Extra Innings

The perils of maximizing the use of scarce talent on software development projects

I once listened to a physician specializing in sports medicine commenting on the energy and stamina demanded of athletes participating in various team sports. According to the physician, though usually there is a consistent physical demand placed on most team players for most games, this does not hold true for outfielders on baseball teams – most of the game these players endure long periods standing in the outfield doing nothing, interrupted occasionally for a few thrilling seconds during which maximum effort might be required to chase and catch the ball, going from one extreme of virtual standstill to the other extreme of full exertion in only an instant. The intent of the commentary was to highlight the stresses placed on the body during athletic competition, but it also served to illustrate the relative activity by outfielders during each inning of baseball games.

Though software development may have little in common with sports medicine, I recently found myself contemplating the similarities it shares with baseball. A characteristic found with both is the concept of a team of skilled members working together to achieve a common goal, but also, as the outfielder analysis above portends, there are other subtle similarities.

Software development projects follow a schedule indicating milestones toward an expected completion date, just as baseball games are scheduled to be played for nine innings. When milestones are missed during software development, the expected completion date slips as more time is added to the schedule, just as in baseball when after nine innings a tie game continues into extra innings.

The Agile software development philosophy embraces the idea of a team where the members themselves decide the best way to utilize their respective skills to the maximum advantage of the project. Other software development philosophies predating Agile and still in common use today mainly rely on management to assemble a team of specialists with distinct responsibilities for each new project. These teams are composed of, among others, developers, testers, business analysts and educators, counterparts to the pitchers, catchers, infielders and outfielders of baseball, a game where specializing in a skill is more commonplace than with some other sports.

In software development projects where the contribution by team members reflects their high degree of specialization, it is not uncommon for some team members to experience periods of slack time while other team members are busily engaged according to their respective skills. In this context a software development team resembles a baseball team where the developer waiting for something to do is like the outfielder waiting for something to catch.

Software professionals working on large development projects command salaries commensurate with their talents, in some cases the required skills being in short supply. To mitigate the possibility of these well paid professionals finding themselves with slack periods on a project, management has devised sophisticated methods to share these resources across multiple projects with the expectation that project time can be scheduled to minimize slack periods. This seems to be a natural response to the dilemma of fully utilizing the time of highly skilled (and highly paid) people.

Let us explore how this idea of maximizing the utilization of team members might apply to baseball. Since outfielders often encounter long periods of inaction, perhaps we can devise a way to expose them to more action on the baseball field similar to the way software professionals are assigned concurrent projects.

Suppose we start with a set of baseball fields like you might find at municipal recreation park at any one of hundreds of small towns across the country. This field arrangement might appear as shown in figure 1, where two baseball fields are arranged in direct opposition such that the two catchers are looking toward each other as they play their respective positions at home plate. For convenient reference, each ball field is named for the

compass direction a ball would travel when it is pitched to the batter during play.

The proximity of the center fields of both baseball fields shown in figure 1 offers a tempting solution. Perhaps it is possible for one player to cover both center fields during two concurrent games. This would contribute to reducing the excessive amount of idle time for the center fielder in both games, providing one player more activity by responding to either game when there is action in center field.

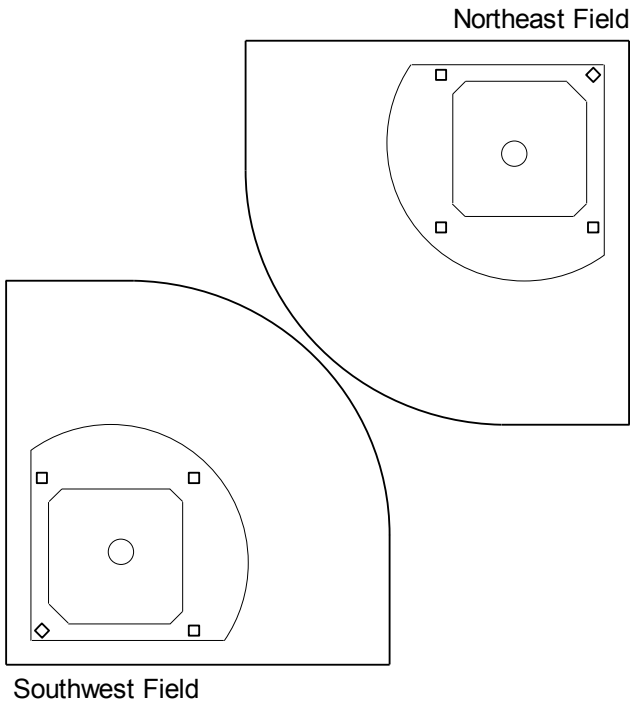

Figure 1

One problem with this arrangement is that the player would not know the best place to stand and be ready to respond to either game. We can address this problem by moving the two baseball fields closer together so that their center fields overlap, as shown in figure 2, with the area representing the "intersection of center fields" labeled accordingly with the acronym IOCF. Now the center fielder merely needs to occupy one position and can be ready to respond to action in either game simply by pivoting to face one or the other home plate. Having only one person play center field during two games addresses the problem where a very capable player could be mostly idle during the action.

