The Product Manager and the Product Development Process

Martin Cagan

Silicon Valley Product Group

Silicon Valley Product Group

THE PRODUCT MANAGER AND THE PRODUCT DEVELOPMENT PROCESS

Martin Cagan, Silicon Valley Product Group

OVERVIEW

The product manager's job must be done in the context of an overall product development process. Typically, this product development process is driven more by the product development organization rather than product management, so in these cases especially it is important that the product manager has an understanding of the strengths and weaknesses of the product development process, and can ensure that she does what is necessary for a successful product.

There are literally hundreds of distinct documented software development processes, covering the full spectrum from very strict and formal, to very light and informal. However, organizations change their development processes infrequently and slowly, so in all likelihood, you will only encounter a few of them.

In this paper we discuss the most common product development process, along with two of the more popular alternatives, each from very different schools of thought. Our focus here is not to explain in great detail the different development models, but rather to highlight the issues in each that pose concerns for the product manager.

CONVENTIONAL/WATERFALL DEVELOPMENT PROCESS

Even though the Waterfall development process is more than 30 years old, and even though it is often cursed by engineers and product managers alike, it is still by far the most common process used to create software products.

Note that it has long been unfashionable for a team to describe their product development process as "waterfall" yet in most cases that is essentially what is still being followed, albeit by many different names including *Successive Refinement*, *SDLC*, *Stage-Gate*, *Phase Review*, *Staged Contracts*, *BLIP* and *Milestone-based.*

In this paper we'll try to capture the strengths as well as the key weaknesses of this approach, and discuss what the product manager can and should do in order to maximize the chance of success with this process.

General Principles

The conventional waterfall process is extremely simple in concept. Software progresses through a well-defined series of phases beginning with a written description of the requirements, then into first high-level architectural design, and then to detailed design, then to code, then testing, and finally deployment. At the end of each phase there is a review of the deliverables from that phase, followed by sign-off and an explicit transition to the next phase.

The Waterfall method can be applied either very informally or very formally, as in the US Department of Defense Standard 2167A (and later 498), which describes in excruciating detail every step of the process and the many document deliverables.

Similarly, the waterfall method is also at the heart of the very informal and much more common scenario where someone from the marketing department gathers some market requirements and delivers them to engineering, where they come up with a schedule, and work on an architectural design and then some detailed designs for the more complicated areas, and then move into implementation and testing, often a beta, and finally deployment.

While we will soon discuss the most serious limitations of this approach, it is also useful to acknowledge the key traits that have kept this process in use for so long:

- Management appreciates the (perceived) predictability of the process. It is possible, although not common, to come up with fairly accurate schedules for even large and complex software projects. This assumes however that you completely and accurately understand the requirements and the technology, and that there will not be changes. With iterative approaches you don't really know how many iterations will be required, which can be disconcerting to management.

- There are deliverables throughout the process. Many people (managers, customers/clients, and even many engineers) are reassured by seeing well thought out and thorough documents and design diagrams. It helps these people to gauge progress towards the end, and also helps them feel better about the level of thinking that has gone into the project (even though there is no way to test whether or not the confidence is justified because unlike software you can't execute paper documents). Many people make the mistake of feeling unjustifiably reassured by impressive specifications and documents.

Product Management Concerns

There are a number of well known concerns with this process especially from the product manager's perspective:

- **PM Issue: Validation Occurs Too Late in Process**

The most costly issue is that there is no actual working software until nearly the end of the process, so there is little if any visibility into whether the software will be useful until after the majority of the investment has been made.

The product manager must ensure that prior to moving into the design and implementation phases, the product must be prototyped and tested on actual target users, so that the specification that is eventually provided to the product development organization is describing a product that has been successfully validated with the target audience.

Likewise, if there are major technical risks, these too should be explored and feasibility questions resolved (by the engineering organization) prior to beginning the actual architectural design and implementation. Before proceeding, the team needs to know that the product specification is something that can be successfully delivered.

