environment. It is an acceptable practice to have a so-called “release iteration” for this activity. It is important that this iteration is constrained to the minimum amount of time required by the release organization, and it should be used only to perform activities required for sign-off and compliance of the release acceptance organization (no new scope).

Releases and Elevations

In an ideal world we could release—straight to the customers after every iteration. Unfortunately, for many reasons, this is often impractical. For example, if you are on a team that builds embedded software, you may need to create an internal release for the integration team (a team that tests your software, and possibly others’ as well) on a hardware platform. Or you may build code that another team will use, so you’ll need to release it internally, to the other teams. There are also times you’ll need to release code to selected customers to get feedback—possibly as an alpha test, but maybe just to play with.

We have coined the term “elevation” for all of these “releases” that are not quite real. We don’t use “internal release,” as they sometimes go to customers, but they are not the real releases.

Example: Release Planning Session

This section describes a typical release planning session. Such sessions often follow a sequence like this:

1. Identify features.
2. Prioritize features.
3. Split features using the minimum marketable feature perspective.
4. Estimate the value of the features.
5. Estimate the cost of the features.
6. Elaborate further by writing stories for features, repeating until you have reasonable clarity on what the features are and their high-level values.
7. Create a specific release plan by date or by scope.
8. Plan elevations.

How long does a release-planning session take? Small projects (three months or less) can often be done in a day. Larger projects will take a few days.

During the session, the team has to remember constantly that it is being driven by two forces:

Using tools in release planning. We want tools to support the Lean-Agile process. The early stages of release planning, though, are best supported with lower-tech, higher-touch tools: everyone present in the room, using stickies or index cards on the wall.

This creates the best environment for the non-linear, multi-dimensional thought processes release planning requires.

Once the release plan has been created, it is good to move the data into an Agile planning tool.

- **Add value for the customer.** The focus is not on building software; it is to increase its value to those who will use the software product we create. The software is a means to an end, but it is not the value itself.
- **Get to market quickly.** Develop plans around minimum marketable features (MMF). View features from the MMF perspective: What is required to develop and release them?

Using Tools in Release Planning

We want tools to support the Lean-Agile process. The early stages of release planning, though, are best supported with lower-tech, higher-touch tools: everyone present in the room, using stickies or index cards on the wall.

This creates the best environment for the non-linear, multi-dimensional thought processes release planning requires.

Once the release plan has been created, it is good to move the data into an Agile planning tool.

In the following sections, we examine each of the steps in a bit more detail.

1. Identify Features

Begin by writing features on stickies or index cards. Briefly describe each feature (usually just a few words), as shown in Figure 7.2. At this point, the team is just trying to establish a high-level scope of the system.



Figure 7.2. Initial features

2. Prioritize Features, Left-to-Right

Once the features are identified, the team does an initial prioritization: Place the most important features on the left and the least important on the right, as shown in Figure 7.3. This only represents a first cut; the team is not committed to this order. It will certainly change as they work through the steps.



Figure 7.3. Initial features, prioritized, left to right

Even this initial prioritization should prompt some interesting conversations. The conversations should be focused on helping everyone learn more about the features and sharing knowledge of the features. Don't get hung up on whether the prioritizations are absolutely correct. Focus on learning as much as possible and hold all decisions as tentative.

3. Split Features Using the Perspective of the MMF

Once the initial set of features is described, it is often easy enough to split up some into what could be called minimum marketable features and then further split into one or more enhancements to those MMFs.

For example, suppose Feature F in Figure 7.3 must be supported on five different platforms: Linux, Windows, Solaris, HP, and AIX. Talking with the customer, the team discovers that only Linux and Windows need to be supported at first. Feature F can be broken into two parts: the core MMF for Linux and Windows and an extension MMF for the others. Call these F1 and F2, respectively. Other features can likewise be decomposed, as shown in Figure 7.4.



