Groundswell: Two Patterns of Technical Leadership

Dirk Riehle, dirk@riehle.org, www.riehle.org.

Summary

The two patterns of this article discuss an experience of technical leadership that I have had and seen many times over: how to help change the behavior of developers such that all developers (of a team or of a development organization) follow a common practice and thereby make the team more effective. Examples are adhering to common programming guidelines, supporting a consistent testing process, and using the same programming environment. The pattern context of application are small to medium-size development organizations that do not have an established standard-setting process but tend to solve problems in an ad-hoc way.

1 Introduction

In 1999, I joined a development organization of highly skilled software developers. These were about 30 developers, split into 3 teams. The company, a technology startup, was already 3 years old. However, for a variety of reasons,

- it had no common programming guidelines,
- it was using widely differing programming environments,
- and it had no consistent component (unit) testing process.

It was not that members of the development organization didn't understand the value of following shared practices. They just couldn't agree how the practices should look like, for example, what the programming guidelines should be or which programming environment was best suited for their tasks.

Two years later, we have changed all this. We now use the same programming guidelines and the same programming environment, and we have a significantly improved development process in place that includes a consistent approach to unit testing. I initiated and lead much of this process. The patterns presented in this article discuss my experiences with these and other change processes in a pattern-based form.

2 Overall Context

Most approaches to organizational change focus on mature organizations that have the resources to appoint dedicated standards committees or process improvement teams [1, 4]. Patterns like *high-level management support* and *sponsored conferences to educate developers* figure prominently in helping these teams achieve their objectives. This approach works well in established organizations that support top-down standards setting.

Sometimes, this approach does not work well: the need for a specific kind of change may not be apparent, or there are no initial resources to support it, or it is simply not clear what the desired result of change should be. Such situations of uncertainty are frequently found in young companies like technology startups or highly innovative departments of established companies like research labs.

Startups and similar environments are frequently used to solving problems in ad-hoc ways. They need to be innovative to survive and flourish. Operational efficiency is typically (and rightfully) not the first thing on their mind. However, the larger a startup grows, the more important it becomes that it streamlines its core operational processes.

Streamlining processes is typically at odds with the predominant culture of ad-hoc solutions. People easily settle into "doing it their way" if no overriding consensus dictates otherwise, and it takes a conscious effort to change this.

The patterns in this article show how to achieve such change, even if no or only little initial management support is provided. The patterns can help you achieve organizational change of a small to medium size. The article discusses two patterns, addressing different but related situations and audiences:

- *Force-vector alignment.* How to achieve consensus on a team-level.
- *Team-level groundswell.* How to achieve consensus in a small to medium-size development organization.

Team-level groundswell builds on force-vector alignment. Both patterns are driven by the same underlying principle of *piecemeal growth* [5, 6]. Both patterns find their limit in where they go beyond what developer-to-developer communication can achieve, for example, when multiple locations are involved or the development organization is too big.

3 Force-vector Alignment

3.1 Problem

How do you achieve team consensus in software development if everyone is used to doing it his or her own way?

3.2 Context

You are an established member of a team that carries out a certain development practice only inconsistently or not at all. For example, you all follow different programming guidelines. The different ways of carrying out the practice are well entrenched with the team members.

Everyone recognizes the advantages of consistently following the same practice. At times, people have tried to define a common practice, but none of these discussions has led to an agreed-upon result. Rather, the discussions have led to frustration and the conviction that agreement is impossible.

Some of the team members are indifferent and willing to accept any solution. Other team members are strong-willed, believing that they do it the right way and everyone else should do it their way.

Every person on the team has a history that made his current way of doing things best for him. Certainly, they are not "bozos" (see the *Don't flip the Bozo bit* pattern [7]), but highly qualified individual contributors.

3.3 Forces

People don't like to change unless good reason is apparent. They may feel threatened by change and resist it.

Failed previous attempts have led to confrontational situations that make discussing the practice hard.

People on the team may believe that a solution that is not their solution means loosing face.

Management does not impose a solution, because it fears losing developers or because it does not care enough about the practice to enforce a solution. (As is apparent from recent economic developments, the fear of loosing developers strongly depends on the state of the job market.)

You strongly believe that a shared practice is necessary and commit yourself to establish it. However, you don't have the formal authority to command adhering to a standard practice (nor do you believe it would do any good).

3.4 Solution

You first evangelize a minimal solution, from which you grow an accepted version of the practice. The process comprises four distinctively different parts, each of which increases the number of people involved:

- selection of a starting point,
- initial growth in circle of close co-workers,
- expansion to large parts of team, and
- team-wide acceptance by majority vote.

