

Chapter 15

The Agile Release Train

Today's development processes typically deliver information asynchronously in large batches. Flow based processes deliver information in a regular cadence in small batches. Cadence lowers transaction costs and makes small batches more economically feasible.

– Don Reinertsen

Chapter 15 Table of Contents

Introduction to the Agile Release Train.....	15-2
Rationale for the Agile Release Train.....	15-3
Principles of the Agile Release Train	15-5
Driving Strategic Alignment.....	15-6
Institutionalizing Product Development Flow	15-7
Designing the Agile Release Train	15-9
Planning the Release.....	15-10
Release Objectives	15-10
Tracking and Managing The Release	15-11
Release Retrospective	15-12
Measuring Release Predictability	15-12
Release Objectives Process Control Band	15-13
Releasing.....	15-14
Releasing on the ART Cadence	15-15
Releasing Less Frequently than the ART Cadence.....	15-15
Releasing More Frequently than the ART Cadence	15-18
Summary.....	15-18

The original title for this Chapter was to be *The Release*, or perhaps *Releasing* or *Release Planning and Execution*. But none of these titles, nor others that I toyed with, communicated the essence of what I intended to communicate. Each of them implied a thing that was historically true, that the *release event*, or the *release planning* event, or both, were a “really big deal” in the enterprise. It represented either the *beginning* (release planning) or the *end* (the release) of some significant project – a major milestone in the history of the company. But that didn’t resonate for me and it harkened back to the psychology of the waterfall, whereby achieving a release was some giant milestone – a cause for celebration – write the press releases – schedule an all hands event – get your CEO on stage - or whatever.

None of these notions reflect life in the agile enterprise. Instead we see a continuous *flow* of releasing value to the users in small, frequent increments - a continuous build of value added to the marketplace. Releasing often; yet typically with little fanfare. Properly done, it’s a big win in the market, but it’s hard to look backwards and determine exactly when the successful tipping point was reached.

That’s not to say that releasing product to the market with traditional fanfare is no longer apropos, and we’ll describe how enterprises go about making newsworthy events out of steady, incremental progress later. But this Chapter will focus primarily on how to *make each product release a successful and routine event* -an event which is indeed planned and eagerly anticipated - and yet one which happens almost on autopilot. We call this process the *Agile Release Train*.

Introduction to the Agile Release Train

In Part I of this book, we introduced the Program level of the Big Picture of with an overview graphic and an explanation. A portion of the graphic is provided in Figure 15-1 for context.

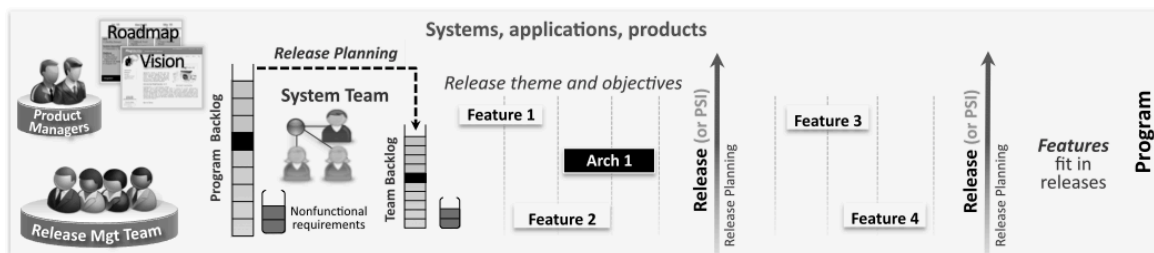


Figure 15-1 Program level of the Big Picture implying the Agile Release Train

In the figure, we see that at the program level, teams and activities are organized around an ongoing series of incremental releases. The releases may be *internal*, and used for evaluation of the system as a whole (in which case we call them *Potentially Shippable*

Increments). The releases may be made *external*, in that it is made generally available to our customers (in which case the *Release* label is more appropriate).

In any case, development of the software asset base occurs with a standard cadence of iterations that has been established by the enterprise. In the Big Picture, we've illustrated four *development* iterations (indicated by a full iteration backlog), followed by one *hardening* iteration (indicated by an empty backlog) prior to each release increment.

This pattern is arbitrary and there is no fixed rule for how many times a team iterates prior to a PSI, or how much, if any time or investment in hardening is required. However, many enterprises apply this model with a repeating pattern of 4-5 development iterations, followed by a hardening iteration, creating a cadence of a shippable increment about every 90 days. This is a fairly natural production rhythm that corresponds to a reasonable external release frequency for customers and also provides a nice quarterly planning cadence for the enterprise itself.

In any case, the length and number of iterations per release increment, and the decision as to when to actually release a PSI, is left to the judgment of each enterprise. However, the planning of an *external* release requires special care and attention, as we'll cover later.

Rationale for the Agile Release Train

While we introduced the Agile Release Train early in Part I of the book, we didn't take the time to justify its existence or use. We simply posited it as an answer to some set of problems or unasked questions. However, as implementing the ART is no small feat for the enterprise, and as its use can be quite instrumental in achieving enterprise success, it seems reasonable to introduce the rationale before we go any further.