- **PM Issue: Changes are Costly and Disruptive**

Any change to decisions from previous stages destabilizes the process and causes considerable pain and cost, as much work has to be reviewed and reworked. Moreover, the coding and testing process often uncovers issues in requirements and in architecture that can cause major delays and pain in this process.

The product manager must constantly represent the voice of the customer and user and there will be times when changes are required. It is important to point out that the cost of postponing the change needs to include the cost of the follow-on release to make the correction. There will still be times when it makes the most sense to defer the change until the next release, but in many cases it will be much less expensive to course correct sooner rather than later.

- **PM Issue: Time-to-Market**

This approach has a high overhead in terms of documentation and process for moving through the phases. With the cost of change described above, it is very easy for Waterfall-based projects to end up taking twice as long as the original schedules predicted.

Further, given that you typically don't get useful validation until a beta process or release, it typically takes 2-3 product cycles before the product becomes useful and sellable. This can bring the total time-to-market, when measured from the start to the release of a viable product, to extremely long – often longer than many organizations can afford.

The key to avoiding this problem, as mentioned above, is to move the validation to the front-end of the process, and ensure that you are validating the product before the architectural design phase begins. This puts additional pressure on the product manager to ensure that they are providing a validated specification for a successful product in the first place.

- **PM Issue: Course Corrections**

With the high overhead and long time-to-market described above, one consequence of this process is that it can take considerable time to make even relatively small changes to the software. Even minor update releases can take several months.

The product manager will need to work with the product team to make course corrections after release as quickly as possible.

Process Summary

We have all seen the consequences of the Waterfall process in practice, and it's not hard to understand the motivation for alternative approaches.

In many ways, the Waterfall process represents an idealistic but naïve view of the software development process, where people are able to anticipate the key issues and fully understand the requirements. When this is the case -- usually for very small projects -- this approach can provide a reasonable path to a quality implementation.

Unfortunately, this is rarely the case with product software. In practice, the consequence is that the product ships later than planned due to changes, and then expensive, time-consuming follow-on releases are required in order to correct issues once real users have a chance to see and use the actual software.

However, the product development process is often deeply entrenched in the product development organization, and the product manager must take steps to ensure that the potential problems are avoided. The most important key is to ensure the product spec is validated prior to moving to the implementation phase (remember to cover all three types of validation as described in "How To Write a Good PRD"), and you can save your team significant time and cost.

AGILE DEVELOPMENT PROCESSES

Many product development organizations have been experimenting with what has become known as "Agile" software development processes[1], the most popular of which is known as *Extreme Programming* (XP)[2] but there are several others including *Crystal*, *Adaptive*, *Scrum*, and *Pragmatic Programming*.

In these cases, the product teams often wonder how the role of the product manager fits into this approach. Or, the product manager may be concerned that if their product development organization uses this method then the resulting product will suffer due to some of the more publicized criticisms of the process.

In this note we'll first explain what the core tenets of Agile Development are, then we'll look at the specific example of XP, and then we'll look at how this model impacts the product manager and how to deal with some of the challenges it poses.

General Principles

There are twelve core principles that the various Agile methods all have in common. The ways in which each method carries out the principles vary sometimes considerably, but the common philosophy is:

1. The top priority is to satisfy the customer through early and frequent delivery of valuable software – valuable software early

2. Deliver working software frequently, from a couple of weeks to a couple of months – frequent releases

3. Working software is the primary measure of progress – software matters more than documents

4. Welcome changing requirements, even late in development – listen and learn rapidly

5. Business people and developers work together daily through the project – intense collaboration

6. Build projects around motivated individuals – give them the environment and support they need and trust them to do their job

---

[1] *Agile Software Development, Principles, Patterns, and Practices* by Robert C. Martin, Prentice-Hall, 2002

[2] *Extreme Programming Explained – Embrace Change* by Kent Beck, Addison-Wesley, 2000

7. The most efficient and effective method of conveying info to and within a development team is through face-to-face conversation

8. The best architectures, requirements and designs emerge from self-organizing teams – agile architectures; good ideas from anywhere