This evolutionary approach let's you find the right solution over time and avoids that you get stuck in the beginning. It also accommodates feedback from your colleagues that needs to be incorporated to make the solution a shared practice rather than an imposed one.

During this process, you play two roles:

- evangelist for the definition of a shared practice,
- maintainer of the shared practice's definition.

In the beginning, you need a starting point. For programming guidelines, for example, the manufacturer's recommendations may suffice. For a programming environment, the market leader's product may suffice.

Taking on the evangelist role, you start with one-to-one marketing. Weighing the benefits of your initial solution, you turn to your closest co-worker, and discuss the benefits of starting to share this practice. The result is a shared solution that is supported by both of you. Extend this to your closest circle of co-workers. Work hard to keep the solution as small as possible.

What you are doing is to get your co-workers on board, one by one, and to grow a solution that is shared by all. At this time, the solution may actually change drastically, as a result of the close work you and your co-workers are carrying out.

It is instrumental that you start small and work incrementally to get a majority support for a solution that started out as your solution but with its widening acceptance becomes the team's solution. Any big discussion, in which everyone has his say, is likely to fail. Picture it this way: if every person is like a force vector, it is highly likely that the sum of all force vectors in a big discussion adds up to zero. However, if you take it on one by one, you align forces to make sure the total force vector grows in one direction.

If you get it right, your initial efforts with your closest co-workers will pay off in a snowball effect. Convinced about a solution to which they contributed, your co-workers will start evangelizing it themselves.

You now have to take on the role of the maintainer of the shared practice's definition. Your main task becomes ensuring consistency. This sounds harder than it is. Once a critical momentum is reached, it is unlikely that new followers of the practice will request any significant change. If they don't agree with the practice, they will simply decide not to follow it (for the time being).

Eventually, however, even if individual people do not join you in following a practice consistently, your work will gain so much weight and acceptance with a large number of people that there is really no good reason for not making it a standard that everyone has to adhere to. At this point of the process it is unlikely that there is any remaining technical argument that hinders it, and your manager will feel confident to impose a practice that already has a large following among team members.

Thus, you need to convince management to impose the practice, which is typically an easy thing to do since most managers prefer consistent and repeatable ways of doing things to individual artistry.

While not always possible, at some point in time, you should slip back into the background. For one, this saves you energy. However, it also takes you out of the spotlight as the person who initiated most of the changes, freeing you up for further tasks, in particular when it comes to going beyond the boundaries of your team.

3.5 Consequences

If not carried out well, you might end up creating bad blood and individual people might resent your efforts. How to avoid such problems needs to be addressed by further patterns. In my experience, these patterns would be about how to interact effectively with people; they would not be about the overall consensus-building process. Therefore I see these patterns in support of force-vector alignment rather than as a part of it.

4 Team-level Groundswell

4.1 Problem

How do you establish shared practices among different teams in a software development organization?

4.2 Context

Your team is following a shared practice, but other teams aren't. They are still in the state of anarchy that your group was in before it established the shared practice. You used *force-vector alignment*, and many colleagues are eager to support you in getting the practice more widely established.

While communication paths between your team and the other teams are less dense than between the members of your team, following a shared practice across all teams still represents a major gain of effectiveness and efficiency.

Your solution is (most likely) a more mature solution than anything used by any of the other teams. However, because constraints may vary from team to team, your practice may still need to be adapted for the other teams.

Management may see the benefits of carrying out the shared practice, but does not want to impose it. There may be different reasons: the team managers do not consider the practice important enough, or they may not be on good enough terms with each other to take over solutions, or the way the practice is defined may not fit all purposes.

4.3 Forces

Force-vector alignment does not work, because it does not scale beyond the limits of a single team: communication is not dense enough between teams if there is not continued shared work.

Additional stakeholders complicate the picture, including the managers of the other teams and a possible Quality Assurance (QA) team, all of which want to have their say in a solution.

Your team's bottom-up approaches to defining its own practices may be met with distrust by the other teams, fearing that you might try to expand beyond you own boundaries.

4.4 Solution

You need to take a two-pronged approach to help introduce a practice shared across all teams:

- you work individually with other teams, establishing beachheads of support, and
- you work with management/stakeholders to provide global support for the practice.

You have a lot going for your solution: you have a well-versed solution that has been tried many times before by your own team. The team understands the solution and doesn't need to be convinced to praise the benefits of its practice to the other teams. You and your team members can convince co-workers in other teams to try your practice and adapt it for their needs.

However, the attitude changes. Members of other teams are less likely to jump onto the bandwagon, just because they believe that following a shared practice is important. More likely will they meet you with a consumer attitude: they may agree on the significance of a shared practice, but they also expect to be served a well-prepared meal.