Like much of the guidance in this book, many of us did not start out with the ART when we headed down the agile path. More typically, we rolled out scrum/XP/or hybrid variant and focused on getting the teams successfully building small increments of working, tested functionality in a short time box. Moreover, if we were only working with one or a few teams, releasing the product did not create many additional challenges - we could release it when we felt like it and whenever the market required it. Coordinating our efforts wasn't extraordinarily complicated either - we could do that by just talking amongst ourselves in the hallways, or maybe we'd have to reserve a conference room to meet with other stakeholders, - quality, sales marketing, etc. would all be present.

This was appropriate and was the *simplest thing that could possibly work*.

However, as the number of teams engaged in agile development in the company increased – or as the enterprise grew with its successes - or as we acquired new teams and new products - or as our customers drove us to higher levels of integration amongst the various components of our solution, a *substantial* problem began to emerge. The problem became:

How do we harness all that new, empowered, but potentially entropic, energy into a cohesive team of teams that can deliver ever larger and more integrated piles of value to our customers?

For many, the first temptation was to re-implement some of the former planning and governance models we had abandoned when we “went agile”. “That should get things on track”, we thought. The results of that approach are pretty obvious – the teams fought back hard against the new overhead and governance, and lack of empowerment that it implied. Worse, those who attempted to implement, or re-implement, these governance practices became impediments to be avoided (“careful, don’t make eye contact with the program manager.....”). So simply stated, that didn’t work.

That left many companies in an in-between state – improving performance of agile teams, but still very hard to build an integrated solution. Whether we understood it or not, the “process model” we had implemented for larger-scale, system releases was starting to look something like Figure 15-2¹.

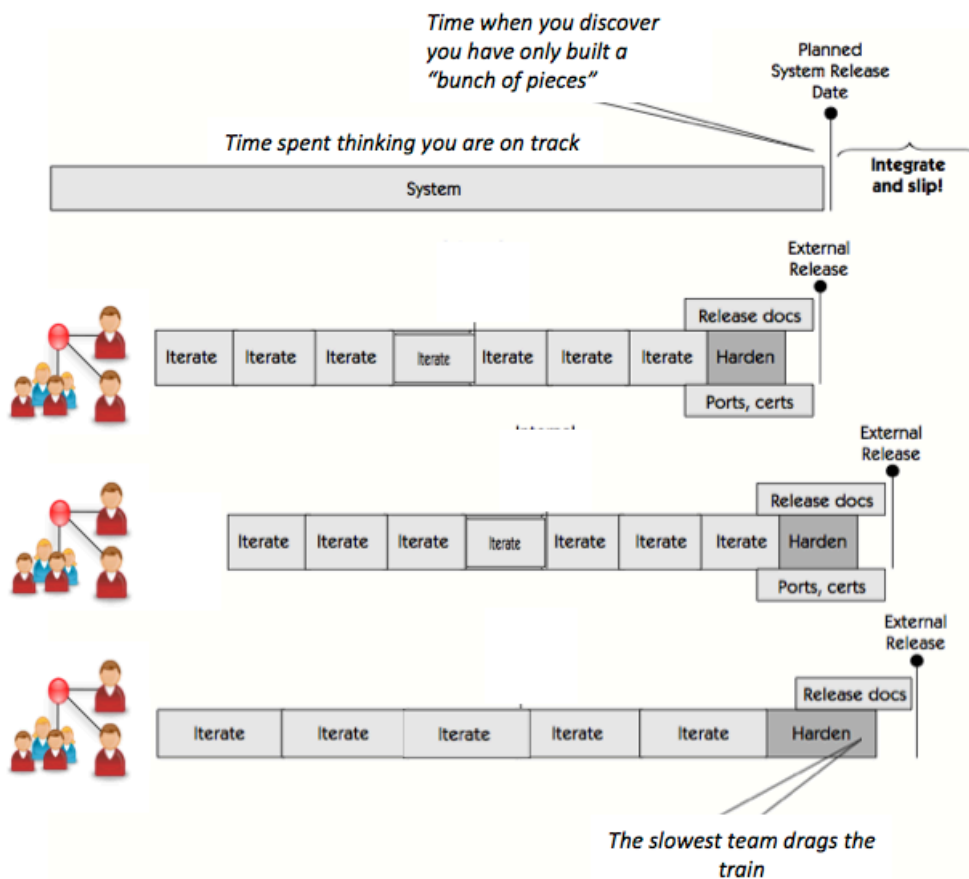


Figure 15-2 Agile development, before the release train

Teams were iterating with some degree of success; indeed some were now maniacally focused and committed to their specific product, feature or component. That seems like real progress.

¹ Adapted from [Leffingwell, 2007] Scaling Software Agility: Best Practices for Large Enterprises

But their iterations lengths were different - continuous integration at the system level was unachievable - the vision for the larger system was hard to decipher or even infer. As a result, as newly agile as we believed ourselves to be, our releases and release quality still suffered from some of the same deferred risk practices as the waterfall model that we had just so recently abandoned!

To address this problem, the Agile Release Train evolved.

Principles of the Agile Release Train

The Agile Release Train provides alignment and helps manage risk by providing program level cadence and synchronization. It is based on agreement and adoption of a set of common operating principles (ok, rules) which are followed by *all* teams who will be placing cargo (user value) on their particular train. These rules include:

- Frequent, periodic planning and release (or PSI) dates for the solution are fixed. (dates are fixed – quality is fixed – scope is variable)
- Teams apply common iteration lengths.
- Intermediate, global, objective milestones are established.
- Continuous system integration is implemented at the top, system level, as well as at the feature and component level.
- Release increments (PSIs) are available at regular (60-120 day typical) intervals for customer preview, internal review, and system-level QA.
- System level *hardening* iterations are used to reduce technical debt and to provide time for specialty release level validation and testing.
- In order for teams to build on like constructs, certain infrastructure components - common interfaces, system development kits, common installs, user stores, licensing utilities and the like - must typically track ahead.