Figure 7.4. Splitting features up into an MMF and its extension

4. Estimate the Value of Features

Since the Product Champion is driving from business value, the first thing to do is estimate the relative value of each feature. We can do this using the Team Estimation Game.² The value of each story is assigned business-value “points” (shown as “BV” in Figure 7.5). However, do not reorder the features based just on these points. Features may have to be developed in a particular order or you may need to get a sense of the cost required for each business value.



Figure 7.5. Assigning business value to the features

You may find that you have difficulties quantifying features by points this way. In this case, just identify the general sequence in which the features need to be built. We have found that many organizations cannot initially set values to the core, required features. In some sense, this doesn’t matter: They will all need to be built before release anyway. If that is the case, don’t worry about it. You should find that, after the release of the core MMFs, you can set relative values for the remaining features.

Remember: Business or customer value is independent of cost. First, determine business or customer value and only then ask the team to estimate the cost. Then, you can calculate ROI.

5. Estimate the Cost of Features

You can use the Team Estimation Game to estimate the cost of the features which are represented in “story points” (shown as “SP” in Figure 7.6).

²Appendix A, Team Estimation Game, contains a description of the Team Estimation game, which we prefer over “Planning Poker.”



Figure 7.6. Assigning cost in story points to features

Once you have the cost for each feature, the product team may decide to reprioritize them. In effect, you now have the capability to evaluate Return (business value) on Investment (cost), which enables new insight into selecting what brings the highest return to the business for the effort spent by the delivery team. A significant value of this technique is that it clearly decouples business value prioritization from technical effort, which is an opportunity to drive from business value first. We find that most business organizations have lost the ability to prioritize based on business value alone because they are so used to batching up large requirement sets with faraway dates that they see no need to sequence features since “they are all important.”

6. Elaborate Features

You might be surprised at how well this approach works at a high level. It works by comparing one item against another—something teams are reasonably good at. Going further requires more accuracy. This requires a more detailed understanding of the features.

Start by writing stories for each of the features, beginning with the higher-priority features, the ones you will be working on sooner. This is called “elaboration.”

After elaborating a few features and increasing your understanding of what is required to build them, you may need to re-estimate both business value and cost. (Notice that this technique has a built-in feedback loop that continuously calibrates the accuracy of the feature estimates. The elaborated stories for each feature are individually estimated and then summed to compare with the feature.) Continue this process until you have a set of features comprised of the core and extension MMFs, along with a number of elaborated stories, and you are confident in the relative estimates of the features. This is shown in Figure 7.7.

