Chapter 6. Time Management

Time management includes the processes required to accomplish timely completion of the project.

—PMBOK® Guide

Time is neutral and does not change things. With courage and initiative, leaders change things.

—*Jesse Jackson*

A predictable organization does not guess about the future and call it a plan; it develops the capacity to rapidly respond to the future as it unfolds.

—*Mary Poppendieck*

Physicists, scientists, mathematicians, and philosophers have debated the idea of time for thousands of years. Is it an unchanging presence in our universe, or is it a separate entity somehow entangled with space? Is it merely a measurement imposed by man, an illusion?

What we know in the project management realm is that time is an important element to be managed. But can we really manage time? Can we stop time, engage time, elongate time, or shrink time? Can we reward time so that it performs better for us? Can we report on time in order to better understand its quality? Time, then, in project management, is an imposed constraint by the business or the customer; managing time really means that we manage the activities and deliverables that occur within that given constraint in order to produce the best possible results. This means that the business can decide to extend or bring in dates, depending on the project's progress, market conditions, or other internal or external factors.

Many software projects traditionally have been managed by what we call a skill/phase-based approach; first, analysts analyze the problem, designers design a solution, coders code, testers test, and then the product is released. That is, the collection of people with skills equates to the different phases of the project; project controls are put in place to manage each group of skilled people, among other things. In this waterfall approach, the analysis and design phases provide an end date for the project; the assumption is that by understanding the work and breaking it down into small enough pieces (tasks), the tasks can be estimated and the end date of the project can be calculated. This approach to managing software project schedules is flawed for several reasons.

First, the "industrial making approach," or serial handoff of a deliverable from one set of skill groups to the next, works well for a repeatable process with a predictable outcome.[1] In other words, if I am a deck builder and I offer a prepackaged solution for a 12'x12' wooden deck, I know when to schedule permits with the township, when to schedule the backhoe and post setter, if needed; I know exactly how many pieces of wood and the number of screws I need, as well as how many hours and men that each job requires. Bad weather not withstanding, this kind of project schedule is easy to create in a prescriptive fashion. In fact, my ability to predict milestones, dates, and hand-offs would be quite reliable.

Although some forethought and insight is necessary to discover points of synchronization, procurement deadlines, and funding milestones, among other events in software development projects, the actual day-to-day

crafting of software is much too complex to predict and control. What happens in one day—in one hour, even—can impact dependent tasks significantly, and the thousands of permutations of outcomes are impossible to capture in a project schedule. The tasks involved with building software, day to day, cannot be predicted and certainly are not repeatable. From one software project to the next, requirements change, technology may or may not be stable, and people bring another level of complexity to the table. Attempting to predict task-level detail in such mayhem is waste, and formulating a date off of this flawed prediction is setting up a project team for failure.

Finally, change happens. Whether the customer asks for a new feature or simply a change to a screen color, many teams have realized that software developed in a series of empirical trials in collaboration with the customer is best to address these changes. Build, learn, build again, learn more. Through these attempts, teams make the right product. Thus, this "artful making"[2] approach to software development suggests that a project end date is very difficult to perfectly predict because we cannot perfectly predict the features. We simply cannot know today exactly how the system will be built; sure, we have a vision, a general direction, but we cannot say exactly how it will operate, or look, or feel to the user. Only by developing in repeated trial-and-error passes can we begin to see the system evolving and how we need to carry it forward. Agile builds these checkpoints into the iterative process.

The flawed thinking that we can predict a system and its capabilities, coupled with frequent and necessary change, should cause us to reconsider our estimation processes and date commitments. This chapter illustrates how to "manage time" by providing visibility into project results so that the team and the customer can negotiate an outcome. We'll call it forecasting instead of prediction. And we'll think of estimates—whether at the strategic or tactical level—as forecasting tools, not commitments written in stone.

So, how do we forecast in an agile project, then, while simultaneously delivering on the dates that are important for the customer? First, we must recognize that delivery is a partnership between the customer and the delivery team. Then, we start top-down. The customer provides the team with a product vision and a roadmap, which are standard practices, agile or not. The team then determines a release date based on a given set of features, or alternatively decides which subset of features it can deliver within a given timeframe. Time management is then a negotiation between the customer (or the business) and the team from one iteration to the next, based on product feedback, changing market conditions, and insight into how rapidly a team can create working features (known as "velocity"). As agile project managers, we make project progress visible at the iteration and release levels, and as you'll discover in this chapter, we allow the team of experts—Drucker's famous "knowledge workers"[3] —to manage themselves on a day-to-day basis.

This chapter differentiates between strategic and tactical planning and provides you with ideas about how to plan an agile project using this paradigm. Additionally, we give you the context in which detailed activity decomposition and estimating occurs, while helping you understand how you can leverage the team's results to report project status, thus enabling the business or the customer to make educated decisions about the emerging product. As you will see, combining high-level release plans with detailed iteration plans addresses complex project planning with the right amount of detail at the right time.

Strategic Versus Tactical Planning

We've concluded that time management really is a factor of managing the activities and deliverables that occur within a given time constraint in order to produce the best possible project results. There are ways that we can more intelligently plan our agile projects so that we can set realistic milestones and facilitate negotiation between the business/customer and the delivery team.

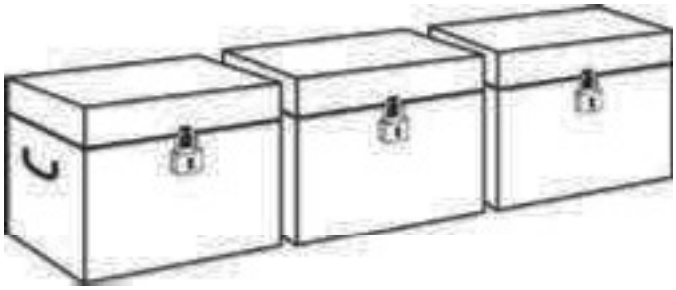
There is a common analogy in business management that is referred to as strategic and tactical planning. As you may know, a strategy is a long-term plan of action designed to achieve a particular goal. A tactic is a method

employed to achieve a certain goal. The goal pursued by strategic planning is at a higher level of abstraction than the goals pursued by tactics. Strategic planning in agile projects, then, occurs at the release/feature level upfront and throughout the project, whereas tactical planning at the task level occurs during iteration planning and is adjusted as needed, on a daily basis, throughout the iteration by the team.