Constraining teams to the dates and fixed quality criteria means that the system, feature and components functionality must be flexible (or we will quickly find ourselves back inside the iron triangle).

While these rules may not seem that constraining, the fact is that this model requires an additional degree of agility and flexibility on all teams and stakeholders who participate:

From the perspective of the team - In order to be assured of meeting a date, a team might need a primary plan and a fallback plan (or perhaps a set of options) they can deploy as necessary to make sure they can “get their cargo on the train”. In some cases, the fallback plan can be as simple as planning to ship the old version. Even then the team must support any new interfaces, provide backwards compatibility, and be certain to not violate any other common requirements (compliance, localizations, etc) that may be imposed on the cargo.

From the perspective of product, program and executive management - The plan is a result of a collaboration, which weighs the input of all stakeholders and also matches input (release requirements) to capacity (development team velocities) so

that flow can be achieved. Rarely, if ever, do expectations of input and output match. Compromise is required. Moreover, the result of the plan is just that, a *plan*, and the exact scope of the final achievement can still not be known for certain, up-front.

While implementing the Agile Release Train is a far from trivial task, implementation is a must for the agile enterprise. It addresses two imperatives that are necessary to achieve success within the enterprise. They are:

- 1) Driving strategic alignment
- 2) Institutionalizing product development flow

We'll look at each of these below.

Driving Strategic Alignment

Empowering individual agile teams to truly focus on rapid value delivery typically unlocks the raw energy, motivation and innovation that has likely been stilted by our pre-agile process and governance models. That's why we do it. However, that alone is not enough, as the teams will naturally tend towards *local optimization*. They'll do what they can to deliver requirements to their customer constituency, but they that have less interest (or perhaps awareness and ability) in taking a more global view². After all, having two masters is more complicated.

However, in the lean enterprise, the highest benefit is achieved when we achieve *global optimization*³. In order to do this, we must implement systems, like the ART, that purposefully drive the teams towards the global targets, as illustrated in Figure 15-3.

2 One VP, who had responsibility for a dozen such agile teams, called this his “twelve tribes of Israel” problem - all empowered, all agile, yet each wandering independently towards their own version of the promised land.

3 Product development flow principle 8.7 *There is more value created with overall alignment than with local excellence.*

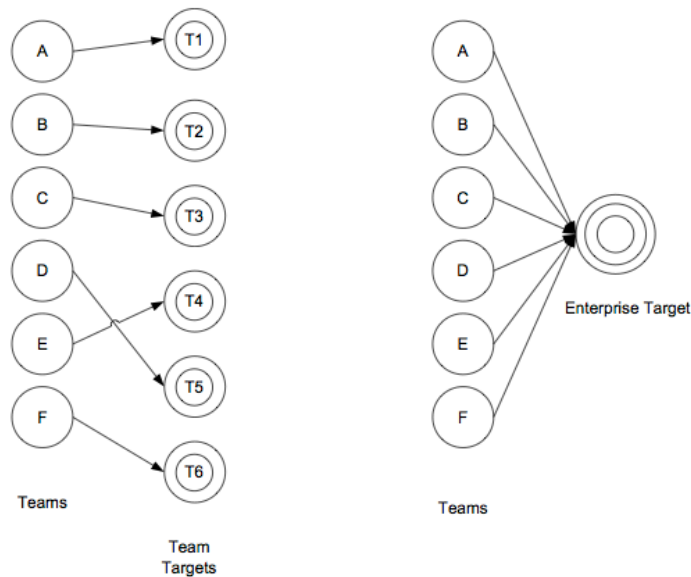


Figure 15-3 Aligning agile teams to a common target

In this way we can align our *mass* to a common direction and achieve far more *force* to address the targets of opportunity. We can have both local *and* global alignment to a common goal.

Institutionalizing Product Development Flow

In addition to driving alignment, The Agile Release Train is instrumental in institutionalizing *product development flow*. In so doing, the ART supports the eight primary product development flow themes that we described in Chapter 1. Understanding this mapping is the key to understanding the criticality and motivation for the ART itself:

Theme 1 - Take an economic view.

Incremental releases substantially improve the ROI of software development by accelerating the release of value to the customer. This helps capture early market share and drives gross margins by delivering features to the market at the time when the market values them most highly.

Theme 2. Actively manage queues.

The short, frequent planning cycles of the Agile Release Train help actively manage queue lengths across the enterprise.

Team backlogs (queues of waiting stories) are generally limited to about the amount of work that can be accomplished in a single PSI. Planning much beyond that is generally not very productive for the teams, because strategic priorities could change at any release boundary.

Release (Program) backlogs (queues of waiting features) are typically limited to those features that can realistically be implemented in the next release or two.

Beyond that, product managers understand that they may be overinvesting in elaboration of features that will never see the light of day.

Portfolio backlogs (queues of waiting epics and future projects) are typically limited to those epics that could likely find their way to release planning in the next six months or so. Too early, or too in-depth, investment in business cases for projects that will not be implemented is a form of waste.

Theme 3. Understand and exploit variability.

