Enough Software Processes, Let's Do Patterns!

Paul E. McMahon, PEM Systems

Abstract. This paper explains common mistakes many organizations have made in the past when implementing defined processes with a goal of improving software practitioner performance. The paper suggests a more effective approach to improvement is to focus on patterns where software practitioners need help in making better decisions. A practical approach to develop patterns is described along with multiple pattern examples, including examples that can help software practitioner supporting roles.

Introduction

What is software development? Is it a craft, or an engineering discipline? Regardless of how you view it, there are over 11 million practitioners developing software today [1]. Therefore a more practical question is:

How can we do better at helping software practitioners with the common challenges they face each day?

One approach that has helped is reusing design patterns [2]. However, software development extends far beyond just design and programming. In fact one organization reported that their typical software developer's workday breaks down as follows [3]:

- Design– 0.5 hour
- Programming– 1.0 hour
- Requirements gathering, analysis– 0.5 hour
- System testing, fixing bugs– 2.0 hours
- Deployment support– 0.5 hour
- Project management, walk-throughs– 0.5 hour
- Admin tasks (e.g. reading email, meetings etc.)– 3.0 hours

This means that for every 1.5 hours of design and programming, there is another 3.5 hours of work for other common software development activities. These numbers are not unusual, and some might argue even with this breakdown many organizations are not spending enough time on requirements, and testing.

This raises another question: Are there other patterns beyond design and programming that could help software practitioners with the other challenges they face each day?

The idea of patterns beyond design and programming is not completely new. Scott Ambler has written extensively about the potential value of process patterns [4]. However, to date, patterns have not received anywhere near the attention they deserve given their potential value to improve both personal and organizational performance.

Motivation for Patterns Beyond Design and Programming

The exact amount of time a software developer spends beyond design and programming varies across different organizations. However, even in organizations where requirements gathering is handled by business analysts and customer interactions are handled by marketing, software developers still spend significant time clarifying ambiguous requirements and interacting with stakeholders.

Furthermore, with today's emphasis on self-directed teams, estimating the work, reporting status, and identifying and working improvements to tools and practices all fall within today's typical software developer's task list. This leads to another question: Is it possible that helping software professionals with pattern recognition beyond design and programming holds great untapped potential for high value personal and organizational performance improvement?

When you consider how much time software practitioners are spending beyond design and programming this potential is difficult to ignore.

What Do We Mean By a Pattern?

To understand the value of patterns let's start with an analogy. We've known for a long time that the best way to build a complex software system is to break the problem down into small chunks and build and deploy those small chunks incrementally in small slices. This approach works because it gets product into the hands of customers sooner where those same customers can provide rapid feedback ensuring the software development team stays on the right track.

So if it makes sense to build and deploy our software systems in small slices to keep developers on track, why wouldn't it make sense to build and deploy the practices we want our developers to follow in small slices to ensure we are giving them the help they need to stay on track?

When I refer to a "practice slice" I mean a common scenario or a small set of related scenarios that a software practitioner typically faces each day.

When I refer to a "pattern" I mean the essentials of a practice slice with less important details left out, and key related questions, tips, and warnings added in to help practitioners make better decisions when facing similar situations [5].

More Motivation for Patterns

What do many organizations do when a new software practitioner comes into the company? They want the new practitioner to learn the "way-we-do-software". So they give them the volumes of their "process documentation," and they tell them to study it because it contains "how-we-expect-you-to-operate". While some of this is necessary, there is another side to consider.

What if you viewed your software practitioners as your customer for your organizational processes?

By looking at it this way, rather than thinking of your company processes as just direction to your practitioners, they also become a vehicle to help you listen and respond better to your

practitioner's needs. When organizations focus on patterns they listen better to the common situations that their practitioners are struggling with. This can generate valuable discussions with experienced practitioners helping less experienced practitioners learn faster how to handle these difficult situations. This can lead to high value and rapid performance improvement.

I think most would agree that our processes should help practitioners perform better. However, where people tend to disagree is on the best path to get to this goal. Now let's discuss more about past mistakes.

Process Deployment in the Past

Practitioners face all kinds of different situations each day and so our processes can't possibly tell practitioners what to do in each situation.

One mistake many organizations have made in the past is trying to define in detail what practitioners should do in every situation. This has led in some organizations to processes that are so heavyweight that they are not usable by human beings. Other organizations have gone to the other extreme making their processes so light as to become trivial and of minimal, or no, value. What practitioners really need falls between these two extremes.

Where Software Practitioners Need Help the Most

There are two types of information most software practitioners need. First is basic information about a process, such as how to conduct it, competency needs, checklists to prepare, and completion criteria. This kind of information is useful primarily for beginners. Traditional process definitions have done a reasonably good job at meeting this need.

The second type is the information that practitioners need while they are executing the process. This includes things to watch out for such as common situations where they may need to make a decision. This may include trade-offs between options they have. This is where most practitioners—once they move beyond the beginner's stage— need help the most, and this is where patterns can help the most.

Implementing the Most Valuable Patterns to Aid Performance

When developing patterns to aid decision-making what is most important is to NOT pick just any common scenario. For many common situations that happen everyday practitioners don't need extra help. Therefore you want to pick the situations where your practitioners often fail to make good decisions. This increases the value of the pattern as it helps the practitioner where they need help the most during a typical day.