Thinking in terms of strategic and tactical planning is certainly not a new concept, but we feel that explicitly adding both sets of planning practices into your project planning toolkit is something that can help you better manage time expectations with your customers and organization.

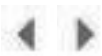
Think of an agile software project consisting of multiple boxes made of steel; we stop dropping in feature bricks when each box is full (see Figure 6-1). We then close the box, padlock it for that iteration, and work the features through to acceptance. Then, we move on to the next box. This is different from software projects that follow the traditional approach where there is only one big box into which we stuff (and often overstuff) feature bricks. The duration of an agile project is measured by how many steel boxes of bricks are necessary to complete the vision for the product. It's the steel box that we're putting our features into, and steel is not bendable and cannot burst at its seams. Furthermore, because we are collaborating with our customers and have the opportunity to release working code faster, customers don't feel the need to overstuff the box. There can always be another box if necessary. Strategic planning happens at the "how many steel boxes do we need to complete our features" level. The team engages in tactics when figuring out how it's going to build each feature within a timebox.

Figure 6-1. The software project as multiple steel boxes



The imposed time constraint in a project is a stake in the ground; it represents the business or customer's need. Ultimately, the customer may decide that more iterations of development are necessary or that rebudgeting for more resources is possible. It is the agile project manager's responsibility to balance the business needs against the delivery team's capabilities, keeping progress visible so that collaborative decision-making about managing time is possible.

We'll start with developing and controlling a project plan at the strategic level (the release plan), and then illustrate how the team plans its tactics for an iteration.



Release Planning: Developing the Schedule at the Strategic Level

The *PMBOK® Guide* refers to schedule development as "an iterative process [that] determines planned start and finish dates for project activities. Schedule development can require that duration estimates and resource estimates are reviewed and revised to create an approved project schedule that can serve as a baseline against which progress can be tracked."[4] The traditional project schedule is the primary tool to measure time and other project performance; the agile equivalent is the agile release plan. Instead of an iterative process that determines start and finish dates for activities, as is done in traditional project schedule planning, the agile release plans determine finish dates for features, which correlate to an iteration's end date.

As you learned in previous chapters, agile release planning is a process in which we gain an understanding of the features that will comprise a collection of iterations. We achieve this knowledge via top-down (or strategic) planning. We do this because we embrace the fact that change will occur in the course of our project, and we save ourselves the weeks (or sometimes months!) identifying every detail upfront. Many teams have found agile release planning to be at the right level, in between prescriptive, plan-driven project schedules and the inappropriate "you'll get it when you get it" mantra of some freshman agile teams. Moreover, because the release plan is expressed as a timeline of which features will drop into each iteration, the customer more easily understands project schedule and progress much more readily; it is exactly the depth of information they're interested in, because features represent value. Tasks, which represent *how* something is built, usually aren't interesting to the end consumer of the project's product.

The Release Plan: Schedule Development at the Strategic Level

The agile release plan helps us focus on a three- to six-iteration window of time. If you have a year-long project, for example, you may want to consider release planning on a quarterly basis, with a high-level project roadmap to guide the overall initiative. Once the project is underway, updates to the release plan should be made at the end of each iteration based on the actual progress and the remaining work for the release. See Chapter 5 , "Scope Management," for additional details about how to conduct release and roadmap planning meetings.

Additionally, consider the way agile teams estimate at the strategic level. Because we've dispelled the myth that we can predict long-term complex initiatives at the task level, creating a strategic plan in hours and days is not the appropriate level of estimation. Rather, many agile teams will use abstract measurements to estimate product backlog items for strategic planning.

The most common abstract measurement is story points, which can follow the Fibonacci sequence: 1, 2, 3, 5, 8, 13, 21 (another common expression is an exponential scale of 1, 2, 4, 8, 16, 32, 64...). It is common for features in the product backlog to be written as "user stories," which is a format for writing requirements from a user's perspective.[5] For example, rather than stating "the system shall...," a user story would say "as a <type of user> I want to be able to do <feature/function> so that <I get value>."[6] These features or user stories are then estimated in a unit of measure called "story points"; assigning a story point "estimate" to a user story says, "We think it's about this complex." All other user stories are then assigned story points in relation to the first one that's estimated. This relative complexity is a way of giving an indication of difficulty at a very high level. A sense of how many story points a team can complete in one iteration lends the ability to forecast the number of iterations a set of features will take to implement.

Regardless of your unit of measure, the team's iteration length, or the involvement level of the customer, the agile release plan has a fundamental, critical assumption: that the functionality delivered every iteration is of the highest quality possible. If the team is delivering code that's not tested and integrated every iteration, there remains an indeterminable amount of work due to the unknown product quality, which is risky. However, if the team frontloads as much testing work as possible during the iteration, so that the incremental deliveries are all of the highest quality, the team may only need a short

hardening iteration at the end of the release in order to release the product to production. Additionally, it has mitigated the risk of the unknown by testing early and often.

While we're on the subject of different iteration types, many teams schedule an Iteration 0 (zero) at the beginning of the project, and an Iteration H (hardening) as the last iteration in a release (to production) cycle. Iteration 0 is used for project approvals, environment setup, ramp-up, discovery and initial overviews, and design discussions. Iteration H is used at the end to prepare for delivery, and it includes activities such as finalizing training and marketing materials and preparing the golden master or installation/download files. Iteration H is not used as a testing cycle; in fact, analysis, design, and testing should happen continuously throughout the iterations themselves, verified by the customer once he signs off on a story as being "done" for the iteration. Rather, Iterations 0 and H are used as minimal bookends for work that is not explicitly new development. Additionally, and very importantly, we firmly believe in asking our teams to deliver at least one feature in Iteration 0 because we've seen many teams revert to form, utilizing Iteration 0 as purely a design or analysis phase.

Figure 6-2 illustrates two release possibilities—internal and external—when thinking about release plans. When teams stay disciplined in delivering working product every iteration, the organization theoretically has the ability to package up and "release" what's been built. Whether or not the package is actually released to customers depends on a number of factors, but it could just as well be released internally to training or support services or externally to a partner for a limited user-acceptance testing cycle. Regardless of the use, sticking to quality increments of product will provide the organization the agility it desires for whatever need is most pressing.

Figure 6-2. A release plan is a high-level plan for a series of iterations. Releases can be internal or external.