Since a high degree of variability is inherent in software development, frequent re-planning provides the opportunity to adjust and adapt to circumstances as fact patterns change. New, unanticipated opportunities can be exploited by quickly adapting plans. Critical paths and bottlenecks become clear. Resources can be adjusted to optimize throughout and better avoid unanticipated delays.

Theme 4. Reduce batch sizes.

Large batch sizes create unnecessary variability and cause delays in delivery and quality. ART reduces batch sizes by releasing only those features to the development teams that are prioritized, elaborated sufficiently for development, and are sized to fit within the next release cycle. This avoids overloading the development teams with multiple development projects, which otherwise causes multiplexing, thrashing and loss of productivity. Face to face planning provides high bandwidth communication and instant feedback, so the transport (handoff) batch delay between teams is minimized.

Theme 5. Apply WIP constraints.

Teams plan their own work and take on only the amount of features that their velocity indicates they can achieve. This forces the input rate (agreed-to, negotiated release objectives) to match capacity (what the teams can do in the release). The current release timebox prevents uncontrolled expansion of work so that the current release does not become a “feature magnet” for new ideas. The global WIP pool, consisting of features and epics in the enterprise backlog, is constrained by the local WIP pools, which reflects the team’s current backlog, as driven by the current PSI.

Theme 6. Control flow under uncertainty - cadence and synchronization.

Cadence and synchronization help us manage uncertainty and variability, by keeping accumulated variances to single interval. In the ART, we achieve this through periodic *planning* (cadence) and *integrating* (synchronization).

Planning - The release train planning *cadence* makes planning predictable, and lowers transaction costs (facilities, overhead, travel.) Planning can be scheduled well in advance, allowing participation by all key stakeholders in most planning events, making face-to-face information transport reliable, efficient and predictable. Periodic re-planning (*resynchronization*) allows us to limit variance and misalignment to a single planning interval.

Integrating – The *synchronization* of regular, system-wide integration provides high fidelity system tests and objective assessment of project status at regular intervals. Transaction costs are lowered as teams are forced to invest in the infrastructure necessary for continuous integration, automated testing, and more automated deployment. Since planning is bottom up, (performed by the teams and based on teams actual known velocity) and short term, delivery becomes

predictable. Most all that has been planned should be reliably available as scheduled.

Theme 7. Get feedback as fast as possible.

The fast feedback of the iteration and release cycle allows us to take fast corrective action. Even within the course of a PSI, feedback is no more than two weeks (or the iteration length) away. Small incremental releases to customers allow us to track more quickly to their actual, rather than anticipated, needs. Incorrect paths can be abandoned more quickly (at worse, at the next planning cycle).

Theme 8. Decentralize control.

Release plans are prepared by the teams who are doing the actual implementation, rather than by a planning office or project management function. Commitments to the plans are bottom-up, based on individual's commitment to their teammates and team-to-team commitments reached during the planning cycle. Once planned, the teams are responsible for execution, albeit subject to appropriate lightweight governance and release management. Agile project management tooling automates routine reporting; management does not have to slow down and annoy the teams to assess actual status.

So as we see, the ART is fundamental in achieving strategic alignment and program-level, product development flow. In the next sections, we'll describe how to implement and manage the Agile Release Train.

Designing the Agile Release Train

One initial activity is to determine the *release train domain* – i.e. who will be planning and working together, and what products, services, features, or components the train will deliver. In the big picture, we've indicated that there is some collection (or pod) of agile teams that constitute a program. That is often the case, and in the smaller enterprise or business unit, the ART domain consists of everyone on the team who will participate in the outcome. So if the assets you are building can be built with 5-8 agile teams, then the planning domain *is* the program and not much more thought is required.

However, in the larger enterprise there may be dozens (or more) of such teams and planning everything together is not feasible. In that case, we must first determine who will be on the train. Considerations should include:

- Trains should be focused on a single, primary product or solution objective.
- Trains work best when between 50-100 people, including stakeholders outside the team, contribute to the train
- Teams with features and components that have a high degree of interdependencies should plan and work together
- Locale is a major consideration – wherever possible, train teams should be collocated, or at least geographic distribution should be as limited as feasible, as that simplifies planning logistics and cooperation amongst the teams

Planning the Release

Once the parameters and the cadence for the ART have been established, the teams can establish a release-planning schedule for the train. Since the dates for the PSIs are fixed, the release planning dates can be fixed as much as a year in advance. This helps lower facility, travel, overhead and other transactions costs associated with the event.

In many contexts, a somewhat standard, two-day release planning agenda can be appropriate, as seen in Figure 15-4.

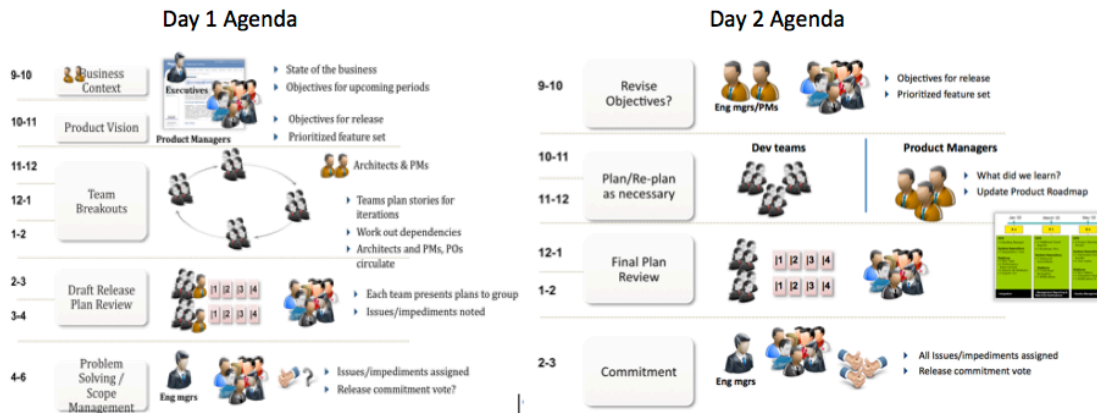


Figure 15-4 Example two-day Release Planning Agenda (See Chapter 15)