9. Continuous attention to technical excellence and good design enhances agility – refactor[3] frequently

10. Agile processes promote sustainable development – should be able to maintain a constant pace indefinitely – no death marches

11. Simplicity is essential – less is more

12. At regular intervals, the team reflects on how to become more effective, and then adjusts its processes accordingly – post-mortems

Extreme Programming Method

To help illustrate what it really means to follow an Agile method, let's look at the specifics of the most popular one, known as *Extreme Programming* (XP). While the creators emphasized that it is not a one-size-fits-all method, and that it should be applied selectively, there are a core set of fundamental principles that form the basis of the method:

- **Pair Programming** – the software is written by pairs of programmers working together around a single computer.

- **Simple Design** – the idea is to design and build only what you need right now and not for some potential future need.

- **On-site Customer** – the developers have a customer that represents what the product needs to do, sitting with the team and constantly available for clarification and decisions.

- **Incremental Development** – the strategy is to start small and do many "mini-releases" to rapidly evolve the product to where it needs to be.

- **Scheduling and Planning** – estimates are provided by the engineers and the customer decides scope and timing for each release.

[3] Refactoring refers to the process of redesigning or reimplementing sections of existing code in order to improve the performance or maintainability.

- **Continuous Code Review** – based on the pair programming model, the developers are constantly reviewing each others' work.

- **Continuous Testing** – the technique is to have the developers write unit tests as they code, and the customer writes functional tests as use cases are defined, and these tests all are run on an automated, ongoing basis.

- **Continuous Build** – the software is constantly built, integrated and tested so that problems are found early and the system is always kept in a buildable state.

- **Continuous Refactoring** – the software developers continually work to simplify and improve the implementation by refactoring the code, while keeping all the tests running.

- **Collective Code Ownership** – rather than the model where each developer "owns" his own section of code, in the XP model every developer can improve any code anywhere in the system anytime he sees an opportunity and need.

- **Open Workspace** – the idea is that the team works together in a large room with the developers in the center.

- **40-hour Weeks** – the belief is that overtime should be limited so that quality remains high.

- **Documentation via Code** – the belief is that the most useful documentation is the software itself, and that the team needs to follow coding standards.

Product Management Concerns

Certainly this is very much a software developer-centric view of the world, as there really isn't much besides the programmers (figuratively and literally at the center) and the user (the customer). However, in general, most of these techniques represent industry best practices and pose little concern to the product manager.

That said, there are several areas of concern for the product manager:

- **PM Issue: The Product Manager Role**

The first has to do with how you figure out what to build. The XP method was originally created for custom software projects to meet a single customer's specific needs (such as an internal employee payroll system), rather than general purpose

products intended for many customers representing a range of needs. In fact, in the books and articles describing the XP method, you will find barely a mention of product management (by any of its many names).

Typically the main concern is this notion of an on-site customer. However, the product manager, to the degree she has studied the target customer and understands his needs, environment and concerns, as well as the competitive situation, can do her best to serve as a proxy for the customer in this process, just as is typically done by the product manager in conventional methods.

However, it is also important to point out that the early and frequent iterations does allow the product manager to quickly identify missing requirements and assess the overall product as it is progressing.

- **PM Issue: The Role of the Product Spec**

While the XP method does employ use cases, also known as stories, there really isn't the concept of a product specification. For custom software this may be fine, but for commercial products there is a significant amount of research and thinking that the product manager must do to prepare to define a successful product (see "How to Write a Good PRD").

Note that this does *not* mean long and tedious requirements documents. The XP belief in software over documents is well-founded. But the product manager must still prepare. However, for the concept validation stage of the product spec process, the XP method can provide excellent support.

- **PM Issue: Primitive Notion of "Customer"**

The method is based on the overly simplistic notion that there is a single "customer" or "user" that the product is for. However, for modern products we know that there are typically several different user profiles, and that each brings different needs and capabilities to the product.

By working with the product designer (see below) the product manager must ensure that each type of "customer" is represented throughout the product development process.