This also introduces some other problems. The center fielder can respond only to one game at a time. There would need to be some way to prevent the possibility of both pitchers throwing their respective pitches at the same time and each batter hitting the ball to center field. A way to deal with this, other than to gamble that such a scenario would not occur and hope for the best, is to have the pitchers of each game alternate their pitching so that neither of them is throwing at the same time. The inevitable result with this approach is that each game takes longer to play.

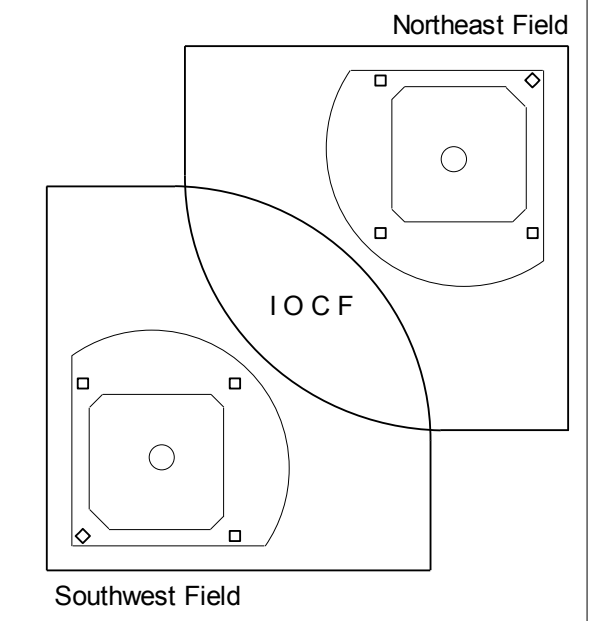

Figure 2

Optimizing the utilization of the center fielder has been achieved, but it comes at the expense of the progress of the games. By interlocking the fields, the pace of each game – its *flow* – is now inextricably tied to the pace of the other game. Neither game can proceed independently of the other, and throttling their pace to avoid the chance of simultaneous demand upon the shared center fielder extends the duration of both games. It is as though both games go into extra innings, putting an additional burden on players and spectators alike.

This phenomenon of maximizing the utilization of a subordinate entity at the expense of its superordinate entity is known as *suboptimization*. Mary and Tom Poppendieck provide an excellent analysis of the perils of suboptimization in the software development industry in their book Lean Software Development – An Agile Toolkit,[1] a book which also

1 Poppendieck, *Lean Software Development – An Agile Toolkit*, 2003, the chapter titled See the Whole.

addresses the topic of flow in software development projects.[2] In their book Leading Lean Software Development – Results Are Not the Point, the Poppendiecks provide a deeper analysis of flow[3] as well as a thorough examination of the potential for cascading delays arising from projects with interlocking schedules.[4]

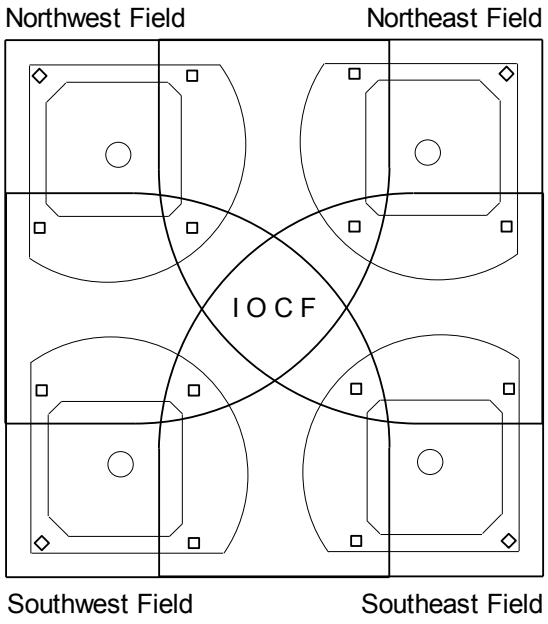

Figure 3

Let us take this baseball field metaphor to another level. Suppose there are four baseball fields arranged where their center fields intersect, as shown in figure 3. Now a single player positioned in the very center of this diagram can cover the center fields in four baseball games simultaneously. This also offers some other frightening possibilities for sharing players. Notice in figure 3 that the right fielder playing on the Southwest Field not only can cover the left field of the Southeast Field, but could just as easily cover either its short stop or third base positions, barely having to move to do so. The same problems we have when only two fields share an intersection at center field become exacerbated when four fields share such a common bond.

It is not uncommon for software professionals to find themselves assigned to two important projects concurrently, and more than a few have found themselves assigned to four at the same time. This usually is accompanied by an expectation by management that any urgent demands for their time and attention which might arise during the development process will be addressed immediately. In most instances there is no mechanism in place to handle the event where those multiple projects all require immediate attention at the same time by the same person, and pressure is brought to bear on the teams to do whatever is necessary to avoid slipping the schedule for any of the projects – extra innings is simply not a pleasant option.