Given the importance of the event in driving strategic alignment, planning and executing the release event is a small project unto itself. We'll cover Release Planning thoroughly in the next Chapter.

Release Objectives

As we will see in the next chapter, one important result of the release planning process is a set of *release objectives*, which define the individual team and aggregate goals of the release. These quantitative objectives are a key artifact of the release planning session and provide us with an important baseline for release governance and tracking. Each objective will have been ranked by business value, as the example in Figure 15-4 illustrates.

Most of the objectives will be features from the backlog, and this gives us the targets we need to track and manage the release.

<u>Objective</u>	<u>Bus Value</u>
1. Thermostat Over-the-Air Update	10
2. Next generation thermostat firmware	4
2. First pricing programs	10
3. Gateway Pointing Re- architecture	6
4. Trade show demo by 3/15	10
5. Release v3.1 upgrade to channel	9

Figure 15-5 An example of release objectives, ranked by business value

Tracking and Managing The Release

With the quality and date fixed, it is certain that some amount of adjustment to content will be needed during the course of the PSI. In support of this need, the enterprise will likely have implemented some form of agile requirements/project management tooling, which provides support for the higher-level status views needed by product managers and other stakeholders. Such tooling should provide hierarchical, feature-level burn down so that the program can assess, *on an aggregate basis*, exactly where they stand within the release as Figure 15-6 shows:

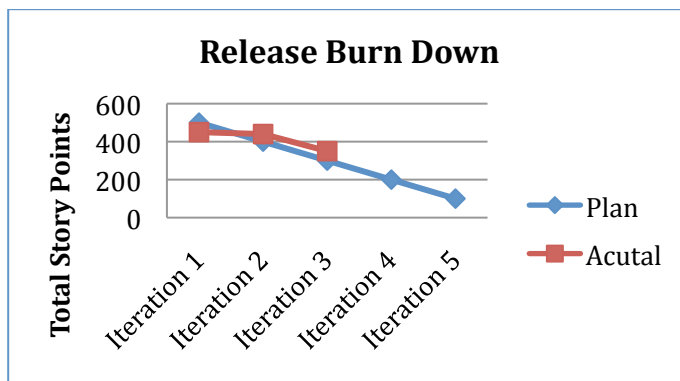


Figure 15-6 Release level burn down

This chart provides a sense of the probability of “landing” (delivering the expected value) of the release. However, by itself, it doesn’t provide any information as to *which* features

may or may not be delivered. For that, the tooling should also provide default reports on the status of each individual feature, which is in turn, based on percentage of story completion for that feature. Such a chart might look something like Figure 15-7.

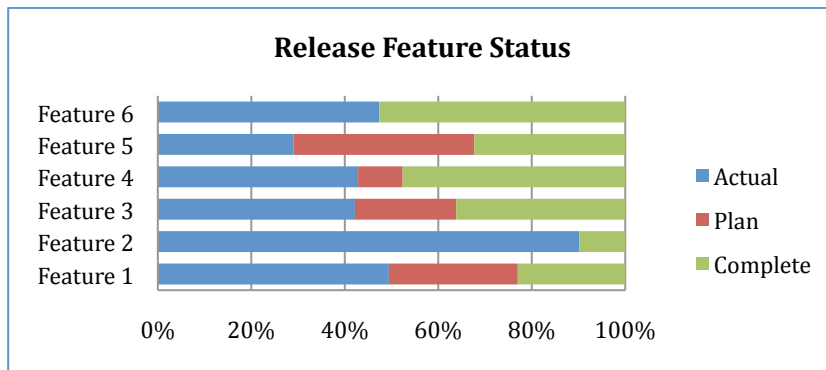


Figure 15-7 Release status by feature

Together, this information provides the objective knowledge of where it is, and even more importantly, what changes might be necessary to successfully deliver the release. After all, content (scope and feature) management is continuous in agile, and an in-flight ART is no exception.

Release Retrospective

Each PSI boundary also provides the opportunity for a program-level release retrospective, wherein the teams assess how well they did and take corrective action to increase the velocity, quality and reliability of the next release increment. This retrospective can also include a quantitative measure of individual and team *predictability*, as we'll describe below.

Measuring Release Predictability

If you ask top executives what they would most like to see out of the software development process, many will answer “*predictability*”. And that is one of the many challenges in agile. We can reliably predict *quality* (by fixing it and adopting effective technical practices) and *date* (by fixing it) and *cost* (by fixing the team size and the PSI date) – but we can't actually predict *functionality* – at least in the longer term. If we did, we'd be right back in the iron triangle that has served us so poorly in the past. Moreover, if we predicted and controlled functionality long term, then we'd have to temporarily ignore the variability and new opportunities the market presents. That isn't agile.

