Why Software Is Like Baseball

Ricardo Valerdi

**THERE ARE MANY** parallels between software and baseball. Besides the teamwork involved, an individual’s contributions to the outcome are important in both settings. Put on your thinking cap for a moment as we explore how programmers can be evaluated the same way baseball players are. (For those of you unfamiliar with the sport, the “Baseball Basics” sidebar provides a very brief tutorial.)

**Statistics, Sabermetrics, and Software**

The convergence of technology and sports has gained tremendous momentum as professional leagues have found ways to connect with a new generation of fans via mobile devices. While watching or attending a baseball game, fans might toggle between social media apps such as Twitter and MLB.com At Bat to check on game-related information and statistics. (For more on MLB.com At Bat, see the related sidebar.)

One such statistic is a team’s *win probability*, $p(\text{win})$, which is based on each play’s outcome. Both teams begin a game with $p(\text{win}) = 0.5$, but after the first result occurs on the field, the probabilities change. For example, scoring a run will increase a baseball team’s $p(\text{win})$ by a certain percentage while reducing the other team’s $p(\text{win})$ by the same amount. The sum of both teams’ $p(\text{win})$ will always equal 1.

The sports industry has heavily invested in such analytics for some time. Finding inefficiencies in the industry has given teams and coaches a slight edge over their competitors that might translate into

- more wins per million dollars (*win efficiency*) and
- additional revenue through ticket sales, corporate sponsorship, and television contracts (*franchise value*)

Professional baseball in particular has a tradition of analytics. This was documented in the book *Moneyball*,¹ which suggested that

- players should be evaluated on past performance rather than potential;
- certain metrics are overvalued, rewarding individual behavior rather than team behavior; and
- there’s hidden value in recruiting often-overlooked players who value team performance rather than expensive superstars who value individual accomplishments.

Such thinking led to the formation of the baseball analytics movement, now called *sabermetrics*. Sabermetrics uses data to make objective decisions about which players to draft, which players to play, how much to pay players, and which personnel trades between teams make the most sense.
Applying Moneyball-type thinking to software projects might help software teams find hidden value and operate more efficiently and effectively. Two sabermetrics concepts come to mind: quantifying a player’s ability to score runs (using the runs created metric) and a player’s ability to help a team win (using the win probability difference metric).

**Runs Created**

Bill James invented the runs-created metric to estimate the number of runs a hitter contributes to the team. To explain why this metric is essential, James used the example of Willie McCovey, a first baseman for the San Francisco Giants in the 1960s and 1970s who was inducted into the US National Baseball Hall of Fame, an honor reserved for the top 1 percent of players:

> With regard to an offensive player, the first key question is how many runs have resulted from what he has done with the bat and on the basepaths. Willie McCovey hit .270 in his career, with 353 doubles, 46 triples, 521 home runs and 1,345 walks—but his job was not to hit doubles, nor to hit singles, nor to hit triples, nor to draw walks or even hit home runs, but rather to put runs on the scoreboard. How many runs resulted from all of these things?

James was arguing that these numbers don’t tell the entire story of McCovey’s career. Rather than focusing on individual metrics, James suggested that the most important metric should be how many runs resulted from McCovey’s contributions. This shifts the focus from individual outcomes to team outcomes, importantly so because runs help a team win.

The conceptual framework for runs created ($RC$) is

$$ RC = \frac{(A \times B)}{C}, $$

where $A$ is the on-base factor (how many times the batter got on base), $B$ is the advancement factor (a weighted sum of the number of bases a batter gained with his or her hits), and $C$ is the opportunity factor (how many times a batter had the opportunity to hit).

The analog in software development is that an individual programmer’s contribution could potentially be measured in terms of thousands of software lines of code created (KSLOCC). However, I suggest measuring the project team’s productivity instead of individual programmer productivity, using this formula:

$$ KSLOCC = \frac{(D \times E)}{F}, $$

where $D$ is the KSLOC created by the team, $E$ is the complexity weights for more difficult KSLOC, and $F$ is the team’s effort in person months.
In other words, instead of measuring an individual programmer’s productivity by using the standard KSLOC-per-person-month metric\(^3\) at the individual-programmer level, I propose two modifications:

- Measure the KSLOC generated by the project team rather than the individual.
- Weight the software by complexity to account for more difficult features or modules.

These measures would work only when measuring productivity at the team level makes sense and when a clear definition of a team exists.

The Win Probability Difference

As I mentioned before, a team’s likelihood of winning a game can be quantified in terms of probabilities throughout the game. This makes a sport more interesting for fans who might want to understand how certain plays affect the game’s potential outcome. It’s also exciting to see when a team with a very low probability of winning suddenly comes back and steals the win from another team.

Multiple factors drive a team’s likelihood of winning a baseball game. One factor is whether the team is playing in its home stadium. Another factor is the sequence of events of the game itself. If the score is 5 to 0 at the game’s early stages (with 6 of 27 outs recorded for the team that’s ahead), the probability of a win for the team that’s ahead will be lower than if the score was the same near the game’s end (with 26 of 27 outs recorded for that team).

The events that lead to the offensive production of runs or defensive production of outs can be attributed to individual player contributions. In line with the earlier discussion about the need to emphasize team behavior over individual behavior, the win probability difference emphasizes how much a player helps his team on offense and defense.\(^4\)

A recent example is the 29 May 2017 game between the Houston Astros and Minnesota Twins. As mentioned before, both teams began with \(p(\text{win}) = 0.5\). By the fifth inning, the Twins were ahead 7 to 2, with \(p(\text{win}) = 0.95\). The game remained in favor of the Twins until the Astros scored in the 8th inning, shifting the Twins’ \(p(\text{win})\) from 0.76 to 0.27 with a single play: a double by Josh Reddick. The difference that double made in terms of \(p(\text{win})\) certainly had the most impact, despite accounting for only two of the team’s 16 total runs. (For a graph of how the \textit{win expectancy}, which is akin to \(p(\text{win})\), changed throughout the game, see www.fangraphs.com/livewins.aspx?date=2017-05-29&team=Twins&dh=0&season=2017.)

The analog to software also pertains to the probability of a successful outcome, which might be cost, schedule, or performance based. A project team member might accomplish a certain milestone, make a technological breakthrough, achieve customer approval, or perform a test that could increase the project’s likelihood of success. As with most project schedule estimates, there are optimistic expectations that the project will be completed on time. If these estimates were updated at each significant event, the team would know its likelihood of success.

Of course, an important difference between baseball and software is the role of external factors that influence the outcomes. In baseball, most outcomes are decided by skill or luck. Some are influenced by external factors such as weather or crowd noise. In software, external factors such as personnel turnover, financial crises, and customer delays might play a much more significant role in the project’s success.

As with the introduction of any new metric, there are unintended consequences. Measuring certain things might lead to a change in behavior that’s desirable in the short term but undesirable in the long term. My goal here has been to provide a different view of how to measure software projects by borrowing from the playbook professional baseball teams use to measure and evaluate their players. If this helps spark ideas and dialogue, my main goal has been met.

Just for fun, because I’m a fan of baseball analytics, I predict that the Houston Astros will win the 2017 World Series because they’re one of the most data-driven teams in professional baseball, which will prove to be a differentiator in the long season. ☺

References


RICARDO VALERDI is an associate professor of systems and industrial engineering at the University of Arizona and a consultant to various Major League Baseball teams. Contact him at rvalerdi@arizona.edu.