Yahoo! Sports: Sprint 100 & Beyond

Keith Nottonson
Yahoo! Sports
Sunnyvale, United States
keithn@yahoo-inc.com

Abstract — Yahoo! Sports is the web’s leading sports destination. This paper examines three transformative events that enabled the Yahoo! Sports team to perform daily code releases, deliver innovative features to users quicker and learn from their data faster.

Keywords - lean, agility, speed, frequent releases

I. Yahoo! Sports

In February 2010, my wife and I watched the Winter Olympics from our hospital room with our newborn son. Little did I know that my work life would soon revolve around sports; working with the Yahoo! Sports team to perfect the art of the daily release, creating a process for delivering innovative experiences, and revisiting the prioritization of all existing documented work.

Running websites visited by hundreds of millions of users a day is complex. At Yahoo! hundreds of development teams rely on one another for code and services. User facing sites, such as sports.yahoo.com, use a lot of code written by other teams. Many of these teams have been influenced by Scrum, with some teams implementing Scrum better than others.

Yahoo! Sports began Sprint 1 in September of 2006. The team was comprised fully of developers. The team ran mostly 2 week sprints with the code being deployed 3 days after the end of the sprint. The team adjusted to overlapping 3 week sprints when quality assurance joined the team with the first 2 weeks being focused on development and the last week focused on test. However, during the test week, the developers would begin the next sprint. So now code was being deployed 8 days after the developers finished coding. This delay from when the code was written to when it made it to the user frustrated the developers.

Unlike other teams I had been on before, the Yahoo! Sports team did have very fixed dates - sporting events. Hardly a day goes by without one. This required the team to trade off scope and occasionally quality rather than schedule and resources, which were usually fairly rigid constraints. When the Sports team encountered obstacles, a speedy resolution was necessary.

I have heard it said that the worst customer for a Scrum team is another Scrum team. For starters, teams have different sprint lengths, velocities and priorities. Often Scrum teams do not want to be distracted during their sprint. Teams may find the team on which they have a dependency has yet another dependency on a different team. A scrum team which is blocked may find it cannot be unblocked and is forced to pursue a different alternative. Sometimes a sporting event would occur before another team could commit to resolving a dependency, which then required the Sports team find a solution themselves. Over the years, this had the effect of making the Sports team very self-reliant and nimble.

Due to this intense release schedule, the high demands of the product, and the many dependencies on other teams, we knew we couldn’t operate as usual. This paper examines three transformational events that enabled the Yahoo! Sports team to keep pace with the demands of the online sports world while maintaining team cohesion, motivation, and high quality. We will examine three transformational events and their impacts on the team: daily releases, the Hacktacular and the Great Bug Sweep.

II. First Transformative Event: Daily Release

Before the World Cup, the Sports team was deploying large feature releases that occurred at least every two weeks. These feature releases suffered over multiple delays. Early on it became apparent the further the releases were apart, the more features 'had' to be in a particular release as many features were time sensitive and could not wait several weeks for the next release. As the World Cup event was set to begin, it became apparent that no matter how well we delivered code; there would always be something that had to be pushed daily. With the size of such an event, performance enhancements and tunings were practically required daily (at least near the beginning of the event). Also, with the sheer volume of users in 22 markets, there was a high likelihood there would be a critical issue that needed to be resolved quickly.

These off cycle releases had ad hoc procedures for deployment. Subsequently, releases had heavy coordination and communication costs, such as identifying what needed to be changed, who was going to make the change, scheduling the change, and coordinating the release of the change. We hypothesized that if we performed daily end user code pushes we could limit confusion, decrease coordination costs, and deliver a better experience. Fortunately, our build and release processes were technically proficient so we could deploy our code around the world in less than an hour. We used the 10 hour time difference with South Africa to our advantage by pushing
Our goal was daily releases, which we met on most days. However, we had a hard and fast rule: if the code branch was not ready by a specific time, we would not release it. We were able to make these hard calls because continuous learning allowed our process to evolve.

During the daily release retrospectives, the team identified three themes on which to concentrate that enabled them to achieve this aggressive cadence:

A. **Better Flow**
- The team had to remain flexible when implementing a new, fast process, in particular, the timing of handoffs. We reviewed the timing of specific handoffs between teams everyday and made changes frequently during the first week as we learned our lessons.
- The more automation the better. We focused on build automation as well as test automation by having the developers and testers share test development responsibility and do their work in parallel.
- Continuous integration had to be more reliable and set up such that there was no confusion around which branch to check in. At first, there was more overhead in the form of branching, environment upkeep and daily triage meetings. Over the course of the World Cup the team got much better and faster.