However, as professionals, we must be able to provide our enterprise with a reasonably reliable predictor of up coming events, at least near term, as well as some sense of the future product roadmap that we intend to execute.

When implemented properly, the ART can provide just such a predictability measure, at least for the next PSI (or maybe two). That gives the enterprise from 3-6 months of visibility into upcoming release content - enough to plan, strategize and support with market communications, release launches, etc. The *release objectives* that we established during release planning are our primary means to do this.

During each the release retrospective, the teams can meet with their business owners to self-assess the percentage of business values they achieved for each objective. This can be done both at the team and program level. For example, a program might rate its accomplishments as in Figure 15-8.

<u>Objective</u>	<u>Bus Value</u>	
	<u>(plan)</u>	<u>actual</u>
1. Thermostat Over-the-Air Update	10	8
2. Next generation thermostat firmware	4	0
2. First pricing programs	10	8
3. Gateway Pointing Re- architecture	6	4
4. Trade show demo by 3/15	10	10
5. Release v3.1 upgrade to channel	9	9
	====	====
Totals	49	39
% achievement:	79%	

Figure 15-8 Plan vs. actual release accomplishments

Release Objectives Process Control Band

In the example above, the program accomplished 79% of its release objectives. The questions arises - how good, or bad, is that? To answer this we must return to our lean principles and the context for ether enterprise program itself.

On the surface, at least, it might appear that accomplishing 100% of release objectives is the only worth goal. Closer analysis, however, tells us differently. For in order for a team to routinely accomplished 100% of its release objectives, they must either

- 1) drive all risk out of the plan by eliminating or curtailing innovation and risk taking
- 2) back off so far on objectives so as to assure completion

Neither of these optimizes the economic impact of our efforts. To achieve that, we need to operate successfully in some acceptable *process control band*, so that the program has

reasonable predictability, and yet allows for the variability, “optionality”, and stretch goals inherent with software development.

In our experience, a program that can reliably achieve *most* of its release objectives is a trusted program that is an extraordinary asset to the enterprise. In this case, the release predictability measure should fall in a process control band something like that in Figure 15-9 over time.

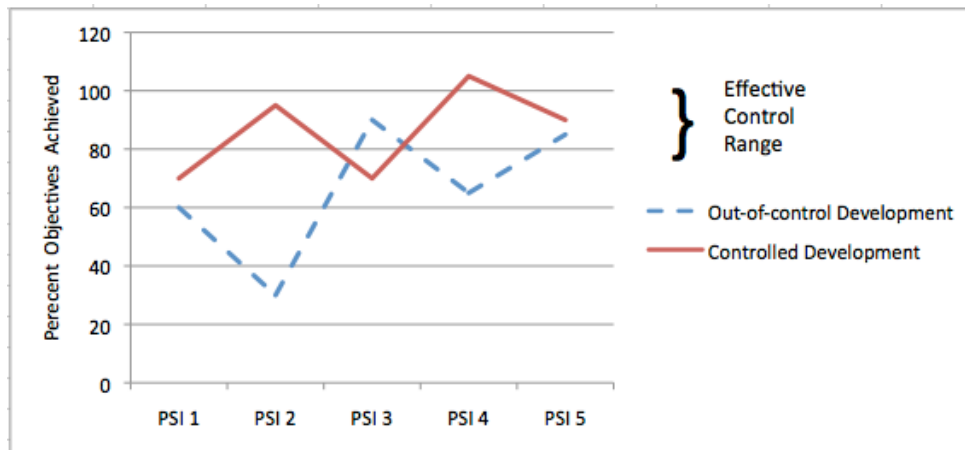


Figure 15-9 Release predictability process control band

In this figure, team A does not routinely hit 100% of its release objective; leaving room for innovation and risk taking. However, it is highly predictable. Team B, however, is all over the map. It’s hard to manage any program or enterprise with the characteristics of Team B.

By creating and measuring this predictability measure at every PSI, the enterprise can eventually achieve the right balance of predictability and risk taking, thereby achieving the optimum economic outcomes.

Releasing

That is all well and good, but there is still work ahead. The Agile Release Train simplifies software development by “making routine, that which can be routine”. Planning and team coordination is simplified. Work in process is limited. Flow is achieved. In describing it so far, however, we have oversimplified one of the more complicated challenges - which is an understanding of when to actually *release* a set of assets to the customers, distribution or marketplace. In order to coordinate the ART with actual releases of products, we must consider three separate cases, *releasing on the same cadence as the ART*, *releasing less frequently*, and *releasing more frequently*. We’ll look at each of these in the sections below.

Releasing on the ART Cadence

When you look simplistically at the big picture, it tends to imply that the planning and release cadences are identical as Figure 15-10 illustrates.