[View full size image]

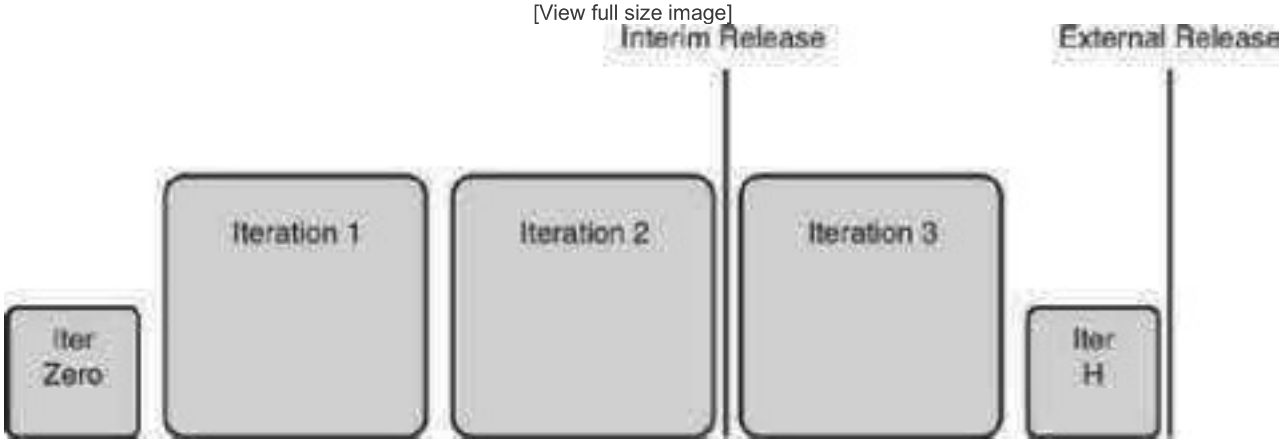


Table 6-1 provides a summary comparison of traditional and agile approaches to schedule development at the strategic level. Agile schedule development at the strategic level is referred to as "release planning."

Table 6-1. Schedule Development at the Strategic Level

Traditional	Agile
Create the final project schedule (usually in a tool that shows bar charts, milestones, and a network diagram) and baseline.	Facilitate the collaborative creation of the release plan, resulting in a quarterly high-level plan indicating release goals, features to be completed, and timeboxes.
Update the activity attributes, resources, project plan, etc., as appropriate.	Assist the team in updating the release plan as needed, based on iteration results.

The Release Plan: Schedule Control at the Strategic Level

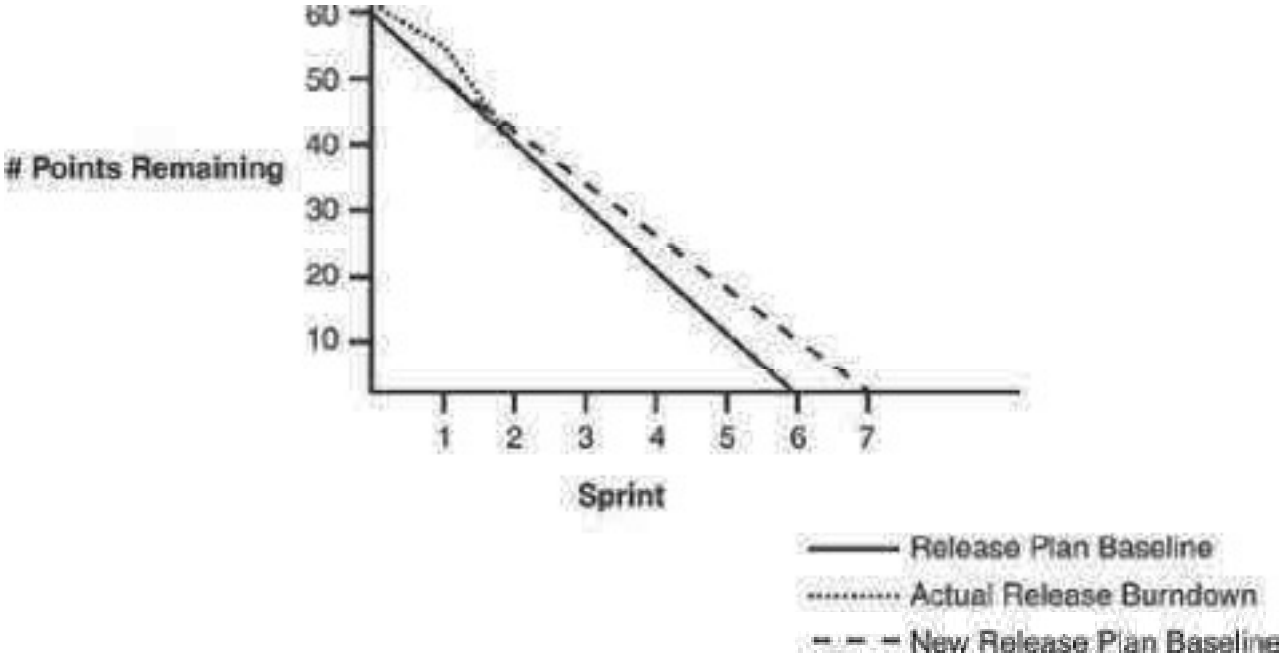
Schedule control is carried out by knowing the current status of the project schedule, and managing changes in the plan that might impact the schedule. The agile release plan is managed in a similar fashion, except that the plan remains high level instead of containing detailed tasks from beginning to end; additionally, the agile project manager facilitates the customer and the team in making schedule changes, if necessary, based on project visibility. We will tell a story to illustrate this concept.

During a release planning meeting, our fictitious team, whom we'll call "GERT," analyzed its previous work and determined that it could reasonably commit to a pace of ten points per iteration, each iteration being four weeks in length. The product owner really needed the release six months later, and he was interested to know about how many features he would receive in the release. The team went through a collaborative exercise of placing user stories into iterations by using sticky notes and flip chart paper, "filling up" each iteration (or steel box) with no more than ten points' worth of stories in each. The GERT team stopped once it filled the final iteration with features, and realized that it had chosen a subset of the product backlog equal to roughly 60 points (10 points x 6 iterations = 60 points). This set of features, called the "release backlog," can be used as a baseline by which project progress is measured. Remember, however, that we want to allow appropriate change as the customer provides product feedback. Thus, "managing" the project schedule in an agile setting is really about making progress visible so that the customer can balance the original product vision along with the necessary changes in scope that arise out of natural product evolution.

