The Development of Net Perceptions Personalization Manager

Peter Clark

pclark.net Consulting, LLC
1736 Berkeley Ave
Saint Paul, MN, 55105 USA
+1 651 698 9917
pclark@pclark.net

ABSTRACT
This paper discusses the author's experiences as the engineering manager of the Net Perceptions Personalization Manager for E-Commerce product. The development effort utilized several "agile" techniques in an effort to deal with rapid requirements churn. These techniques allowed the team to ship on schedule despite changing requirements.

Keywords
Agile development, timeboxed scheduling, personalization

1 INTRODUCTION AND BACKGROUND

Net Perceptions[1] is the market leader in personalization and recommendation tools for e-commerce and call centers. By early 2000, the company's core technology, the Net Perceptions Recommendation Engine, had won numerous awards, several patents, and had garnered numerous customers for the company. However, the recommendation engine frequently required substantial integration work and lacked the supporting infrastructure required to make the system easy for non-technical marketing staff to use.

The Net Perceptions Personalization Manager for E-Commerce[2] product, referred to as NPPM, was intended to address these concerns. The product vision was to extend the recommendation engine with a rules system, provide a simpler API, and add a web-based user interface and reporting system to allow marketers to control promotions on an e-commerce site. The details evolved a lot, but this vision was constant throughout the development effort.

As this was the company's flagship product, the planning process was intense and included a broad cross-section of the product stakeholders – marketing, support, sales, testing, development, and senior management. These groups brought different perspectives to the table, and they were asked to collaborate more closely than they'd done in the past. Everything was up for discussion, and debates continued throughout the development lifecycle as each group learned more about what the marketplace wanted.

The development team set up a process that allowed us to make steady progress in the face of steady change. This report will focus on that process, and our lessons learned.

2 BIRTHING PANGS - GETTING STARTED

During early 2000, the development team comprised twenty-four developers broken out into four teams, under three different engineering managers. Each team was initially responsible for delivering a maintenance release of one of the company's three existing application products, as well as rapidly building a prototype of NPPM.

This proved to be a flawed strategy. The need to deliver maintenance releases was a distraction and added to the coordination challenges. The teams selected tools optimal for their own tasks without an eye towards integration. The front-end user interface was developed in Microsoft ASP, which offered the fastest path to a demoable user interface prototype, and a back-end written in Java, which allowed the most leverage of the prototype rules technology. Tying these two systems together would present problems.

During the time that the engineering groups were working on completing the maintenance releases, the marketing team had fleshed out a complete first pass at requirements and functional specifications. These specs were not finalized, but were a good starting point.

3 FINDING OUR STRIDE - THE NPPM DEVELOPMENT PROCESS

Although the requirements docs provided a good start, they weren't detailed enough to code against. As the marketing and product management team spent time with customers, the requirements evolved rapidly – many features that were "must-have" one week were "off the list" the next. The team adopted several techniques to react to this.

Timeboxed Iterative Development

Timeboxed, iterative development had not been used much at Net Perceptions prior to this project. The team adopted timeboxed iterations for four reasons: risk management, clarity of tasks, predictable rhythm, and insulation from constant change. These will each be touched on below.

Risk areas: The main user metaphor for NPPM was that of a rules-driven marketing campaign, but the engineering team wasn't quite sure what that meant. Our first iteration was to ignore all the proposed features except for one: process and execute the simplest campaign possible. This let us focus; there was a concrete, measurable finish line to the iteration, and a reasonable amount of time allotted to get there. We similarly picked the high-risk areas for each subsequent iteration and tackled them first.

Clarity: Each iteration started with a concise list of stories that the system needed to support, which usually fit on one page. This provided clear goals for the team. When work couldn't be completed in the allotted time,

stories were moved to the next iteration, and we budgeted some time in the schedule to allow for that to happen occasionally.

