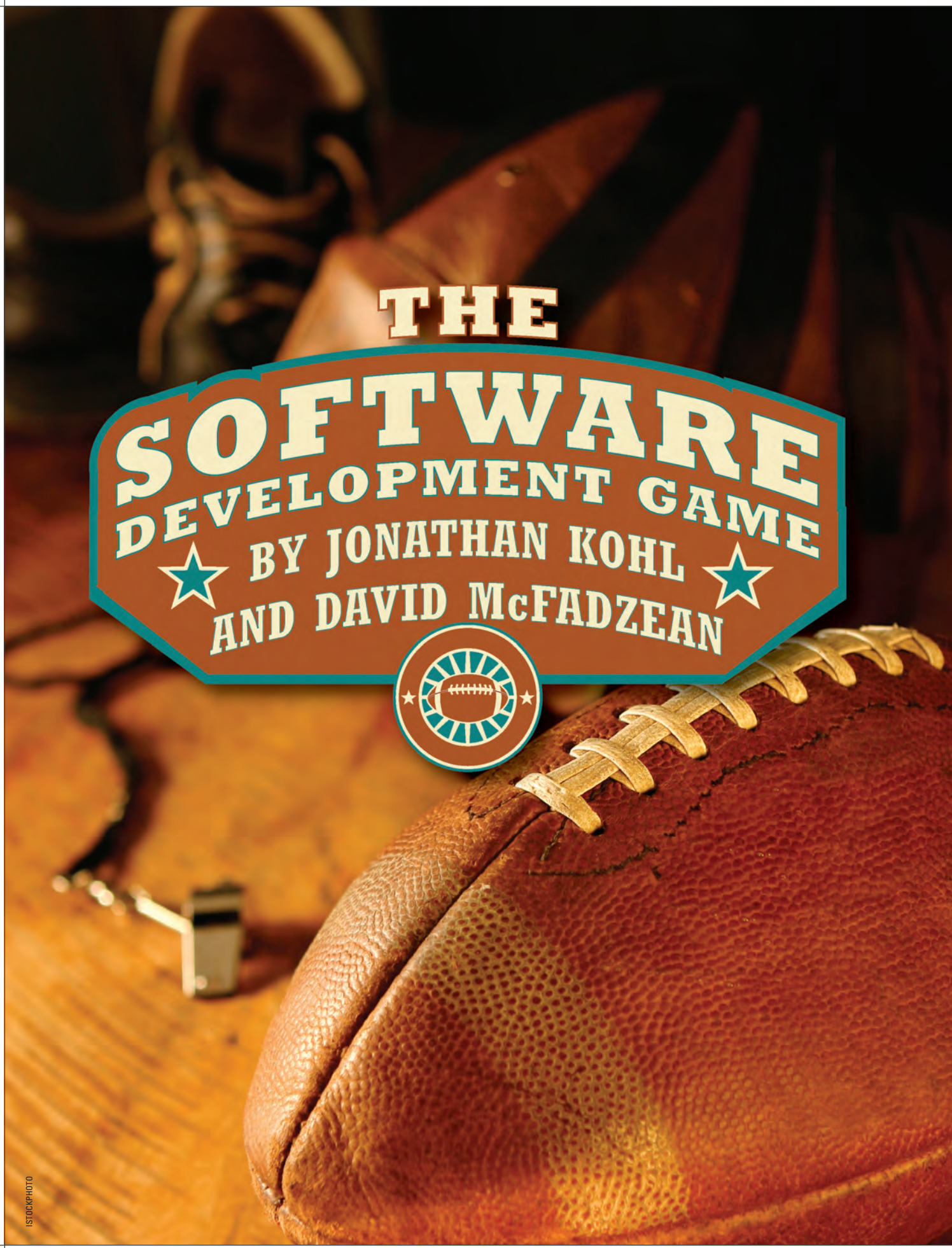

ISTOCKPHOTO

Many teams struggle to choose or adapt a software development process. We've developed a process strategy called the software development game (SDG) for managing the mix of process, tools, and technology on software development teams. SDG lets you pick a process—any process—and, using gaming concepts, helps you adapt it to your own needs.

How can serious software development be treated like a game? While you may play games for fun in your spare time, games are also serious business. Sports have professional leagues that support entire industries around their games. The military uses war games to test strategies and train soldiers. The SDG has been influenced by both game theory [1] (although we aren't using any formal mathematical modeling) and a more recent concept called *gamification* [2].