Back to our team. After the first iteration, the team only completed seven story points; velocity was lower than initially expected because the team ran into some unexpected environment setup issues. The project manager, team, and customer discussed the situation and decided to extend the release plan to seven total iterations, versus the six originally predicted, in order to account for the lower velocity. At this point, the customer could have just as well reduced the amount of scope in the release backlog, but he decided that he had a little wiggle room in the overall release end date and could float another iteration. When discovering that actual velocity is higher or lower than expected, the product owner can decide if he wants to reprioritize user stories, drop user stories from the release backlog, or extend the release by *n* iterations.

The GERT team completed 12 story points of work in the second iteration. The customer was delighted! Even though the team and the customer were happy about the increase in velocity, they decided to keep the extended date just in case they needed the buffer. Figure 6-3 shows how the team began with a commitment to 60 points over six months using four-week iterations and baselined this. Then the burndown of what the team delivered each iteration was noted, resulting in the decision to commit to a new baseline.

Figure 6-3. This release burndown chart represents the GERT team's progress through iterations 1 and 2 of its project.



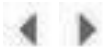
Managing a number of iterations in a release plan is dependent on the real-time status of the working product, as well as on the collaboration and negotiation between the customer and delivery team. The agile project manager facilitates this discussion and makes certain that both sides have the information each needs in order to reach a collaborative decision about the release.

As new discoveries are made about the system, any changes that need to be made can be done by adding an item to the product backlog. In Agile, it is expected that these changes are necessary in order to build the right product. The product and release backlogs therefore function as a natural change management system.

Table 6-2 provides a summary comparison of schedule control at the strategic level for the traditional and agile approaches to project management. In agile terminology, schedule control at the strategic level is referred to as "agile release plan management."

Table 6-2. Schedule Control at the Strategic Level

Traditional	Agile
Update the schedule based on approved changes and re-baseline.	Facilitate the review meeting where the team updates the release plan based on the team's velocity and changes in the backlog.
Calculate schedule variance and schedule performance index.	Remind the team of its duty to update the work remaining in the iteration backlog; at the end of every iteration, the release burndown chart is updated. The team also measures its velocity.
Track and document requested changes.	The customer updates the product and/or release backlog.
Identify and analyze recommended corrective action to get the project back on track.	Facilitate a discussion between the customer and the team to discuss any corrective action that should be taken (i.e., add another iteration, add a team, drop features, terminate the project, etc.).



Iteration Planning: Developing the Schedule at the Tactical Level

Iteration planning brings us inside the steel box. It is the meeting in which the team breaks down the items from a prioritized product or release backlog into small tasks and assigns estimates and owners sufficiently so that the team can agree to the iteration plan. It is in this meeting that the activities (tasks) are defined, sequenced, and estimated by the team. Like the steel box, the iteration end date is fixed and padlocked; the team, based on its past performance (that is, velocity) will choose enough work to fill up the box. The iteration planning meeting can take up to eight hours for a month-long iteration, or up to four hours for a two-week iteration.

An iteration planning meeting includes the delivery team, the project manager, and the customer (or customer's representative, often referred to as the "product owner"). Others may attend the meeting if they wish to observe or contribute supplemental information about the features to be built.

The output of this meeting is the iteration backlog, which is a list of the team's tasks and other information necessary to manage its day-to-day tactics in turning product backlog items into working product increments for the iteration. A main difference between the release backlog and the iteration backlog is its unit of measure for the estimates: release backlog items—or features—are measured in story points or other high-level estimates, whereas iteration backlog items—or tasks—are estimated in hours.

Because the timeframe is so narrow—one to four weeks—teams become quite proficient at predicting the work within the iteration. Teams perform the traditional planning processes of activity definition, sequencing, and estimation in the planning meeting and throughout the duration of the iteration as needed, leveraging the daily stand-up meeting to gain valuable insights that may affect the tactics.

Activity Definition

The *PMBOK® Guide* classifies activity definition as "identifying and documenting the work that is planned to be performed."[7] In traditional project planning, a project manager would think about all the tasks that might ever happen during the course of the project as part of documenting the expected work to be performed. In fact, project managers often have at hand their methodology templates or project structures used previously that contain the work breakdown structure and subsequent tasks; the project manager simply inserts the new project name and makes a few adjustments. This reuse of a previous project plan is in line with traditional thinking, that new product development can be predicted and repeated. As we know, planning in an agile project should account for the change that will happen and involves teams working as a cohesive unit, task-sharing their way to iteration success.

Thus, activity definition happens in an agile project when the iteration has arrived in which the feature will be worked on; this breakdown occurs in the iteration planning meeting by the team, not the project manager. The iteration planning meeting occurs on the first day of every iteration; it is supported by the *PMBOK® Guide* and is referred to as "rolling wave planning."[8]

To prepare for the iteration planning meeting, the customer brings the list of highest-priority features and no more. The team should have already looked ahead at this list in order to estimate the complexity of the high-priority items. The customer will discuss what the requirement is all about, how he or she will consider it is sufficiently implemented (known as acceptance criteria), and answer any questions that the team has about the intent and purpose of the requirement. Think of this first part of the meeting as the "What" and "Why" of iteration planning. This detailed understanding of the business case or user intent is imperative for accurate and reliable activity definition.

During the second part of the meeting, the team will determine all of the tasks necessary to complete the requirements that

the product owner considers highest priority—the "How." The team will discuss details such as, What stored procedure do we need to write? What user interface changes are necessary? Do we need another table in the database? Will performance be impacted? As the team members flesh out the answers to these questions, they will begin to make note of their tasks, as well as how long each of the tasks will take. A very popular way to record this information is on sticky notes arranged on a task board. Sometimes, electronic tools are used, but remember the simplest, most visible solution is often best.

The ultimate goal of iteration planning is to have a set of activities (or tasks) that the team will perform within the boundaries of the padlocked timebox (or iteration). This resulting list of tasks, regardless of the means by which it's captured, is referred to in agile as the "iteration backlog." Most importantly, the team derives its own tasks and its own sense of what it can accomplish by the iteration end date. Figure 6-4 depicts a team's task board, a community location in which cards representing tasks are placed on the board and moved to the column that represents the feature's status. Task boards are very popular with co-located teams.