B. **Smaller Pieces**
- We needed to deliver scaled-down versions of features sooner by working on smaller chunks of code. Over time the team got better at identifying the minimal marketable feature and delivering smaller increments of code to production to a small subset of users.
- To maintain flexibility at release, we needed to compartmentalize the code by implementing feature flags that allowed code to be checked in and turned off, so it would not appear in production.

C. **Culture Shift**
- At first, the daily handoff from dev to QA tended to drift past the allocated time block. The team had to adjust their mindset. In the past, it might be weeks before the next deployment so getting a feature in at that last moment was often necessary.

With a release happening every weekday, missing a particular release was not such a hardship, rather that feature would deploy the following day.

- There was very little time (or none) allotted in the cycle should a regression be found. A bug could be pushed to the next day but a regression would push the release date. The team had to realize not all regressions necessarily block the release. Rather regressions need to be prioritized against the value of the work in the release. Also, the QA team spent a lot of time automating to find regressions sooner in the cycle so developers could fix them in a timely manner.

III. **SECOND TRANSFORMATIVE EVENT: THE HACKTACULAR**

Like many exciting Internet companies, Yahoo! promotes creativity and problem solving by holding “hack days.” During these hack events employees work outside the confines of their normal work demands to pursue their own innovative ideas for new products and tackle problems in new ways.

Due to conflicts with scheduled sporting events, the Yahoo! Sports team often could not participate in hack days. So the team scheduled a hack event for themselves, the Hacktacular. The Hacktacular was a team-building exercise in which the team got to work on innovative projects they themselves chose. We gathered the teams working together on the project in one location and allowed the team to work on ideas not on the roadmap. Our Hacktacular had four parts:

1. Hacksplanation - event introduction
2. Hack Shoppe - people suggest innovative ideas in hopes of gaining interest and support
3. Hacktacular - teams create the selected product feature
4. Roundup – product demonstration

A. **The Hacksplanation**

During the Hacksplanation phase of the event, we provided the Sports Team with current user-engagement metrics, which we compared with our yearly target. We then discussed the goal of the event: release engaging features that would significantly increase visits-per-user. Next, we established the guidelines: deploy the hacks within one week of the event, work with at least three cross-functional team members who preferably worked in a different locale, and have fun.
B. The Hack Shoppe

Everyone with an idea was given 60 seconds to pitch their idea and field questions from the group. Some ideas were introduced by a team and some individuals presented multiple ideas. 30 hacks were presented in 37 minutes. Afterwards we decided which ideas to pursue and created teams to deploy each idea. The teams were given several weeks (referred to as the pre-hack period) to whiteboard ideas and designs, consult with User Experience Designers, and plan their hacks to varying degrees.

C. The Hacktacular & Roundup

The Hacktacular was a three day event where each of the product teams gathered to implement their ideas. On the last day, we held a “Roundup” where each team demonstrated their project to the group. In the end, 8 products were delivered.

Before embarking on our second Hacktacular in March 2011, we performed a retrospective. We learned three major lessons from the first event.

First, the scope was too large for most projects. While we did release five out of the eight demonstrated projects, none met the goal of launching within a week of the Hacktacular.

Second, rather than focusing a single user engagement metric, we should have focused on instrumentation, which would enable us to rapidly adapt to our customers’ behavior.

Third, we should have focused on delivering a part of the feature and then used customer feedback to improve and enhance the entire feature. We realized that in a worst-case scenario we could always turn-off a feature that simply didn’t work.

Finally, we should have invested more time in the pre-hack activities, such as problem definition, solution exploration, UED design tasks, and defining the minimal viable product.

IV. Third Transformative Event: The Great Bug Sweep

The Great Bug Sweep began as a simple clean-up of the Sports team’s Bugzilla system, the Sports team’s defect tracking tool introduced in 2001. In 2006, they expanded the program to also track feature work and tasks. All tasks in the system were prioritized P1 (most important) to P5 (least important). My original intention was to clean-up the open tickets in order to get more reliable data for lead time metrics and cumulative flow diagrams.