Game theory is a mathematical discipline used for modeling areas as diverse as economics, war, business, artificial intelligence, and biological evolution. At its core, game theory views every situation involving cooperation and conflict as a game. Some games have a defined time limit of play and a clear winner and loser, while others are experience based and ongoing—like a quest.

Recently, a movement called the *gamification of work* has become popular. Gamification involves imposing a game-like structure on certain aspects of professional situations to aid in productivity and motivation. Gamification can be as simple as offering rewards for completing certain tasks, or as complex as transforming an entire business practice into a game-like system. Because we can be so productive while performing repetitive tasks within social or gaming situations, researchers are trying to figure out how to tap into that potential to motivate within the workplace. (Gamification of work and game theory are not necessarily related, but there is an overlap. Understanding game theory can help gamification efforts, and gamification ideas can enhance game theory implementation.)

On software development teams, the team vision, purpose, rules of conduct, and informal practices are often created and enforced informally. This can result in confusion about the mission and purpose of the development team within the organization. At best, this informality leads to misunderstandings and communication breakdown; at worst, it results in a poor alignment to leadership's goals for the organization. Either way, both the team members and the organizations lose out when there is wasted effort that isn't contributing to value creation.

While formal game theory involves the use of mathematical models, analyzing gaming behavior is also effective. We have studied one aspect of game theory that looks at how people optimize their decision processes. In the SDG, we use game-like processes to help teams align with goals, provide clarity and coherence on issues, and offer visibility into the decision-making process. The SDG provides structure and accountability on a process that is frequently ad hoc, political, and unclear to team members. By gamifying decision making, the SDG helps software development teams determine and record their internal practices and their mix of technology, process, and tools. It can also serve as a framework to adapt existing policy and practices or to implement suggested changes for improvement after a team retrospective.

While both of us have been influenced by game theory concepts when leading software development efforts, it was David who decided to create a software development game framework based on the game *Nomic* by Peter Suber [3]. *Nomic* is a game about decision making where players agree on an initial rule set to govern game play, then they raise and vote on proposals to change the rules. So, changing the rules of the game is considered a valid move. *Nomic* is frequently played online, and games adapt over time as the players incorporate new ideas and changes. This is a great fit for dynamic software development teams that are frequently confronted with changing environments.

Rules of Play

To implement an SDG instance, a software development team starts with a minimal set of rules and an initial goal to create a *learning organization*—a group of people who continually enhance their capabilities to create what they want to create [4]. Where the game evolves from there is entirely up to the players (team members), but if it goes well, they become more productive and efficient and make better decisions as the game progresses. The SDG can start at any level—executive, management, teams, or individuals. Later, the game can expand to include more players and teams as it proves its usefulness.

David started as the facilitator. He created the game concept and educated team members on the process and the goals of the game. Once David had management buy in and the team agreed to try it out, he explained the initial rule set to govern game play and set up a meeting to see if all team members agreed to the rule set. A game page was created on the development team wiki describing the initial rule set.

Explanation of rules:

Rule 1: The initial goal of the game is to create a learning organization that enables the players to make high-quality choices and decisions. This rule should likely be refined to integrate the mission of the organization playing the game, as we specified above.

"If a proposal is vague, team members will offer up ideas and alternatives, and proposal clarification is a natural outcome. A proposal can become more concrete through discussion and debate."

Rule 2: All players must unanimously agree to all rule changes. The voting rule initially specifies unanimity to pass any proposal. Most games amend this early on to specify some sort of majority vote in order to avoid stalemates, but the initial rule errs on the side of caution so that the foundations can be laid out carefully.

Rule 3: Proposals may add, amend, or repeal a rule. This describes the initial set of "moves" that can be made in the game—introducing a new rule, changing an existing rule, or removing an existing rule. The game will usually evolve more sophisticated rules, such as giving certain classes of players the right to veto vote under some conditions; creating a category of immutable rules that cannot be amended (unless they are removed from that category); and introducing new types of acts such as resolutions, goals, standards, and guidelines.

Rule 4: All rules should be logically self-consistent. Ensuring that rules are logically self-consistent helps encourage fair play and motivates the players to keep the rule set sane. Whenever an inconsistency is introduced (accidentally or by design), the players will be motivated to resolve the inconsistency by amendment or repeal.