Figure 6-4. An iteration task board

[View full size image]

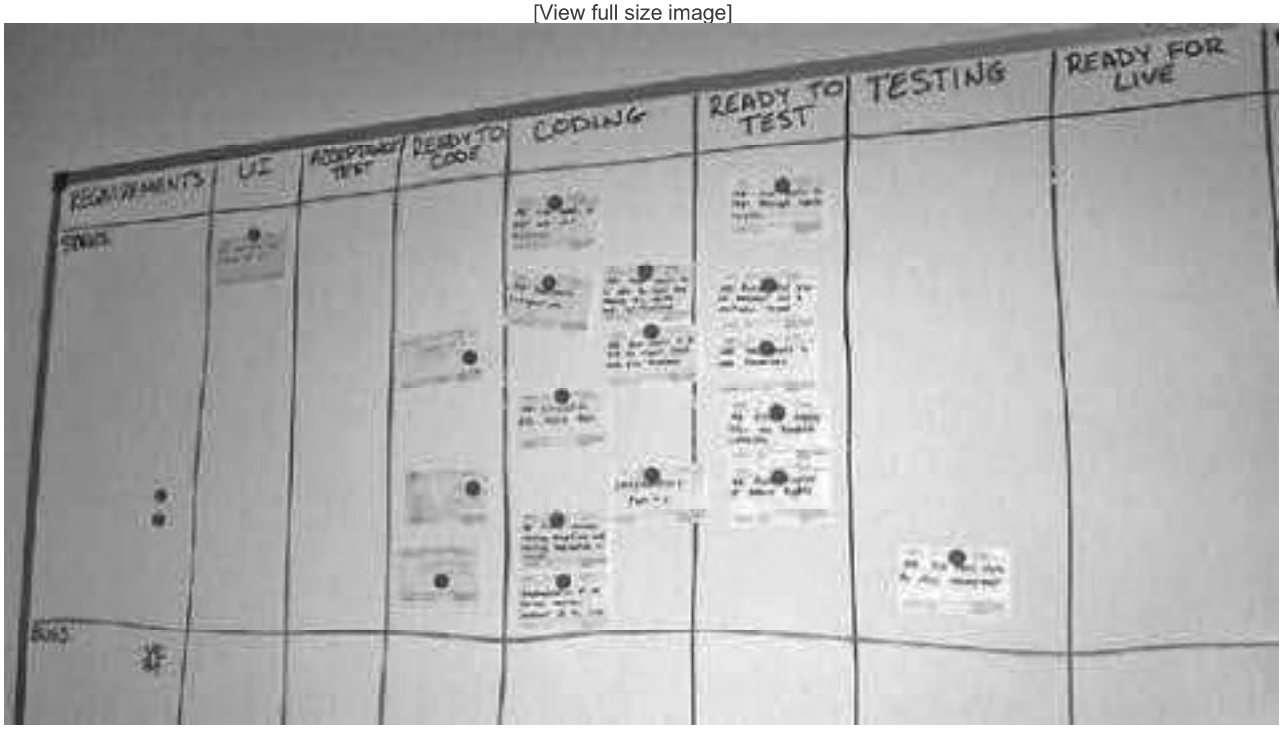


Table 6-3 presents a summary comparison of traditional and agile approaches to activity definition. In the agile vernacular, activity definition is referred to as "defining and tasking out user stories." It is done by the team as part of developing the schedule at the tactical level.

Table 6-3. Activity Definition

Traditional	Agile
Prepare an activity list showing all scheduled activities to be performed on the project.	Each iteration, the team will break down the selected features into a list of tasks that become part of the iteration backlog, or iteration plan.
Define activity attributes such as task owner, predecessors, successors, constraints, assumptions, level of effort, etc., and include in the project schedule.	Activity attributes such as task owner and estimate are defined during the iteration planning meeting and included in the iteration plan.
Milestones are defined for the entire project and inserted into the project schedule.	Themes are identified for each iteration during release planning; often, project managers refer to the increment of working code delivered at the end of each iteration as a milestone.
Requested changes resulting from the activity definition exercise are put through the change request process.	If new requirements are discovered during iteration planning, the customer should capture this information by adding the item to the product backlog. Sometimes the new requirements are included in the iteration that is being planned, but only if the customer agrees to remove lower-priority features from the iteration and place them back into the product backlog, so that the steel timebox is not compromised.

It is important to note that the *PMBOK® Guide* also explains that activity definition "will identify the deliverables at the lowest level in the work breakdown structure, which is called the work package. Project work packages are decomposed into smaller components called schedule activities to provide a basis for estimating, scheduling, executing and monitoring and controlling the project work."[9] In an agile project, this is done on an iteration-by-iteration basis, and for the most part, the work package equates to the feature, not to individual tasks.

Activity Duration Estimating

Agile project managers don't track duration of activities. In fact, duration is removed from the activity altogether. Duration is represented by the padlocked iteration timebox; it has definitive start and end dates that do not change. The team, then, decomposes features into tasks, which are then estimated in terms of work effort hours, and doesn't plan for more tasks than what it can do during the timebox. The estimates are aggregated and the team members compare this to the amount of work they estimate they can do (their capacity based on hours). They also make sure that the total number of story points doesn't greatly exceed what they've completed in previous iterations. Newer teams usually also compare their task hours to each team member's individual capacity; more mature teams have embraced role-sharing and thus have devised ways of thinking about capacity as a whole. In fact, for some advanced teams, the high-level product backlog estimate alone suffices to answer the question, "Is it in or out for this iteration?" Many teams work through the product backlog of features, in order of priority, and do not move to the next feature until the current one is finished—by the entire team!

Sometimes, the team discovers during iteration planning that the sum of the task efforts exceeds the size of the iteration timebox. When this occurs, some of the work needs to be shifted either into the next iteration or back into the backlog. After adjustments are made to the iteration's scope, the team can commit to the iteration—the agreement to padlock the timebox.