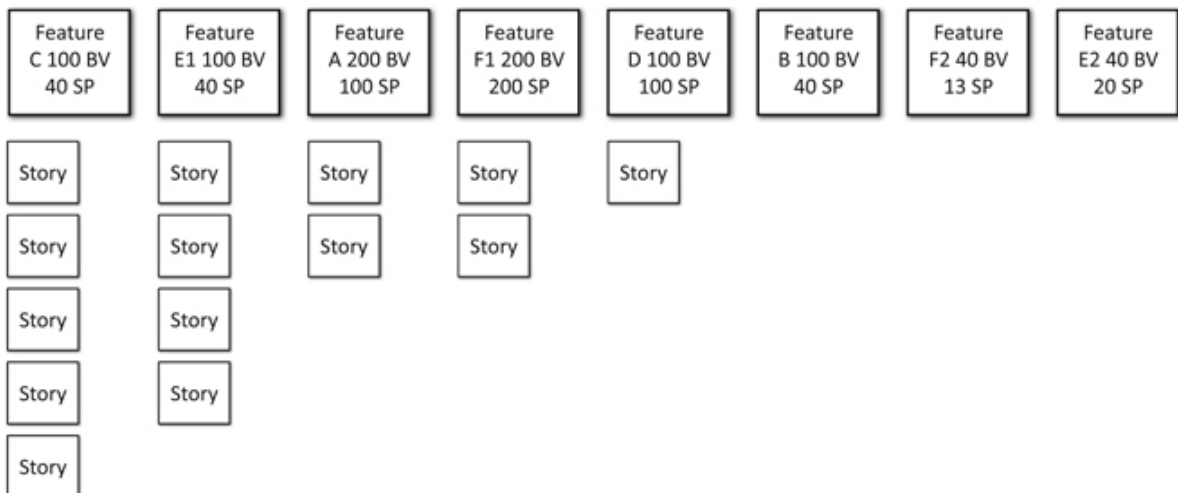


Figure 7.7. Result of Feature and story elaboration

7. Create the Release Plan

Now the team is ready to plan releases. There are two approaches to this: planning by date and planning by scope. Which to use depends on your needs, which are often mandated by governmental regulations or market conditions.

Planning by Date

There are times when a project must come in by a certain date: Government regulations require certain features by a certain time, software is required for a conference, or our industry requires major releases at a certain time of year. If this is the case, then the release plan entails setting the date and ensuring the right amount of functionality can be achieved within the allotted time.

For example, suppose you have four months to finish product development and every feature except B, F2, and E2 is required by that date. The release plan is shown in Figure 7.8.

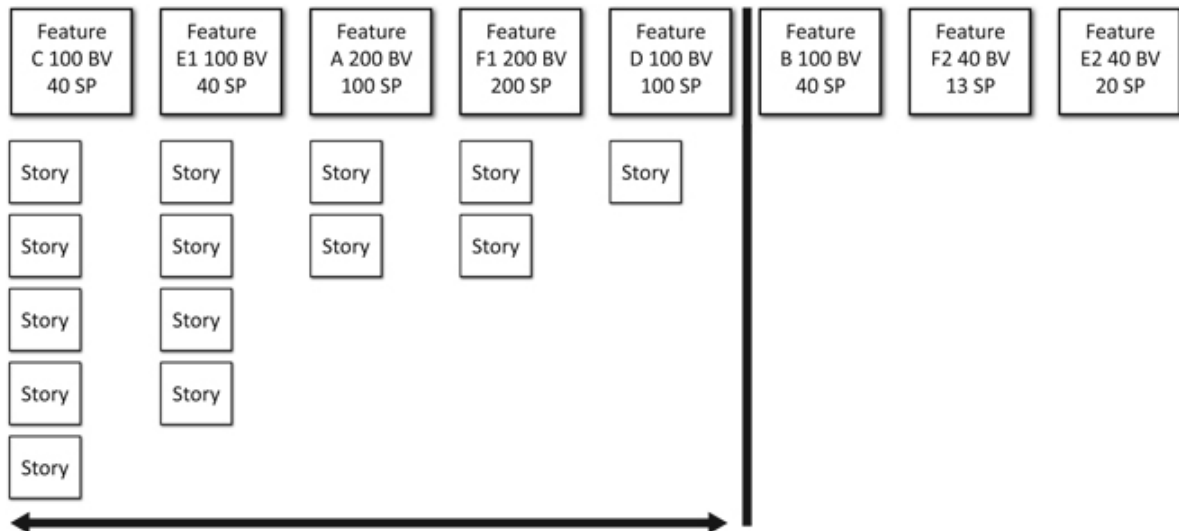


Figure 7.8. Planning by date

Add up the estimated number of story points for these features. That is how many points must be completed in each iteration. In this example, there are 480 story points. There are 17 weeks available. Suppose Iteration 0 requires a week at the beginning and there are two weeks at the end for alpha testing. That means 480 points over 14 weeks for development, or 34 story points per two-week iteration.

Total Points / Number of Weeks Available for Development = Required Team Velocity

If the team can handle that level (velocity), that is great. If not, you have to focus on what is truly minimal for each of the identified features. What can be cut out? What must be left in? At the beginning, you cannot know for sure, which is why the focus must be on starting work on only the features, or aspects of features, that are truly essential. Iterative development will enable you to discover the core functionality need.