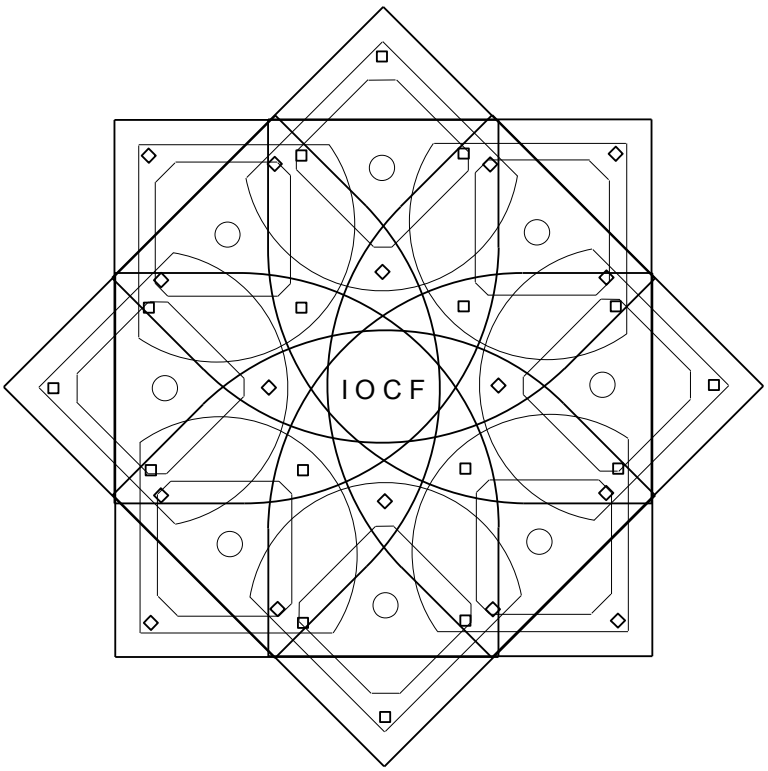

Figure 4

Taking this to another absurd level, consider a field arrangement where there are eight baseball fields all sharing center field, as depicted in figure 4. It boggles the mind just to conceive from this cluttered diagram all of the possibilities for player sharing, let alone managing the logistics required for implementation. Yet in the pursuit of reducing the idle time of individual team members it is not uncommon in software development organizations to have eight concurrent projects, each following a tight schedule, with nearly all assigned personnel contributing their respective efforts to more than one project.

2 Ibid, the chapter titled Deliver as Fast as Possible.
3 Poppendieck, *Leading Lean Software Development – Results Are Not the Point*, 2010, the chapter titled Reliable Delivery.
4 Ibid, p. 113.

There are some other observations we might make about the parallels between teams in software development and in baseball. Agile teams usually are expected to remain committed to one project until software is delivered to the customer. Indeed, the members commit to the team and the team usually stays together for a series of projects, eventually establishing what is known as a team *velocity*. This becomes a significant measurement by which to estimate the time it might require the team to undertake a project of comparable complexity. Similarly, in baseball a player commits to one team and the team stays together for the entire season. At the end of the season it becomes possible to determine, based on their respective win/loss records, which two teams are most likely to be playing each other during the final playoff game.

In a software development environment where people are assigned to multiple projects at the same time, even the concept of *team* is vague since it is impossible for anyone to commit to any one of the teams. With indiscriminate trading of teammates it becomes unlikely that any team velocity can be established by which to plan future projects for the team.

One way to address the concern of excessive idle time of personnel on projects arose in a retrospective of an Agile project at IBM, where it was recommended that the scope of a project also include "run at" features[5], those which the team will make an attempt at completing but are not guaranteed to be included in the delivered product. This allows the scope of a project to change rather than its schedule.

Another way is to assign software professionals work on multiple activities where only one of those is a project adhering to a schedule. The other activities may be lower priority customer objectives, which may become scheduled projects in the future, or internal work to improve the process by which software is developed. This removes the constraints imposed when a person is working on more than one concurrent project with scheduled milestones, easily facilitating a halt to any other work in progress when the primary project demands their instant attention. This may appear to be in violation of reducing the task-switching[6] blamed in part for degrading the quality of work, but if management continues to assign multiple tasks to team members, better that no two of those are projects following tight schedules, schedules which would become jeopardized if each one should simultaneously demand immediate action. As illustrated in our example, a shared center fielder cannot handle two balls hit to center field at the same time.

Perhaps in the future more software development organizations will migrate to methodologies such as Agile where optimization efforts are applied not to individual team members but to the team itself, allowing the game to flow at its own pace unencumbered by the possibility of unexpected delays which would require extending the schedule to accommodate extra innings.

Jim McDonough – February 5, 2011

5 Ibid, the chapter titled Aligned Leaders, p. 224.

6 Task-switching is one of the Seven Wastes of Software Development as described in Poppendieck, *Lean Software Development – An Agile Toolkit*, 2003, p. 4.