Estimates Get Better with Experience

I had to drive to Ensenada, Mexico, one summer. To get there, I crossed the border in San Diego and headed due south. Not having driven in Mexico much, and in fact, not at all in any country that uses kilometers as the measure of distance, I was quite interested in the sign that read "Ensenada, 90km." Now, when driving in the States, if I see a sign that says "35 miles," I know that at my current rate (which is normally around 70 mph), I can reach my destination in 30 minutes. In Mexico, however, I didn't know what 90km "feels" like, and I couldn't quickly do the math to convert distance to time. So I had to just say, "Well, it's 90 and it will take me however long it will take me to get there." Once I saw the next sign that said "Ensenada, 40km," I could begin to predict my arrival based on how far I had come and how long it had taken. This "speaking in kilometers" exercise occurred to me to be similar to how we measure feature complexity in story points. Story points are a measure of complexity. Based on our "burn rate"—or in this case "driving rate"—through creating functionality, we can predict within a normal range of certainty how long our requirements will take us to complete. And the *huevos rancheros* after driving a long day through a product backlog are worth it!

After a series of iterations, a team's ability to predict the amount of work it can accomplish becomes more precise. This is because the team has established an average velocity, has figured out how to work together, and has learned about the emerging system.

Table 6-4 provides a comparison of the traditional and agile perspectives on activity duration estimating. Activity duration estimating is referred to in the agile world as "task estimating," as part of developing the schedule at the tactical level.

Table 6-4. Activity Duration Estimating

Traditional	Agile
Estimate activity durations.	Team members provide estimates for tasks during iteration planning and throughout the iteration when new tasks are needed or when existing tasks change.

Activity Sequencing

Activity sequencing "involves identifying and documenting the logical relationships among schedule activities."[10] Activity sequencing in a traditional project can be quite complicated, with "relationships" between activities ranging from finish-to-start, to finish-to-finish, to start-to-finish, to start-to-start. These relationship types are used in traditional project scheduling to keep track of what happens first, second, and simultaneously, and is especially useful in creating automatic project schedules. Knowing when one activity can start as well as knowing if certain conditions must be met for another activity is important. This is relevant for managing projects by both traditional and agile methods. The difference in agile is that we don't worry about this level of detail until the iteration is at hand.

The trouble with pinpointing activity sequence before the project starts is that a software project's tasks are numerous and are subject to change; once requirements or tasks change, we have to try to mirror these changes in the project schedule. This takes an incredible amount of time and is often inaccurate. The main difference with agile activity sequencing is that it is performed by the team in iteration planning, and changes throughout the iteration based on daily team synchronization

as it discovers more details about building the feature, based on ad-hoc discussions or new information.

During iteration planning, the project manager and the team are aware of the iteration start and end dates, and they also keep in mind other important dates or milestones (beta program, training, and so on). They can then plan their activity sequencing based on this high-level information. Sometimes the team will discover a technical dependency in the product backlog during iteration planning, at which time the team will let the customer know about it. At the task level, however, the team will figure out the dependencies and the order in which work needs to happen. A great litmus test is to have a round-robin exercise before committing to the iteration in which each team member answers two questions: First, what are you going to work on tomorrow when you come in? And second, what's the next thing you're going to work on and with whom do you plan to work? This exercise helps by kick-starting the first few dependency discoveries; after that, the team will discover more during the iteration.

It is also helpful if the project manager brings a visual to help during planning. Often, we have found that just simply drawing a calendar on the whiteboard to show the days during the iteration is helpful. It serves as a memory jog; this information helps teams think about not only the work, but other things to consider as well. Another example of this is shown in Figure 6-5 , which is an actual project manager's rendering of the iteration calendar that he would draw on the whiteboard to remind the team of important milestones or meetings. This would sometimes cause a reordering of tasks.

Figure 6-5. Timeline for iteration task sequencing

It is important to note that there are two types of dependencies in agile projects: overall project dependencies and task dependencies. The team members, because they know more than the managers about the details of the work they are performing, can best manage the task dependencies. The project manager can bring the overall project dependencies to the table. For example, if I know that my project team is awaiting a third-party code drop by the end of December, I can help manage that external dependency and shield the team from the burden. However, the fact that Bob is waiting on Joe to make the database schema changes is something that I will entrust the both of them to handle. As long as Joe does this in time for Bob to finish coding and testing by the end of the iteration, that is all I am concerned with.

Table 6-5 compares activity sequencing from the traditional and agile perspectives. There is no specific agile term for activity sequencing. It is simply part of developing and finalizing the schedule at the tactical level.

Table 6-5. Activity Sequencing

Traditional	Agile
Define activity sequencing for the project for display in project schedule network diagrams.	Facilitate iteration planning, where the team will determine how to handle task sequencing. The team and customer together determine how technical and external dependencies might affect the order of the items in the product backlog.
Updates to activity lists, attributes, and change log are captured and processed accordingly.	Updates to the iteration backlog in response to discovered or resolved dependencies are maintained by the team.

Activity Resource Estimating

Resource estimating in traditional project management is often determined after the activities have been defined.

This is reversed in agile. That is, a team's members (resources) are determined upfront and are dedicated to the project. The stray resource coming in and out of the project team is an exception (think DBA or third-party integration specialist). This is because qualitative studies have shown that individuals are ultimately more effective if dedicated and not time-sliced across multiple endeavors; subjectively, the dedicated teams we have worked with display much higher morale than their time-sliced colleagues.

Teams experience an increase in efficiency and productivity when allowed to work in a dedicated fashion toward delivering the iteration's features. Additionally, most teams that are together for a series of iterations will begin to think like an organic group. Instead of thinking linearly, as in "I'll code and you test and Pat will write documentation," it becomes "Let's do this together." You'll see team members jump in and start learning new skills. It's a switch from linear thinking to team thinking. It's a big jump here for the project manager to not think in terms of loading resources into the tasks of a project schedule and instead having teams that are cross-functional and dedicated and "loaded" into iterations.

Human resources, then, are not estimated to fill the timebox; rather, the team comes along with the timebox, and the team chooses however many features it can fit within its allotted time for the iteration.

There are material resources to consider as well—such as servers, credit card readers, switches, and so on. In agile, we can discover these items during the iteration, but there's also another time for this discovery—in agile release planning. Taking a high-level look at the expected or desired functionality over a series of iterations allows us to know when certain material resources will be necessary. For example, a team I was working with knew it would need a data architect as well as two new servers for a new data warehouse, but this wouldn't be necessary for another four months based on the release plan. The team was able to forge ahead while the project manager made the necessary decisions to procure the contractor and the equipment in time for that stage of the project.