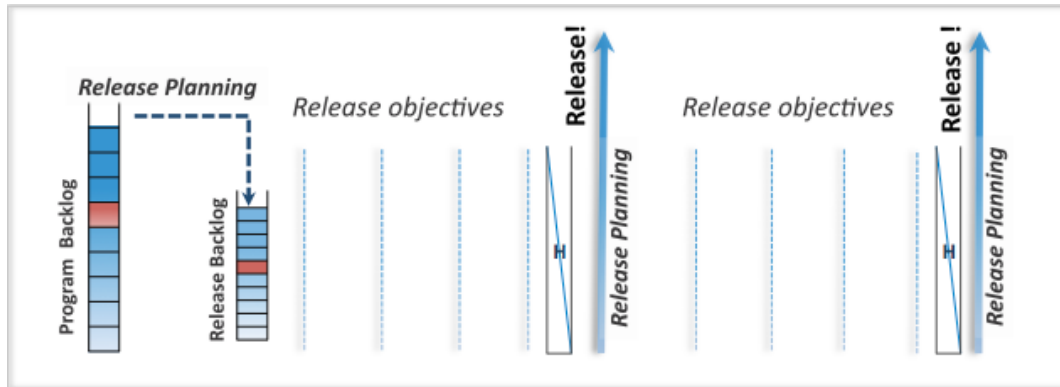


Figure 15-10 Releasing on the ART cadence

This is the simplest case, as planning, releasing, and release retrospectives are coordinated by the same cadence and same calendar dates. In addition, the hardening iterations are timed nicely to support the more extensive release activities, which can include everything from preparation of release notes and customer documentation, standards validation, load and performance testing, built in demos and tutorials, updating user documentation and the like.

This is the model used by a number of SaaS (Software as a Service) companies. In that case, the release train is conceptually simple as they release frequently and have only a single platform to support. All customers are migrated to the new release at the same time. There is one planning session and only one release per PSI.

This is incredibly convenient to the program as *all* activities - internal supporting functions as well as and customer-facing activities - can be driven by the same cadence and synchronization as the development of the assets themselves. Program planning is harmonized. The model is conceptually simple. Life is good, and lean.

However, this is not always a practical case, so other models come into play.

Releasing Less Frequently than the ART Cadence

In many cases, releasing on a fast PSI cadence may not be possible. For example, in some enterprise settings, deployed systems constitute critical infrastructure of a customers operating environment. Even if the customers would like to have the new software, service level and license agreements may be prohibitive and there is the overhead and disruption of installation. Plus there is always the fear of regressive bugs affecting customer operations, potentially on a large scale. Scary stuff.

In other cases, the timelines of enterprises building systems that contain both software *and* hardware, such as mobile phones and other devices, are often driven by long-lead

hardware items - displays, chip sets, keyboards, cases and the like. Here, there are actual laws of physics involved that must be obeyed. You have to have the new hardware first, so releasing early and incrementally is not an option.

In these cases, releasing on a PSI cadence is simply not an option and the planning and releasing activities must be decoupled.

There are other reasons to decouple the PSI cadence from the release cadence as well. For example, while the sales and marketing and development teams are headed to the same end goal—that is, more and higher quality product released to the market more quickly—their intermediate objectives may be quite different.

For development, the goal is to deliver more value to the market more quickly on a cadence that provides the highest productivity and fastest market feedback. Indeed, you often see development teams pushing for ever-shorter release cycles to put pressure on themselves to fix the major impediments in their own internal processes. Best practices such as concurrent testing and continuous integration are best implemented and mastered under these intense pressures. Also, the more routine the operations of daily build, release retrospectives, iteration acceptance testing, and the like become, the easier it is to manage their daily and weekly activities and to thereby avoid the “death march” experiences at the end of each release cycle. So development likes to push for very fast release cycles.

For sales and marketing, the goal is also to deliver more software as rapidly as possible, but they also strive to optimize the *market impact* of their efforts. This means release frequency may be limited by practicalities such as:

- not disrupting customers with too frequent upgrades.
- not complicating any big deals in process that are based on the current GA product.
- having worthy (bigger) news to take to the market, typically in synchronization with analyst tours, trade shows, and so on.
- avoiding overloading internal support infrastructures for marketing communication, sales and support training, and so on.

Separation of Development from Release and Marketing Concerns

So while the goals are the same, the perspectives and concerns of these two departments may be quite different. Therefore, we need to provide a model that allows *for separation of those concerns* to achieve a higher degree of flexibility and greater market impact.

Figure 15-11 illustrates such a model:

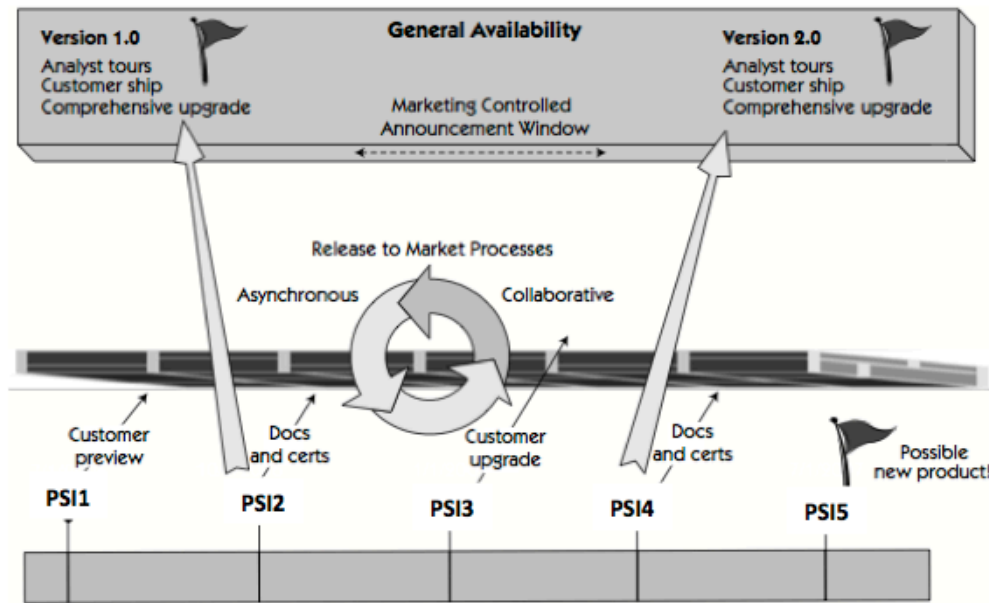


Figure 15-11 Separation of development, release and marketing concerns

This model decouples asset development from product release via a “general availability firewall”. The firewall allows “just the right number of releases” to reach the market at “just the right time.”