This type of estimation does not necessarily give you better accuracy than traditional methods. But it does show you where you need to look to make your decisions. Very often it becomes clear that the true MMFs can be built in time, whereas you are uncertain about features you would just *like* to have. Sometimes, it becomes clear you are in trouble. If you are somewhere in the middle, then at least you have an idea about which features you need to investigate further.

Planning by Scope

Another approach is to plan by scope. This works much like planning by date; however, this time you begin with those MMFs that are required. Calculate the number of story points in the MMFs, divide by the team's velocity (the ability to complete stories in an iteration) and the result is the time required to do the work.

Total Points / Team Velocity = Number of Weeks Required for Development

If the result is too long, reassess to see what features or elements can be dropped to make it shorter.

Proper planning avoids risk

Both of these approaches help teams focus and avoid risk. They help teams:

- Work on the most important features
- Avoid starting less-important features until the most important ones are finished
- Minimize WIP

These are crucial. Consider a time when you were working on a project only to discover you were going to have to cut scope. The predicament is that at this point, you have:

Agile Estimation Isn't Exact, But It Is Better. In our classes, we are often asked how we can get precise estimates with Agile methods. This question seems to imply that the asker is somehow getting these desired accurate estimates with his or her non-Agile method. We don't claim that using Agile methods will improve accuracy over non-Agile estimating at the very beginning. It will, however, create clarity at a faster pace. But when it comes to the claim that we must be accurate, we are reminded of the following joke: Two campers are awakened in the middle of the night by the sounds of a bear snuffling at their tent entrance. One calmly starts putting on his shoes. The other exclaims—“Are you crazy? You can't outrun a bear!” The other responds—“I don't have to outrun the bear, I only have to outrun you!”

- Already completed some less-important features—which you started because at the beginning of the project you were confident it was all going to happen; and
- Started some features you would like to cut but doing so now would cause you to lose work you’ve already done—you’d have wasted time and added complexity for no value (almost certainly the code that’s in there for these features will stay in there).

Planning-by-date and planning-by-scope methods help ensure that the team works on the most important features known at the time and that other features are not started until the important ones are finished.

8. Plan the Elevations

There may be another degree of complexity to consider when there is more than one team involved in the software development or there is a subset of the software that can be tested but cannot yet be released.

The first case can be made more difficult if there is hardware on which to test as well. In these cases, an internal release is necessary to test the system—either its technical integrity through integration testing or its quality to customers through external system testing using alpha or beta testers. We call these pseudo/external releases “elevations.” We are moving the software farther down the value stream, but not all the way to the customer. We will consider two different types of elevations.

Elevations for Integration Testing

Very often a team will build software must interact with software that other teams are building. You cannot be sure exactly how it will function until the other teams use it. Or teams are creating multiple products that must be used on a hardware platform. Until the software is actually on the hardware, you cannot know for sure how it will function.

A Case Study

Company profile: Large software product company

Challenges: Tightly coupled, complex product enhancements being built at the same time. Not clear of the exact scope of features.

Insight: During a planning session where all related features were put on a wall and all interested parties were present, one of our consultants asked the question—“how many people here are 100% certain that all of these features will be built in the timeframe we have?” To no one’s surprise, no one raised their hand. Then the consultant asked—“which of these features must be done by the deadline or you don’t have a product?” There was actually fairly consistent agreement on this question. These were the features selected for the first release.

Lean suggests doing the essential things first in the fastest time possible by building quality in. By de-scoping early, we focus on the Pareto Rule of 20% of the work providing 80% of the value. By time-boxing our development, we minimize the affect of Parkinson’s Law that “work expands so as to fill the time allotted for its completion.”

One type of elevation planning is to look at the milestones the software must reach prior to true release. In a situation like this it could be

- Software passes a team’s functional test.
- Software passes several teams’ functional test.
- Software works on a specified hardware platform.
- Software has been alpha-tested by a set of customers.