Table 6-6 provides a summary comparison of activity resource estimating from the traditional and agile perspectives. There is no specific agile term for activity resource estimating. It is simply part of developing the schedule at the tactical level as a dedicated cross-functional team.

Table 6-6. Activity Resource Estimating

Traditional	Agile
Identify the resources required for each work package.	The team is dedicated to the iteration and is known before the iteration begins. Other resources can be planned for at the agile release planning level.
Create a calendar showing resource availability and needs.	Remind teams to think of the times when they won't work on features during the iteration—that is, holidays, training, leave of absence, and so on.

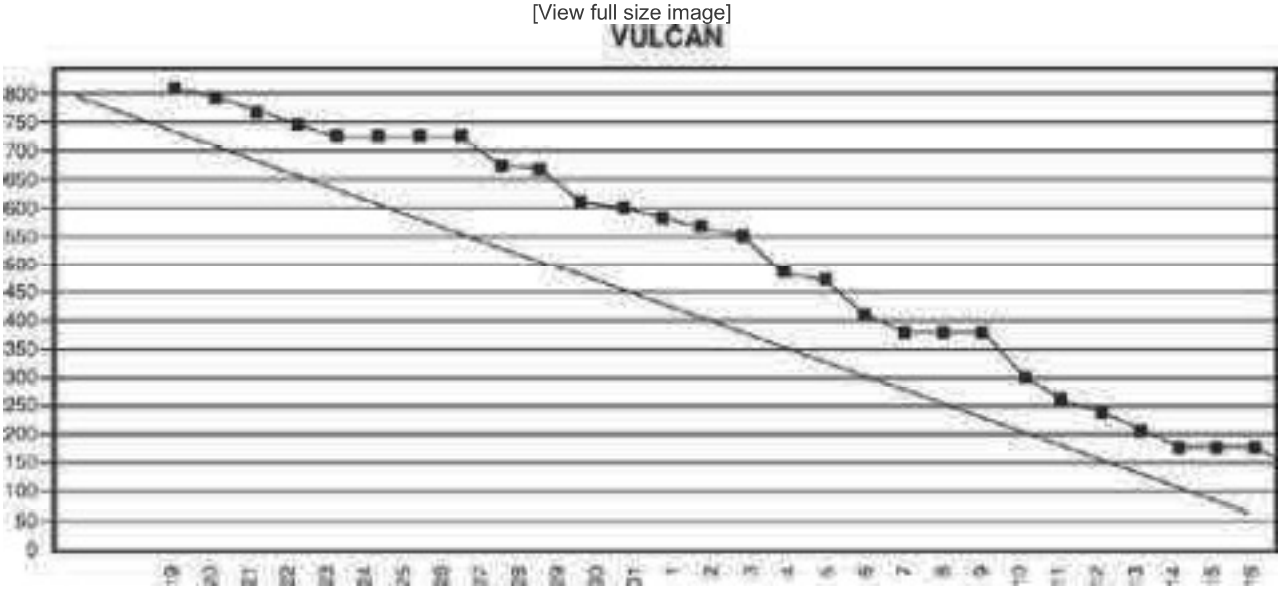
The Iteration Plan: Schedule Control at the Tactical Level

Although our release (or series of iterations) can be managed by watching the release burndown chart (refer to Figure 6-3) and facilitating the customer's decisions, the primary tools to measure iteration progress are the iteration backlog (or plan), iteration burndown chart, and the daily stand-up meeting.

The iteration burndown chart, shown in Figure 6-6 , is a graph that represents the team's remaining work in the iteration. Each day, team members update their tasks in the iteration backlog with the remaining hours for each task; these daily numbers are summarized and charted on the iteration burndown chart, which plots remaining work over the length of the iteration. As the team starts to work through their tasks, the idea is that the line will start to "burn down"; that is, the graph line will begin to follow a downward trend, representing a decreasing amount of work over the course of the iteration.

Figure 6-6. An example of a team's iteration burndown chart. The project manager and the team can tell early on that the team's velocity is not as high as originally thought. This will impact what is planned for the next iteration in the agile release plan.

[View full size image]



Progress of the iteration can be determined from this graph line. A flat line means that either the team is not updating its

remaining work hours, or there is an obstacle preventing progress. An upward spike means that new work was discovered. In Figure 6-6 , the team should have approached the product owner around the sixth day of the iteration, as it is clear that the team had overcommitted its work for the iteration. The agile project manager's goal is to remove obstacles that get in the way of completing the functionality that the team committed to in iteration planning. The burndown chart is a first indicator that the iteration is either on or off track, and if the team recognizes that it overcommitted to the iteration, it will discuss this with the customer to determine which low-priority feature should be dropped from the iteration; likewise, if the team undercommitted, the team and customer will figure out which feature to add to the iteration.

The second tool for gauging progress of the iteration is the 15-minute daily stand-up meeting. Team members report their status and impediments to their progress to each other; the agile project manager is on hand to listen for these impediments and take ownership if the team needs his help. External project members, such as stakeholders and managers, are encouraged to attend these daily meetings as observers, but only the team members are allowed to discuss their work. The team members outline the results of inspecting where they are in the iteration and making adaptations to their work plan based on real results.

By synchronizing in the daily stand-up meetings and working together in an ad-hoc fashion throughout the iteration, a team manages its tasks. Sometimes this means that a team member might change the order of his work or delete a task because it is no longer relevant. He will update his team members verbally in case their tasks are related to his, as well as update his tasks in the team's iteration backlog. It's a proactive, team-based approach to managing work within the timebox.

Once the iteration has concluded, the team will provide a high-level report to stakeholders at the iteration review meeting regarding which features it completed, as well as which features it did not complete. This is also a great time to apprise attendees of the status of the release based on the results of the iteration. After the team demonstrates the working features in the iteration review meeting, the team will hold an iteration retrospective to discuss details and actions to take regarding the planning processes in the next iteration planning meeting. Following are some examples of the types of questions that might be asked during an iteration review in order to focus on how the release might be impacted overall:

- What features did the team(s) complete this iteration?
- Was the iteration accomplishment more or less than what was expected?
- Is the team able to fully test the features? If not, what work remains, and how does that impact our release plan?
- What is the team's observed velocity?
- Is this velocity increasing over time? Decreasing?
- What are the factors bringing about the increase or decrease?
- How might this impact the other iterations in the release?
- Should we ask for more iterations? Do we need an iteration to integrate or run performance tests? Or should we remove the lowest-priority features from the release and put them back into the product backlog?
- How does the team feel about the plan based on observed results?