The process is flexible, asynchronous, and collaborative among the development, operations, and sales and marketing teams. For example, in the figure above:

- PSI1 is an internal release, suitable for customer previews.
- PSI2 is the released to the market as Version 1.0.
- PSI3 is a quiet release, perhaps just deployed to customers under maintenance, but it may not be newsworthy as it follows Version 1.0 by only 60 or 90 days.
- PSI4 is positioned to the market as a major new release because it incorporates two releases of new functionality.
- PSI5 may even be positioned as a new product, rather than a continuum of the existing product line.

Of course, this release schedule is just an example that shows the flexibility of the model, and your train will address different objectives. There is no need to predict in advance exactly how an agile release train will evolve – just build *it* and figure out what to do with *it* later.

The Firewall can be Opaque or Transparent

The firewall can be as opaque or transparent as the enterprise needs. In other words, development of new functionality can be kept confidential, or customers can be told about the availability of new releases, or even the product roadmap. The development team is free to establish the best production rhythm it can master, continuously building incremental product functionality. Marketing is free to deliver external releases on an ad

hoc basis - responding contemporaneously to current market conditions, competitive responses, etc. - or on a planned, programmatic basis – whatever meets their needs.

Releasing More Frequently than the ART Cadence

For programs and enterprises who are building systems of systems, either of the two cases above can still appear to be overly simplistic. In these cases, while the larger system may lend itself to either of these models above, various components of the system may have to serve different masters.

For example, in a securities trading system, back office transaction servers may need to be updated for ongoing securities compliance on their own independent schedule. Client side software may be gated by availability of new types of securities to be traded, or access to new securities marketplaces.

In our case study example, the Tendril System is composed of a number of different components. For various business reasons, these components may need to be released to the market at various times and for various reasons as the figure below shows.

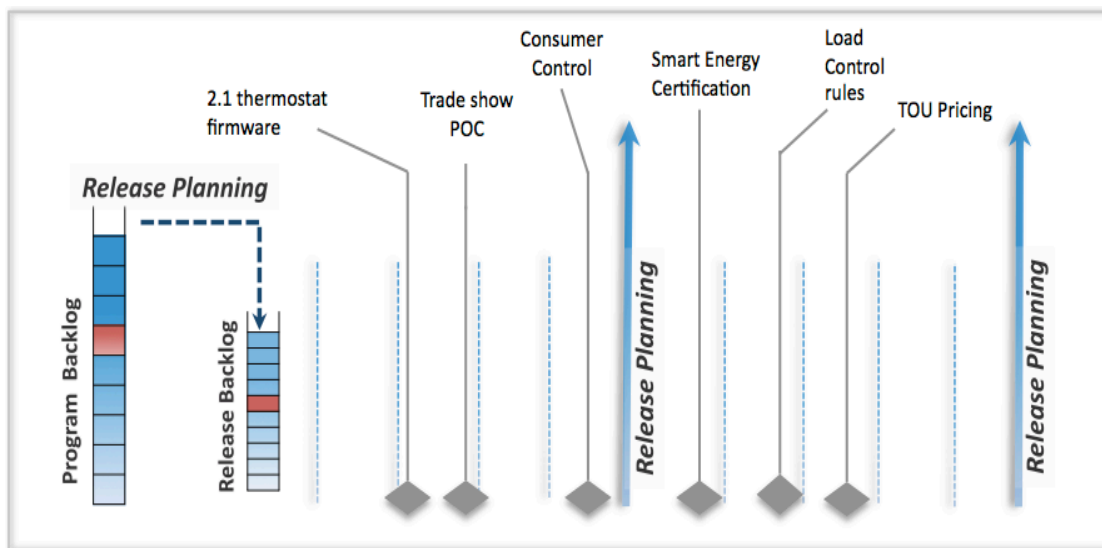


Figure 15-12 Releasing more frequently than ART cadence

In this third case, releasing more frequently, the PSI cadence becomes a *planning*, rather than *release* cadence. The periodic planning function still provides the cadence, synchronization and alignment the enterprise needs to manage variability, but forcing the development of all assets to the same cadence is unnecessary, and over-constrains the solution.

Summary

This chapter introduced the Agile Release Train as a mechanism to drive strategic alignment and institutionalize product development flow. We described the rational

behind the model, along with the mechanics for implementing a release train. We also used the ART to introduce a predictability measure the enterprise can use to help predict near term deliverables. Finally, we showed how the ART is highly flexible, and how it can be used to provide enterprise alignment, synchronization and cadence, even if release requirements do not align perfectly with the ART cadence itself. We hope that this introduction will provide the motivation and background you need to implement a release train in your program so that you too can achieve the steady drumbeat of value delivery that characterizes truly agile programs.

In the next Chapter, we'll describe a thorough approach to Release Planning, which you can use to put your own train firmly on the tracks.