- **PM Issue: The Product Designer Role**

A major concern is the lack of product designer (UI design) role. It is hard to conceive of methods with what we know about software today where the programmers believe they can do effective user interface design, but for many custom software projects, this

is all too often the case. However, for products that must be sold on their merits (versus custom software developed under contract), the user interface/user experience is absolutely critical, and requires the skills of professional designers, so it is important that you incorporate this key role into the process.

These changes can fit into the XP process by treating the first iterations as an evolving, high-fidelity prototype, doing validation to ensure the team will be building the right product, and then proceeding to use the following iterations to develop the product implementation. The key is to ensure you're building a product that customers will want to buy, and will be able to figure out how to use. It is not enough that one customer or the product manager understand it. The product must be validated with the broader representation of the target market.

- **PM Issue: Scalable, Reliable and Maintainable Product**

Another area of concern is with the consequences of some of these techniques for a scalable, high-performance, reliable, and maintainable product. These concerns quickly get into near-religious arguments about the best way to develop and test software, which are largely beyond the scope of the product management role, but the main point relative to the product manager is to ensure the release requirements are clearly defined up front. The engineering organization can then address the concerns of how to manage the risks in the best way they see fit.

- **PM Issue: The QA Role**

The XP process typically looks to the customer to define the use cases (called stories) which serve as the basis for functional tests. On small customer software efforts this may be fine, but for larger, general product efforts, there is a need for a dedicated role to concentrate on identifying and creating the necessary test cases to ensure scalability, functionality, performance, fault-tolerance, localizability, etc. This is typically the QA function, and there is no reason it can't be used with the XP model. The key is that the developers are responsible for the unit tests, and the QA people are responsible for the other types of tests (e.g. system, integration, functional tests).

- **PM Issue: Project Scheduling**

One of the challenges of iterative approaches like XP is that while it is much easier to predict the timeframe for a given iteration, it is very hard to predict the number of iterations that will be required to reach a releasable product. If there are time-to-market considerations, such as major events where the product is scheduled to debut, then this uncertainty will need to be factored in.

Overall, the total time to market will likely be significantly less than with a waterfall approach, but the organization will have to live with the uncertainty in terms of overall timing.

- **PM Issue: Product Deployment**

The final area of concern has to do with deploying the product into production. XP challenges the long-held belief that the cost of change grows dramatically over time. In other words, by following the practices of XP, you can reduce the impact of change once a system is in production. While in general this can be true for custom internal systems, for many types of commercial software products, the impact of change remains as high, if not higher, than ever, especially with Internet services with large, active user communities.

Fortunately, we have discussed elsewhere the need for "Gentle Deployment" practices, and these techniques can help the product team lessen the negative impact of the more frequent releases and updates that XP projects encourage.

Process Summary

The Agile Development Process in general, and the Extreme Programming method in particular, were created to address some truly important problems with the conventional methods of producing software, especially around increasing client/customer communication, reducing the time it takes to figure out if what you are building is something the customer will value, and reducing risk through incremental development and building out the highest priority features first. There are several very valuable techniques, especially around pair programming, incremental development, and continuous and automated test and build.

However, for product organizations building commercially available products and services, it is important to augment these methods with product management, product design, and quality assurance to ensure you're building a product that will be useful to, and usable by, a broad range of users and customers, and that will be robust enough to operate in a range of customer environments.

THE RATIONAL UNIFIED PROCESS (RUP)

Where the XP process is very programmer-centric, the Rational Unified Process[4] (RUP) is more architecture-centric, although not to the same degree. RUP is in fact a much broader software development process, which also includes a methodology for

[4] *The Rational Unified Process: An Introduction*, by Phillipe Kruchten, Addison-Wesley, 2003

how to actually think about architectural design problems and come up with software solutions.

The promoters of the process have been trying to position RUP as an "agile" process, but this is largely creative marketing. It is hard to seriously consider RUP a lightweight process, since it is essentially an iterative method where each iteration is a form of mini-waterfall. That said, the RUP does improve on the conventional waterfall process, and for some product organizations this is a reasonable compromise solution.