This example would require three elevations prior to the final release:

2. Move the software to the other teams that will use it.
3. Load and test the software on the hardware platform using internal testers.
4. Enable external users to try out the software.

These elevations are shown graphically in Figure 7.9.

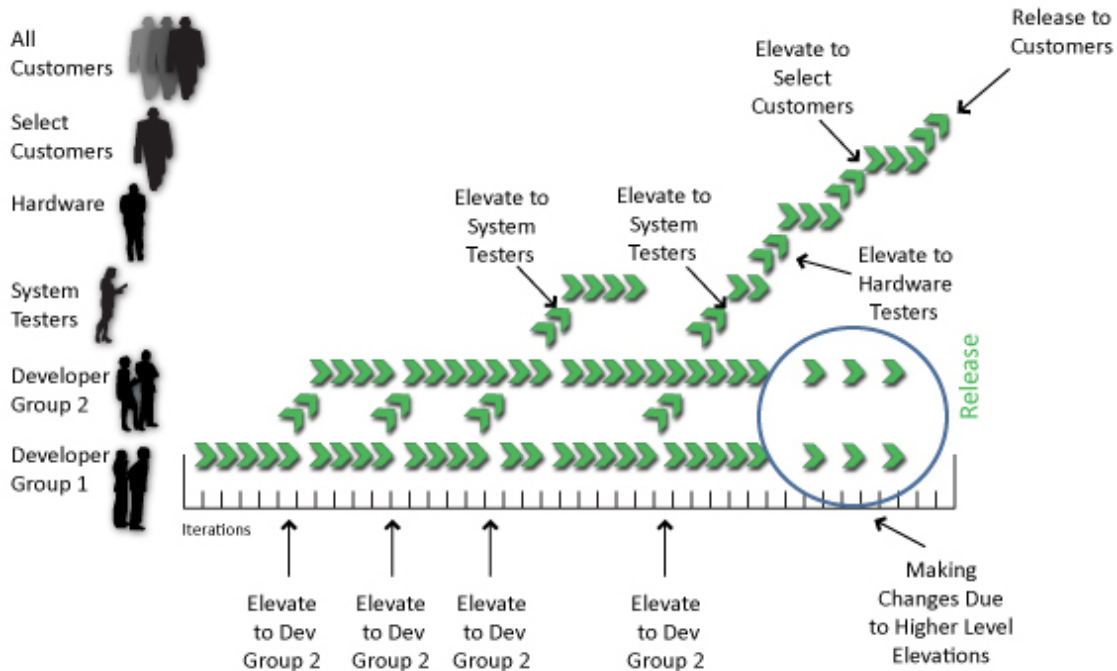


Figure 7.9. Elevations across teams and testing platforms

Elevations to Different Platforms

A different elevation pattern exists when the software you are writing must work on different operating systems. For example, suppose you are writing software for Windows, Linux, and mobile platforms. Figure 7.10 illustrates that elevation plan.

The following is an excerpt from *Lean-Agile Software Development: Achieving Enterprise Agility* by Shalloway, Beaver, and Trott. No portions may be reproduced without the express permission of Net Objectives, Inc.