Rhythm: Each iteration was timeboxed to be 6 to 8 weeks in duration: 5 to 6 weeks of coding, with 1 to 2 weeks of system testing and integration time at the end of each cycle. The stories to be implemented within an iteration had to fit within that time. This allowed the team a set amount of time to work independently, with time budgeted for integration and testing by the testing group.

Insulation: With the support of senior management, the product team was instructed that the development team's task list within the current iteration was fixed. The stories for future iterations could be changed as needed to reflect changing priorities, but the current iteration was off-limits. This gave the development team a fixed target within the context of a single iteration.

The stories for an upcoming iteration were negotiated between the engineering manager and the product manager prior to the end of the current iteration, with review from the entire product team. There was lots of give-and-take in the discussion, as proposed new stories were prioritized and estimated. The timeboxing drove the prioritization effort and gave the product management team the comfort that if a story slipped out of current iteration, they'd have an opportunity to review it for the next one.

Establishing the use of timeboxed, story-driven iterations to control the impact of scope change was the most significant management practice we put into place. It allowed us to react quickly to changes, without letting those changes throw the development process into chaos.

Automated Unit Testing and Daily Builds

The Java-driven side of the team used the JUnit[3] testing tool to build automated tests into the code. As the code changed to respond to new requirements, JUnit provided rapid feedback when we broke things. Developers were able to test their code quite comprehensively, on demand. The JUnit tests were also tied in to the build system.

The back-end system was set up for daily builds, which automatically checked the entire source tree out of the revision control system (CVS), built it, ran automated test scripts (supplementing the JUnit tests), and mailed the results to the entire development team. This gave us daily feedback on the system's health.

Focus on the Simple

The development team tried to avoid complexity where possible. When we couldn't, we tried to hide it behind simple interfaces. The interface between the aforementioned ASP front-end and Java backend was via an XML document stored in a database. Both systems had to generate and parse XML, but the XML specification was codified in a DTD and could be automatically checked.

We also looked for ways to reduce overall complexity. As a large database-driven Java system, the use of an Enterprise Java Bean (EJB) app server would seem natural. However, we determined that we couldn't justify the added complexity, both development- and deployment-time. The system was built as a set of Java servlets running inside the Apache Tomcat [4] environment.

Similar accommodations were made for reporting tools.

Tight Customer Interaction

Both the development team and the product management team worked closely with a small set of "lighthouse customers" – early adopters who committed to working with us to help shape the product. We demoed incomplete, development builds of NPPM to various representatives from these customers and took their input back to the development team to integrate into the next iteration.

4 CONCLUSIONS

The schedule only slipped once. Our first committed release date was at the end of November, which we committed to in late July. In late September, as we neared a feature-complete version of the product, the product team agreed to move the ship date 4 weeks, to December 22nd, to accommodate additional testing and custome requests

The product did in fact ship on December 22nd, and was successfully deployed at several customer sites by the end of Q1 2001. Anecdotal comments from the QA team were that this was one of the highest-quality pieces of software Net Perceptions had ever produced. The use of story-driven timeboxed iterations was key to making our date and managing requirements throughout the development effort. The use of JUnit and nightly builds gave us instantaneous information on the health of the codebase. The focus on simplicity kept us from over-engineering, and the customer involvement kept everyone focused on why we were building the system in the first place.

ACKNOWLEDGEMENTS

The work of the Net Perceptions for E-Commerce 6.0 team was incredible. This paper is dedicated to their effort. Any errors in this paper are the sole responsibility of the author.

REFERENCES

1. Net Perceptions (NASDAQ:NETP). See http://www.netperceptions.com
2. As of this writing, product information for Net Perceptions Personalization Manager is available at: http://www.netperceptions.com/solutions/retail/
3. JUnit is an open-souce automated regression testing tool. See http://www.junit.org
4. Tomcat is the open-source reference implementation for the Java Servlet 2.2 specification. See http://jakarta.apache.org/