



The Product Marketing & Product Management Experts™

## Embracing Agile Development

*by Greg Cohen*

Back in 2000, at the height of the dotcom, I was working with a small IdeaLab! company in Silicon Valley. IdeaLab! had a philosophy of seeding a number of small projects, putting them out in the real world as quickly as possible to observe actual performance, and then iterating very fast. This meant parallelizing a number of tasks that would typically happen in serial, such as requirements gathering, requirements definition, UI design and testing, software development, and QA. This forced the team to rethink everything its members had ever learned about waterfall development.

We went from concept to beta in two months and started attracting real users, real distribution partners, and real advertisers for our nascent product (historical note: we had to sell advertising directly back then. Google AdSense didn't exist, and Yahoo! was the number one search engine.) From there we settled into a five week release cycle: three for development and two for QA. Although this may seem fast, it was still TOO SLOW. Requirements grew and priorities changed with each day of actual customer use and every partner sale.

The team turned to an Agile development method known as Extreme Programming or XP for short. XP emerged as a method for custom development for IT shops in 1996 and was pioneered by Kent Beck. In its original form, developers sat near the actual users and there was no product manager acting as a go between. Although the name XP may evoke images of bungee jumping and other adrenaline junky sports, the term "extreme" was used because it combined a number of existing development techniques and thus carried them to an extreme.

The methodology has twelve practices, five of which were particularly relevant to me as a product manager.

1. **No requirements document** – In XP there are no lengthy requirements documents. Each requirement is placed on an index card as a story. The index card acts as a placeholder for a conversation between the product manager and the developer. XP does not actually state whether you should or should not write requirements documents and for more technical requirements, a separate document would be created. But the important aspect is the conversation, which drives a shared understanding of the requirements.
2. **Relative sizing** – On each index card, the developer would write the estimated time to complete the requirement. The estimate is measured in ideal engineering time, or the time it would take to code the requirement without any interruptions. After each week of development, we added up the estimates on the completed



### **The Product Marketing & Product Management Experts™**

- cards. This gave us a fast and accurate measure of actual throughput. We then assumed the following week's throughput would be the same as the previous week. Although you could certainly point out flaws in this method, it remained surprisingly accurate. It also had the benefit of normalizing engineering estimates from those engineers who were either too optimistic or too pessimistic. As long as the engineer consistently overestimated or underestimated, we could gain an understanding of the real time required after the first week of development.
3. **Release planning** – There is nothing easier than planning an XP release. I would just place the index cards in priority order. When a priority changed, I just reordered the cards. Since I also knew our throughput, thus how many units of work we could complete in a week, I knew exactly where the team would be at the end of our three week development cycle. Thus, I knew the requirements that would make the release and the ones which wouldn't. If I had to get a feature in, I could always extend the development cycle, but I resisted this temptation as it throws off the cadence of the team, and the next release was only three weeks away.
  4. **Unit tests** – In XP, engineers are required to write the unit test for each requirement on which they are working before they write a single line of code. This forces the developer to think through the design and the boundary cases upfront and seek clarification where needed. It also lets the engineer know when his or her code is complete and produces a rich automated test suite. If a bug is found, it is added to the unit test and never re-occurs in production.
  5. **Continuous integration** – The team checked in code daily and ran it against the unit test suite. On any day, we could push the release if needed. The release might not have all the features we wanted, but it was stable and could be put into production. It also meant that I could look at the working code anytime during the release cycle to confirm that the design met the requirement.

The net result was the team became dramatically more efficient. Our quality immediately went up, freeing the engineering team to focus on new development rather than bug fixes. Most astonishing, we were able to shrink our QA cycle from two weeks to four hours. We went from releasing every five weeks to every three weeks, a remarkable productivity and flexibility boost.

I found XP liberating as a product manager, which might seem like an unusual word to describe a development methodology. It meant we didn't have to know all the answers, and we could adapt to the changing market and corporate priorities with relative ease. In fact, with our three week release cycle, we were able to re-prioritize and adjust course 17 times year. That's pretty amazing!



### **The Product Marketing & Product Management Experts™**

The engineers, who had previous experience with the methodology, liked XP too. They did not have to slog through 100 page documents and sit through tiresome meetings. Instead they spent more time programming and every piece of code they developed was deployed and used. Let's face it; there is nothing more demoralizing than having months of work thrown out because the priorities shifted. With XP that never happens.

Still XP has its limits. In particular, it becomes very challenging once you have more than 10 developers. Therefore, it is not appropriate for many situations. Fortunately Agile development, which took from XP and other lightweight methodologies, can work for larger projects.

Guidewire, located in San Mateo, CA, exemplifies how Agile can be leveraged across a large team and complicated product offering. The company develops a web-based claims management system for insurance carriers around the world. Guidewire's integrated suite of policy, claim, and billing applications is mission critical, encapsulates complex business rules and workflows, and is responsible for billions of dollars of claims. Furthermore, their customers cannot absorb frequent updates because that would require re-training and disruptive change.

Guidewire, therefore, releases to their customers once per year; but they develop around eight monthly new feature sprints plus four monthly QA and tuning sprints. To maintain flexibility over the annual release cycle, Guidewire commits no more than 70% of their development capacity upfront. Between each sprint, the company reprioritizes the release.

More impressively, Guidewire's 70 developers maintain 10 active branches of code. They pull from stable daily and push to stable approximately once per week. Each time an engineer checks in a new change list to a branch, a new build is generated and 30,000 automated tests are run against that build within 30 minutes. This allows the developers to receive immediate feedback on whether their check-in caused new problems. If there are no test breaks, the branch can then be pushed to stable.

By using Agile development, Guidewire enjoys fewer problems in QA, visibility into stable code, lowered maintenance costs, and the ability to reprioritize every four weeks. Further, the company has significantly lessened schedule and quality risks from late changes.

In summary, product managers realize three primary benefits from Agile development:

- **Flexibility** – You no longer have to predict the future. You can incorporate real time data into your planning, and you can reprioritize the release with each development cycle without disrupting the team and schedule. This is not to say you do not need to do your homework, but it accepts the fact that markets are



**The Product Marketing & Product Management Experts™**

dynamic. If you could predict them with 100% accuracy, you would have probably taken up day trading instead.

- **Visibility** – frequent builds of stable code allow you to see requirements as they are coded and even test them with constituents throughout the development cycle. Further, you can easily measure your progress over the release.
- **Quality** – unit testing ensures high quality code, prevents old bugs from reappearing, reduces maintenance, and lowers risk of late changes.

The benefits of Agile can make you exponentially more effective as a product manager, which is why I say “embrace Agile development.” It is your best friend.

*Greg Cohen is a principal consultant at the 280 Group and on the board of the Silicon Valley Product Management Association. He has over a decade of product management and marketing experience, including Software-as-a-Service, channel sales, open source software, and agile development.*

*copyright 2007*