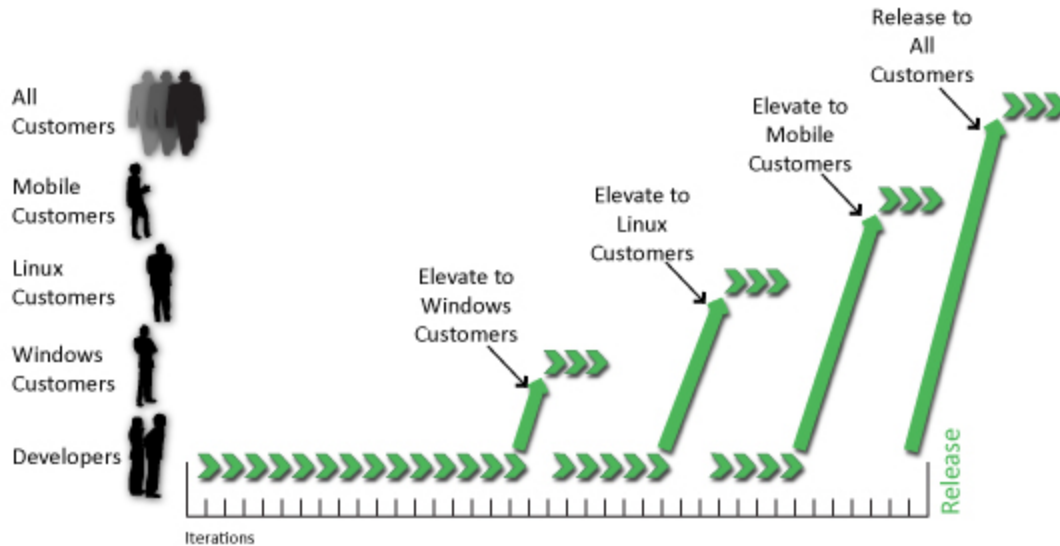


Figure 7.10. Elevations to different operating systems

Elevation Summary

There are no set rules for elevations. The ideal case is continuous integration across all development. But when different platforms, operating systems, hardware, customer bases, and so on are present, that is not always possible. Elevation planning, however, enables you to investigate the best way to get feedback about a larger, working part of the system. Acceptance Test-Driven Development with an emphasis on design patterns and refactoring enables the organization to benefit holistically from emergent design techniques. For example, skilled organizations that mock to test and refactor to design patterns can do more in-place and continuous integration than would be required to incorporate Lean-Agile in complex-release organizations that deliver across different platforms. Chapter 9, *The Role of Quality Assurance in Lean-Agile Software Development*, covers this in more detail.

A Few Notes

We end this chapter with a few more release-planning thoughts on estimation and risk—and Pareto versus Parkinson.

On Estimation and Risk

Many people think that there is risk attached to missing your estimate. At worst it might be embarrassing; however, the real risk is in missing your *delivery dates*. It is not important to be able to predict at the level of the story; what is important is predicting at the release level.

Risk also plays a role in prioritizing features. Usually, we prioritize by the business value each feature represents—possibly offset by the cost of creating it. However, sometimes prioritization is affected by the potential cost of delay. For example, let's say we have Feature A and Feature B. Feature A may be twice as important as Feature B, but we need Feature B for a conference

coming up in three months. We may actually do Feature B first to ensure its completion before the conference if delaying Feature A is not too costly.

Pareto vs. Parkinson

We have heard Lean software development likened to following Pareto's Law: 80 percent of the value comes from 20 percent of the work. In other words, find that 20 percent of features that will provide your customers with 80 percent of their value; then, find the next features that will provide the greatest value to your customers.

The problem with this is that if there is no time-boxing—no end-date—Parkinson's Law may apply: "Work expands so as to fill its time for completion." Parkinson's Law is particularly dangerous when multiple product managers are competing for a team's resources. Manager A is focusing the team on one thing and Manager B is concerned about when she will have the team's availability. You can counteract the effect of Parkinson's Law, by having the team follow Pareto's Law in the shortest amount of time they can. In other words, have the team always focus on building the smallest things as quickly as they can, end to end, while ensuring quality.

Add the most value possible in the least amount of time possible with the right level of quality.

Summary

An organization that maintains visible release plans that are driven by continuous validation of velocity have a powerfully competitive weapon—key tactical and strategic moves can be analyzed continuously for maximum value. Enterprise Agility is achieved when the delivery organization is actively engaged in the release planning activity, through estimation and the discovery of options based on effort.

Try This

These exercises are best done as a conversation with someone in your organization. After each exercise, ask each other if there are any actions either of you can take to improve your situation.

Consider a few typical past projects.