David then guided the team through initial game play. After agreeing on the initial rule set, the team set to work on solving a difficult issue: determining C++ coding standards for the team. Choosing coding standards can be one of the most contentious issues any development team can face. (Those of you who code for a living understand how difficult this can be; those of you who don't, imagine trying to find compromise between opposing political parties or religions.)

A proposal for a coding standard was put forward and voted in with a majority. After the vote and resolution, meeting details and the coding standard resolution were recorded on the development team wiki. By bringing the coding standards into the game, they now became rules of the game itself. By bringing software development policy and practices into the game, the team created a mechanism to follow and govern changes.

Evolving the Game

The SDG requires a framework for communication, raising issues, creating proposals to vote on, holding votes, and tallying results. David used a combination of a wiki, face-to-face meetings, email, and in-office instant messaging. In his role as facilitator, he answered questions, explained concepts, and watched for potential team issues that could be brought under the SDG.

For example, if a team member was complaining to colleagues about a lack of standards around builds, David would ask that person if the issue was important enough to be solved by the team. If it was, then he encouraged the team member to bring a proposal to the team so they could vote on it. A proposal could be as simple as: "Broken builds are a serious productivity issue. Some of us are spending hours trying to fix the build instead of completing tasks. We need to agree to fix the build problem and come up with ideas to address the problem." While that might seem like a simple proposition to pass because it's easy to agree to solve a problem, the hard part is actually doing something about it. If a proposal is vague, team members will offer up ideas and alternatives, and proposal clarification is a natural outcome. A proposal can become more concrete through discussion and debate. Ideally, the team will generate proposals with ownership and responsibility assigned to team members. From our prior example, a more specific proposal that would be actionable is: "Broken builds must be fixed before any new code is committed to the version control system."

Thinking up solutions for problems can take time and can cause a face-to-face meeting to drag out. Furthermore, some personality types think better outside of a group and may approach team members after a face-to-face meeting.

The team agreed to use technology to make the process more efficient—proposals and votes on them could be initiated and executed electronically. If a proposal required more information than could be conveyed in email or was of a serious nature, the facilitator could initiate a face-to-face meeting to hear the proposal and hold a vote.

Now, imagine that you are the DevOps team member who has come up with a proposal to fix the build problem. You're the team member who feels the broken build pain the most, and your potential solution works well. You've tested it out and your findings are positive. You explain your proposal to adopt a solution within the SDG, but you fail to get a majority vote. You are disappointed, and no other alternatives received a majority vote. You *know* this is the right way to go, so what do you do? If you want the vote, you will need to do what people in politics do and lobby for support.

- Educate team members on the merits of your proposal.
- Try to get key, influential people on your side to vote for the proposal.
- Appeal to the skeptics: *How about a proposal to identify measurable outcomes and do periodic checks on the system to see if it is solving problems or not?*
- Make a formal proposal and vote.

- Hope your lobbying efforts pay off and the proposal passes.

Once team members are comfortable with the process, it doesn't take long for them to realize that any proposal can be brought forward—even the most self-serving ones. If there is team consensus to implement a change, the motivation behind it doesn't matter. It might be as simple as one team member becoming bored with the current technology and wanting to move to something new. It might seem selfish to say, "I don't want to work on Java web apps that much anymore. I'd love to work on mobile projects." But if it is brought up in a forum, you'd be surprised how many others on the team feel the same way, including managers and product managers. Management may feel the organization needs to move to new technology to not fall behind, and product managers may be researching what competitors are doing, but neither group wants to bother the busy development team about it right now.

Without a forum to raise an issue openly and honestly, this kind of idea goes underground. In the worst case, it festers as a frustrated team member complains to others or attempts to use subversive or manipulative methods to try out a new technology platform. Once the right stakeholders are informed and they buy in to a proposal, it can be a powerful technique to introduce change, even with self-serving motivations.

Once David's team had proposed and voted on a number of resolutions, the rule set expanded. This required categorization. Two potential categories are *rules that govern the game itself*, and *rules that govern software development activities*. In addition to the initial SDG rules, rules were added to govern rule changes, proposals (create or withdraw proposals), voting rules (what constitutes majority), and multivotes (tie breakers, etc.). For the software development activities, rules were grouped according to team policies (vision statement, processes to follow) and development standards (coding standards, code reviews, and build and testing activities). As the rule set expanded, roles were added so that team players could have ownership in certain areas of the game based on their expertise and interest level. For example, roles can involve facilitating game play itself, overseeing technical components of the software development system, and guiding product direction. Roles were expanded to include managers and other stakeholders when their participation was needed.