The answers to these questions may indicate a need for a re-baseline and communication of the updated release plan.

In Chapter 7 , "Cost Management," we discuss the use of Earned Value Management (EVM) techniques to control costs; there you will also find information about schedule variances should you choose to employ these techniques on your agile project.

The most important thing to remember about managing the release plan and the iteration plan is that "project status" is

real time, based on complete increments of functionality. There is a basic assumption that the features being delivered at the end of the iteration are of high quality; that no "technical debt" is carried over from one iteration to the next. Agile release planning management brings the stakeholders and team together in the iteration review meeting to communicate, in person, around the real status of the project: the capability to run tested features.

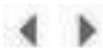
The idea that the team manages its tasks on a day-to-day basis can be frightening for new agile project managers because they feel a loss of control. Sometimes we meet project managers who feel that their team members cannot be trusted. Letting go and empowering a team can be a challenge, especially when a manager is not accustomed to working in that manner.

Many agile project managers have found that managing the project at a macro—or release—level, is a considerably more valuable use of time. In fact, many project managers we have spoken with say that by allowing the team to manage the day-to-day tasks, they have more time to work on contract negotiation and procurement, manage buy-versus-build decisions, remove obstacles for the team, work on agile reporting (in a program or project portfolio situation), as well as educate and help others understand and work within the iterative and incremental process.

Table 6-7 provides a summary comparison of the traditional and agile approaches to schedule control at the tactical level. In agile parlance, schedule control at the tactical level is simply referred to as "teamwork."

Table 6-7. Schedule Control at the Tactical Level

Traditional	Agile
Update the schedule based on approved changes, and re-baseline.	No changes are allowed in the iteration once the iteration has been planned, so that the team can work the features through to completion without distraction. However, the customer can decide to terminate the iteration if changes are such that they negate the value of the current iteration.
Calculate schedule variance and schedule performance index.	Remind the team of its duty to update the work remaining in the iteration backlog on a daily basis. The team also measures its velocity.
Track and document requested changes.	The customer updates the product and/or release backlog.
Identify and analyze recommended corrective action to get the project back on track.	Facilitate a discussion between the customer and the team to discuss any adaptive action that should be taken—that is, add another iteration, add a team, drop features, terminate the project, and so on.



Summary

This chapter makes the following points:

- An agile release is made up of a series of iterations. Releases can be internal or external, or just simply a period of time (usually quarters).
- We plan strategically for a release and tactically for each iteration.
- We estimate items in the product and/or release backlog at a high level, using "story points"; we estimate tasks in the iteration backlog at a detailed level, using hours.
- Product and release backlog items are referred to as "items," "requirements," "features," or "user stories." Any term is acceptable. However, do note that user stories have a defined format: "As a <type of user> I want to be able to do <feature/function>".
- The agile release plan progress is reviewed at the end of every iteration and adjusted based on conversations between the customer and the team.
- The agile release plan is used to help manage resources. However, agile assumes that the team is fully dedicated for the duration of the project.
- An agile release plan is communicated with the understanding that response to change is expected and built in. It is never meant to be prescriptive or locked down.
- Release and iteration burndown charts provide a first look into the status of the release and iteration, respectively.
- A team's average velocity and the burndown chart are two of the most important measurements of progress against the release backlog.
- Iterations can be thought of as padlocked steel boxes, where work should fill the boxes but not cause it to burst at the seams. This allows the team to work at a sustainable pace and establish a rhythm.
- Features that go into the padlocked boxes are negotiated by the team and the customer based on priority; we work on a "fixed-time, variable scope" model.
- Activity definition is done during iteration planning, where the team defines the tasks required to implement the features selected for the iteration.
- Agile processes do not track duration, because the duration is always the same—it's the length of the iteration. Only work effort is tracked, at the task level.
- Dependencies are discussed and addressed by the team in the iteration planning meeting; task dependencies are managed by the team and project dependencies are managed by the project manager.

Table 6-8 runs down the differences between a traditional and an agile project manager's time management behaviors.

Table 6-8. Agile Project Manager's Change List for Time Management

I used to do this:	Now I do this:
Determine the requirements for the project before developing the project schedule.	Facilitate discussions with the customer and team in order to define the features in detail that are appropriate to the horizon. Help the customer create a product backlog that is easy to maintain.
Work with a resource manager to determine when I would need what types of skill sets, based on the detailed project schedule.	Enjoy the consistency of a dedicated team where resources no longer come and go; continue to work with resource managers on the remaining exceptions based on the needs outlined in the release plan.
Ask for estimates of the tasks required to complete the requirements.	Ensure that the team is providing high-level estimates for the items in the product and/or release backlog and then more granular estimates for tasks as they are defined in the iteration planning meeting.
Identify all task dependencies and adjust the schedule to show the relationships and how best to manage them.	Trust in the team to manage its task dependencies; focus on the broader project dependencies and work with others to ensure proper coordination.
Create the project schedule.	Facilitate release and iteration planning meetings, where the team creates the high-level strategic release plans and the more detailed tactical iteration plans.
Update the project schedule; work diligently with change control to prevent scope creep.	Help the team in updating the burndown charts that indicate iteration and release progress; remind the customer of the need for adjustments in the release and product backlog and product roadmap based on team performance and deliverables accepted each iteration.

Endnotes

1. Ogunnaike. *Process Dynamics, Modeling, and Control*
2. Austin. *Artful Making*.
3. Coined by Peter Drucker in 1959, a knowledge worker is "one who works primarily with information or one who develops and uses knowledge in the workplace." http://en.wikipedia.org/wiki/knowledge_worker.
4. *PMBOK® Guide*, 143.
5. Don't get confused by the naming conventions for items in the backlog! A feature, user story, and requirement are often the same thing—what distinguishes a user story is the format in which it's written. In this book, we'll refer to requirements, stories, and features and use them interchangeably. For more information on user stories and story points, read Mike Cohn's *User Stories Applied and Agile Estimating and Planning*.
6. Originated by Mike Cohn. For more information, see his book, *User Stories Applied*. (Boston: Addison-Wesley, 2004).
7. *PMBOK® Guide*, 127.
8. Ibid., 128.
9. Ibid., 130.
10. Ibid.