- Most successful Waterfall projects require de-scoping in order to reach target dates. If this was the case for any of your past projects, when did de-scoping occur?
- What would have happened if de-scoping would have occurred *before* the development team started implementation?
- How does release planning (with visible velocity) aid in the discovery of right-sized, high-value work?

The following is an excerpt from Lean-Agile Software Development: Achieving Enterprise Agility by Shalloway, Beaver, and Trott. No portions may be reproduced without the express permission of Net Objectives, Inc.

Recommended Reading

The following works offer helpful insights into the topics of this chapter.

Denne, Mark, and Jane Cleland-Huang. *Software by Numbers: Low-Risk, High-Return Development*. Upper Saddle River, NJ: Prentice Hall PTR, 2003.

Reinertsen, Donald G. *Managing the Design Factory*. New York: Free Press, 1997.

NET OBJECTIVES LEAN-AGILE APPROACH

INTEGRATED AND COHESIVE

All of our trainers, consultants, and coaches follow a consistent Lean-Agile approach to sustainable product development. By providing services at all of these levels, we provide you teams and management with a consistent message.

PROCESS EXECUTION

Net Objectives helps you initiate Agile adoption across teams and management with process training and follow-on coaching to accelerate and ease the transition to Lean-Agile practices.

SKILLS & COMPETENCIES

Both technical and process skills and competencies are essential for effective Agile software development. Net Objectives provides your teams with the knowledge and understanding required to build the right functionality in the right way to provide the greatest value and build a sustainable development environment.

ENTERPRISE STRATEGIES

Enterprise Agility requires a perspective of software development that embraces Lean principles as well as Agile methodologies. Our experienced consultants can help you develop a realistic strategy to leverage the benefits of Agile development within your organization.



Contact Us:
sales@netobjectives.com
1-888-LEAN-244
(1-888-532-6244)

*We deliver unique solutions
that lead to tangible improvements in software development
for your business, organization and teams.*

SERVICES OVERVIEW

TRAINING FOR AGILE DEVELOPERS AND MANAGERS

Net Objectives provides essential Lean-Agile technical and process training to organizations, teams and individuals through in-house course delivery worldwide and public course offerings across the US.

CURRICULA — CUSTOM COURSES AND PROGRAMS

Our Lean-Agile Core Curriculum provides the foundation for Agile Teams to succeed.

Lean Software Development

- Implementing Scrum for Your Team
- Agile Enterprise Release Planning
- Sustainable Test-Driven Development
- Agile Estimation with User Stories
- Design Patterns

In addition, we offer the most comprehensive technical and process training for Agile professionals in the industry as well as our own Certifications for Scrum Master and Product Champion.

PROCESS AND TECHNICAL TEAM COACHING

Our coaches facilitate your teams with their experience and wisdom by providing guidance, direction and motivation to quickly put their newly acquired competencies to work. Coaching ensures immediate knowledge transfer while working on your problem domain.

LEAN-AGILE ASSESSMENTS

Understand what Agility means to your organization and how best to implement your initiative by utilizing our Assessment services that include value mapping, strategic planning and execution. Our consultants will define an actionable plan that best fits your needs.

LEAN-AGILE CONSULTING

Seasoned Lean-Agile consultants provide you with an outside view to see what structural and cultural changes need to be made in order to create an organization that fosters effective Agile development that best serves your business and deliver value to your customers.

FREE INFORMATION

CONTACT US FOR A FREE CONSULTATION

Receive a free no-obligation consultation to discuss your needs, requirements and objectives. Learn about our courses, curricula, coaching and consulting services. We will arrange a free consultation with instructors or consultants most qualified to answer all your questions.

Call toll free at 1-888-LEAN-244 (1-888-532-6244) or email sales@netobjectives.com

REGISTER PROFESSIONAL LEAN-AGILE RESOURCES

Visit our website and register for access to professional Lean-Agile resources for management and developers. Enjoy access to webinars, podcasts, blogs, whitepapers, articles and more to help you become more Agile. Register at <http://www.netobjectives.com/user/register>.