Here, of particular help is technical or organizational support for the practice. For example, when you are evangelizing shared programming guidelines, you might already have a pre-configured pretty printer (code beautifier) at hand, or, if you are pushing a shared programming environment, you can provide additional tools that specifically support the suggested product and make life easier for developers.

In all likelihood, however, this is not going to be enough. Your team's reach into the other teams is too weak to ensure that the adapted practices remain sufficiently close to your original definition, if they are used at all. If you are unfortunate, you end up with inconsistent practices and no consensus across the teams.

You therefore need to push the shared practice top-down as well. Your best friends are top-level management and, if you have such groups and get them to cooperate, Quality Assurance or Process Development. These parties have a vested interested in making people adhere to common practices. The history and high-quality definition of your practices should help you convince the other stakeholders. Top-level management may just want to delegate, while Quality Assurance or Process Development may want to have its say in the definition of the practice. This is fine: after all, their perspectives are unique and important, and no practice is cast in stone.

The combination of beachheads into the other teams with high-level management and institutional support should ensure that the other team managers take a serious look at the suggested shared practice and accept it for their team.

Once more, you or institutional support need to serve as the maintainer of one coherent definition of the practice. It works somewhat different, though, from force-vector alignment. You typically expect feedback from teams now, rather than from individual contributors. Again, you will have to strike a delicate balance between accommodating the different teams' needs and a solution that is as lightweight as possible.

You need to be prepared to support the other teams in their transition phase. However, by then you already built up support inside your own team for the practice that it should be easy to provide it to the other teams or even transfer it to Quality Assurance to take care of it.

5 Conclusions

Both patterns have worked for me on a variety of issues, including:

- introducing common programming guidelines,
- agreeing on a common software development environment,
- introducing consistent unit testing to the development process,
- setting up consistent rules for using configuration management,
- introducing new programming concepts, and
- improving developer education through voluntary reading groups.

Along the way of applying these patterns, I have used many other techniques. For example, to emphasize the value of unit testing using JUnit [9], a large part of development sat in on a web seminar by the Extreme Programming methodologists. They did so, because developers recognized these methodologists as authorities to listen to. This tech-

nique has been called *Big Jolt* [4]. You use outside experts to lend more authority to what you are saying. This may be a somewhat frustrating experience, but enlisting outside experts is certainly an effective technique.

Another interesting observation is a similarity of the processes outlined here with the technology adoption process described by Moore [8]. Moore discusses the lifecycle of a new technology and how customers accept it. It consists of four distinct phases:

- an early adopter's phase, where only technology visionaries adopt the technology,
- the bowling alley, where you gain one customer after another by a good-enough niche product,
- the tornado, in which everyone jumps onto the product because it promises obvious benefits, and
- main street, in which you sell the product to established customers.

Force-vector alignment is clearly is an early adopter's pattern. You gain support by "selling" your practice to close co-workers who have the visionary capabilities to see its benefits. Team-level groundswell falls into the bowling alley and tornado category, where you win over one team after another until you hit it big and everyone jumps on it.

On a more fundamental level, I believe this comparison points towards a crucial trait you should have when embarking on organizational change processes like those outlined in this article: you need the heart and spirit of an entrepreneur to cope with the trials, setbacks, and uncertainty that marks your way. If so, you will be rewarded manifold.

Acknowledgements

I would like to thank David Kane for helping me improve the article in preparation for PLoP 2001.

References

[1] Brad Appleton. Patterns for Conducting Process Improvement. PLoP 1997 Conference Proceedings, Washington University Technical Report #WUCS-97-34.

[2] James O. Coplien, "A Generative Development Process Pattern Language", Pattern Languages of Program Design, James O. Coplien, Douglas C. Schmidt (Ed.), Addison-Wesley, 1995, pp. 178-237.

[3] Neil Harrison, "Organizational Patterns for Teams", Pattern Languages of Program Design 2, John M. Vlissides, James O. Coplien, Norman L. Kerth (Ed.), Addison-Wesley, 1996, pp. 345-352.

[4] David Delano, Linda Rising, "Introducing Technology into the Workplace", PLoP 1997 Conference Proceedings, Washington University Technical Report #WUCS-97-34.

[5] Richard P. Gabriel. Patterns of Software. Oxford University Press, 1996.

[6] Brian Foote and Joseph Yoder. "Big Ball of Mud," Pattern Languages of Program Design 4, Neil Harrison, Brian Foot, Hans Rohnert (Ed.), Addison-Wesley, 2000.

[7] Jim McCarthy. Dynamics of Software Development. Microsoft Press, 1995.

[8] Geoffrey Moore. Inside the Tornado. Harper Perennial, 1999.

[9] Kent Beck and Erich Gamma. The JUnit Framework. www.junit.org.