The SDG evolved further to include gamification aspects for repeated tasks. Achievements for repeated tasks that might not be that pleasant were added as quests in the game. For example, business travel can be difficult and tiring, so the team decided to reward the top travelers on the team by giving them a shout out on the team wiki. There also were humorous booby prizes awarded to the last person who set off the building alarm or to the person who broke the build the most frequently.

This particular SDG instance has evolved to incorporate more and more of the daily life of the development team, while providing structure around communicating issues and making decisions on how to move forward.

Why It Works

This isn't a one-team, one-time success story. David has implemented several SDG instances on different teams at different companies over the past few years. We have found that making the problem-solving and decision-making processes visible helps improve communication and reduces confusion. Much misunderstanding on development teams stems from differing expectations about what the team or individuals should accomplish and a lack of alignment toward organizational goals. Since decisions are democratic—anyone can table an issue, the team votes on all changes, and decisions are binding—team members feel included and valued as integral parts of the process. The SDG provides a framework for raising concerns and changing existing practices and tools in a way that helps teams cope with the changes in their external environment by adapting their internal practices as needed. Furthermore, if the team finds that the game framework itself isn't working for them anymore, they change the rules to improve it. Using game-like concepts in the workplace is a way to harness the natural behavioral dynamics that occur within groups. Since the game itself can be adapted, teams don't find themselves stuck with a rigid process that isn't appropriate for their new circumstances. Rules can be amended or even repealed if they no longer add value.

Management and other leaders might be nervous about the SDG at first. It should be clear for both management and team members that the game only applies to areas over which the development team has ownership. The team shouldn't

Implementing Your Own Software Development Game

1. Start off with simple game play rules (feel free to use our example).
2. Use a facilitator to guide game play, manage meetings, tally scores, and record and update rules.
3. Start simple, and let the game evolve. Don't try to do too much.
   - Develop team policy and alignment to organizational goals.
   - Consider using the game to help implement retrospective ideas.
4. Use the game to discover what your existing processes are, record and ratify them, and make them visible to all team members.
5. Don't let the rules become unwieldy:
   - Try to keep rules brief and lightweight.
   - If rules are too numerous, work on scaling them back.
6. As the game expands, introduce additional roles to help with administration.

contradict existing corporate policies or try to overturn decisions made by leadership. For example, team members can't just go and vote themselves raises and bonuses or decide on their own to scrap the existing product line. For areas that are governed by other stakeholders, the team can bring issues to their attention, but the existing organizational structure and policies should remain intact. (If leaders want to add the game to other areas, that is fine, but don't try to use the game to undermine them.) Leaders will find that the game can create clarity and coherence of their vision of the company and their product and service mix. Team alignment on actions and goals may increase, and the transparency on decisions means management can review when and why certain technical directions were taken when proposals were voted in.

An SDG helps teams make decisions, particularly if the teams are self-organizing. It also helps build team cohesion and encourages diversity of opinion and healthy dissent. If there are serious problems, an SDG can provide a framework to help a team change course on projects and tasks to reach organizational goals.

A fabulous place to start using an SDG is to help implement changes after a retrospective. How many times do we have a great meeting after a release, outlining problems we encountered and possible solutions, only to forget about them until the next retrospective? In the meantime, we didn't do anything; we were too busy working on tasks. We had great intentions, but without a system to help us decide on courses of action and to measure progress, we forgot about our solution ideas. With an SDG, retrospective ideas can be implemented through the game, rather than forgotten until next time.

Conclusion

Software development processes can be difficult concepts to apply broadly. What worked for one team in its unique context may not work for your team. Adaptation is important in cases when a team tries out a process and finds that some practices don't work or that key components are completely absent. When processes fail, a convenient response is "You need to do what works for you and your team." That makes sense, but what specific, concrete practices do you use to find out what process works for you? We've had good success figuring that out for our teams by using the software development game. **{end}**

jonathan@kohl.ca
davidmc@gmail.com

Sticky Notes

For more on the following topics go to www.StickyMinds.com/bettersoftware.

- References
- Further reading