Bugzilla, which had been in use by the team for a decade, contained over 30,000 ‘zombie’ tickets, neither open nor closed. Over the past 10 years these tickets had been marked “resolved fixed” but not “verified” or “verified fixed” and not “closed.” So I became a zombie slayer. I closed all of these tickets without remorse. This reduction allowed the data charted by the diagrams to become more useful. My end goal was to create flow between the p1 and p2 tasks and planned swim lanes for projects designated p3 thru p5. This meant redefining how the team set priorities and used the Bugzilla program.

The distribution of open tickets by priority was a bell curve skewed to the higher priorities. A quarter of the tickets were P3. What was astonishing was the percentage of open tickets with no priority was 35.6%. These were the first tickets I started reprioritizing, banishing more than half to P5 based solely on the criteria of open date, leaving only the tickets opened over the past six months for the other four priorities.

The end goal of reshaping our prioritization was to enable a more flow-based system. Therefore, we made a deliberate decision to reshape our priorities such that P1 represented real-time situations, P2 unplanned work, and using the full breadth of P3 thru P5 to prioritize planned work taken from the product backlog.

There were several take-away lessons from the Great Bug Sweep. First, in order to actually create a cultural shift, I had to make the process completely transparent. The transformation of the data in the tracking system was done over several months, all the while discussing issues with my team. For instance, while reviewing the 200 plus open P1 tickets, I never reprioritized or closed a ticket without first discussing it. This provoked conversation within the team about what were our true priorities. Now P1 priorities, which have remained in the single digits for months, actually reflect real-time priorities.

Second, it is critical to keep all the information in one central place. When I began to clean-up Bugzilla, I realized that the team also tracked projects in various other spreadsheets and wikis. This decentralized system impeded the team’s ability to address real-time priorities. So I synthesized all the information into one central place, Bugzilla. After this was done, constant pruning and steady triage was necessary to keep the information useful.

Third, use this wealth of centralized information to learn lessons quicker. You need to constantly look at the data, compare results to expectations, discuss why expectations were not met, share this information with the team, and integrate this information into establishing future goals.

V. Sprint 100 & Beyond

Sprint 100 took place five years later, lasting one week. The product owners on the sports team were fully engaged and adept at making decisions in real-time. This allowed the freedom to not have to track work in hours. Rather, the team focused on a forced rank sequence of work that needed to be delivered and adjusted priorities accordingly. At the planning meeting, the team reviewed the sprint priorities, identified next steps and discussed any blockers, with the meeting time boxed to one hour.
Stand ups took place on Monday, Wednesday and Friday at 11 AM. The stand-up was attended by over 20 people with two offices connected by video conferencing and a few people dialed into the audio bridge. The Sports team did not enforce the 15 minute hard constraint, partially because it was not a daily standup and partially because they liked each other, but mostly because this was an effective and fun communication channel. Every day following the standup, members of the team would remain in the rooms and on the phone to review mocks and designs, demonstrate working software, or explore ways to remove obstacles.

Since demonstrations of new functionality occurred throughout the week whenever a feature was ready, the Sports team did not have a formal sprint review session. To bring a sprint to a close, we would adroitly review unresolved tickets remaining in the sprint queue and share a short retrospective before planning the next sprint. The release branch would be cut later that afternoon and the code deployed to production for our users’ enjoyment the following Tuesday. The Sports team could also push code daily, reserving this release pace for large sporting events like the Olympics, the World Cup, and the beginning of the March Madness Tournament.

VI. CONCLUSION

One year after joining the Yahoo! Sports team, my son walks and dribbles a small ball and my team celebrated a successful March Madness, participated in their second Hacktacular, and released Sprint 119 a day after cutting the release branch. This coming week the NHL Playoffs begin followed three days later by the NBA playoffs.

The Yahoo! Sports team for me began as a journey performing daily releases during the World Cup where we focused on better flow, smaller pieces and the supporting culture adjustments. We then mixed it up and created space in our schedule to throw the Hacktacular that encouraged the team to work together in new and interesting ways cross-functionally and co-located. Finally, with the attention to detail in our work queues, we can more easily prioritize and deliver the most valuable work first with little delay.

My wife lovingly calls my son Bumps because he is always smashing into walls and falling down as he learns to walk and run. I feel a lot like my son must feel as I help guide the Sports team to quicker and bigger wins. You have to take risks. You have to venture beyond your comfort zone. If something does not work the first time, you have to try again. If one thing does not work, you have to try something else. You have to let them feel pain. You have to keep it fun. And you have to keep giving energy and love, because falling down hurts, but learning to run is freedom.

ACKNOWLEDGMENT

I would like to thank Rhea Stadick for her direction, assistance and guidance.