General Principles

Like XP, the RUP is meant to be tailored to the needs of the specific product effort, but the RUP has the following general principles:

1. Develop iteratively
2. Manage requirements
3. Use component-based architectures
4. Model visually
5. Continuously verify quality
6. Control change

In general, the RUP encourages iterative development – which in this case you can think of as a series of mini-waterfall cycles. But unlike XP, there is still considerable formalism in how each iteration is performed, especially in the areas of visual modeling and architectural design.

With RUP, there are four overall stages in the lifecycle:

- **Inception Stage** – define the business case and the project scope
- **Elaboration Stage** – refine the requirements and design the architecture
- **Construction Stage** – build the system to the point of a beta release
- **Transition Stage** – finish the product and reach general product release

Each of these stages may have zero or more iterations (mini-waterfalls).

There are several important benefits to this approach over the basic waterfall:

- Serious issues in terms of value and feasibility are identified much sooner than with traditional waterfall
- Risks are managed proactively and realistically
- This approach can yield a solid and sustainable product architecture

Product Management Concerns

Overall, the RUP is fairly "product manager-friendly" as far as software development processes go. The product manager role (by different names) is accounted for in the Inception and Elaboration stages, and the iterative nature has validation and course-correction designed right in.

That said, the architecture-centric perspective creates a few key areas of concern for the product manager to pay special attention to:

- **PM Issue: Overemphasis on Visual Modeling**

The company behind the process is the major provider of visual modeling tools, and it is no surprise that these tools, and the methodologies they support, play a central role in this method. Kept in perspective, these tools can occasionally be useful to the team to visualize and communicate important concepts, especially relating to architecture.

However, it is very easy to make the mistake of putting too much faith and confidence in an elegant picture. The experienced product manager knows that there is no real substitute for software. You can show modeling diagrams to end-users and they'll often love them, but turn out to hate the system that is actually generated. It is much more valuable to create usability prototypes to assess whether end-users will be able to figure out how to use the product and judge its value.

- **PM Issue: Dangers in Starting Architectural Design before Validation**

There is a dynamic that happens as soon as the architectural design begins. Coming up with a good product architecture takes a good deal of effort and while the theory is that you can make significant changes when necessary in future iterations, the team quickly becomes hesitant to make changes at the architectural level, and therefore the product can quickly become constrained. The key is to validate the product's functionality and usability prior to any architectural or implementation work.

- **PM Issue: Emphasis on Building Product Right versus Right Product**

While the RUP approach is better in this respect than the waterfall method, the process is still biased towards building a quality implementation of the product, which

it does support well. The product manager needs to ensure that the product team understands the importance of validating the product's value and usability in addition to the feasibility.

Process Summary

The Rational Unified Process is a heavy-weight process suitable for larger, mission-critical type projects – both commercial product efforts and large custom contracts. It is especially comprehensive in that it also includes a development methodology which helps guide in coming up with good architectural solutions, as well as a rich set of supporting tools.

The RUP does address some of the limitations of the conventional or waterfall lifecycle. Moreover, the process is compatible with several good product management principles. The main weaknesses for many teams is that it is very process and methodology heavy, and these teams may prefer a lighter approach such as one of the Agile processes.

SUMMARY

The product manager will not usually be in control of the software development process that the product development team uses, so it is important to understand the challenges that each process presents. After 30 years of trying, there is still no perfect product development process. And it is also still true that the determining factor is usually the quality and experience of the team members rather than the process or tools.

That said, there are several strategies that the good product manager can use to increase the chances of delivering a great product regardless of the product development process used.

About the Author

Martin Cagan is the Managing Partner of the Silicon Valley Product Group. During the course of the past 20 years, Martin Cagan has served as an executive responsible for defining and building products for some of the most successful companies in the world, including Hewlett-Packard, Netscape Communications, America Online, and most recently as VP Product Management and Design for eBay. The Silicon Valley Product Group (www.svproduct.com) is dedicated to serving the needs of the high-tech product management community by providing content, services, and professional development for product management organizations worldwide.