I have developed patterns with a number of clients. First we examined the common scenarios their practitioners were facing each day. Then we prioritized the scenarios and selected a small set where there were tendencies toward poor decisions that hurt performance. [5]

We then added in key questions practitioners should be asking, tips and warnings based on further discussions with experienced practitioners. Then we trained the people in the patterns, and we gave them simple aids they could take back to use as on-the-job reminders.

Patterns are a simple and practical way to communicate to less experienced practitioners how the more experienced practitioners in your organization would handle the common difficult situations they are likely to face.

Pattern Examples

Following are three simple pattern examples, specifically for software developers, created by expressing scenario essentials in a user story form [6], and adding in related questions practitioners should be asking, tips based on the answers to the questions, and warnings.

Pattern One: "I don't Understand a Requirement"

User Story: "As a software developer, I want more guidance in what I should do when I don't understand a requirement, so I can build the best software in the least amount of time to meet my customer's needs and my commitments."

Questions practitioners should be asking:

- Is my customer collaborative, and working closely with me to discover the requirements?
- Do I have a fixed schedule and cost?

Tips:

- Do nothing different if your customer understands the cost of iterating on requirements and is working closely with you to discover the requirements.
- Raise a risk if your customer is non-collaborative and you have a fixed cost and schedule.

Pattern Two: "My Testing and Peer Reviews are Taking Too Long"

User Story: "As a software developer, I want to know what to do when my testing and peer reviews are taking too long, and I am getting pressure from my manager to finish on time."

Questions practitioners should be asking:

- Does my software have life-critical consequences if it fails?
- Does my project have an agreed way of working with respect to testing? If so, have I reviewed my testing process to see what options I have?
- Does my peer review process provide options for focused, streamlined peer reviews based on criteria? If so, have I considered those options?

Tips:

- Consider focusing your low level testing just on areas you have changed, if allowed by your agreed way of working.
- Consider focusing tests and peer reviews on high risk areas, if allowed by your agreed way of working.

Warning:

- Be sure to assess any risk involved in reduced testing or streamlining peer reviews, such as:
- Missing key dependencies.
- Not following the agreed way of working.

Pattern Three: "How Should I Handle a Design Risk?"

User Story: "As a software developer, I want more help in how to handle a design risk, when the alternative designs are going to extend the schedule and I am getting pressure to finish on time."

Questions practitioners should be asking:

- Have I discussed my design with a more experienced coworker or colleague who has implemented a similar design?

Tips:

- Call for a peer review.
- Call for a limited peer review with key people focusing just on the design risk, if allowed by your agreed way of working.

Some of these questions and tips may seem obvious, but keep in mind that practitioners tend to forget the obvious under project pressures. One of the key values that patterns provide over simpler checklists is they give your practitioners context and remind them of choices they have, but might have forgotten.

It is critical for practitioners to think about their own context when making on-the-job decisions and patterns can aid practitioner critical-thinking in ways that traditional checklists and process definitions sometimes fall short. When developing your own patterns be sure to conduct discussions with your experienced practitioners so the questions, tips and warnings are relevant to your own organization's context, constraints, and culture.

More Sample Scenarios and Guidance To Kick-Start Your Own Pattern Development

Below are three more sample scenarios with related questions, tips, warnings, and some additional guidance that can help kick-start your own pattern development. The scenarios provided are all based on real experiences I have observed with actual clients and I have found that these types of scenarios tend to repeat in many organizations[1] [5].

The patterns in this section are not limited to software developers. I have included patterns that may be of interest to software practitioner supporting roles such as coaches, software testers, team leaders, Scrum Masters, project managers, and process improvement professionals. I have included these additional patterns to demonstrate how patterns can be used to aid the performance of software supporting roles as well software developers.

The scenarios in this section include additional context information beyond the simpler user story form provided previously. This additional information may be beneficial for practitioners when first learning the scenario in training, whereas the simpler user story form may be sufficient as a quick on-the-job reminder. The related questions have been developed based on my own consulting experiences, and the use of the Essence framework [7, 8, 9, 10, 11].

Pattern Four: "Getting Your Coaches on the Same Page"

Coaches can help a team, but multiple coaches helping the same team need to coach consistently. This is particularly important when teams are transitioning to a new agreed way of working as conflicting advice can do more harm than good. This scenario should be of interest to coaches, Scrum Masters, team leaders, developers, testers and process improvement professionals.

Scenario start:

I was coaching a team that was moving from a traditional waterfall development approach to an agile-Scrum approach. This company had a history of being run by PMI certified project managers requiring detailed work breakdown structures up front with detailed tasks assigned and tracked by all project managers.

When I first explained to the practitioners in the training class the concept of self-directed teams, everyone liked the approach and seemed to understand it. But after the training the team had trouble following the new process. After the first few daily stand-up meetings [12] the team members would go back to their workstations and sit waiting for someone to tell them what to do next. This was because the culture in this organization was for people to just do what they were told.

Another reason the practitioners had trouble following the new practices was because some of the new practices seemed to conflict with the existing organizational policies, as well as the existing culture. For example, testers in this organization were not allowed to develop code.

The project manager, who was learning to be a Scrum Master, expressed concern about the situation and wanted to know if she should re-institute her two hour weekly meeting where she had traditionally assigned tasks to individuals and driven actions to closure. I replied:

"No, this is not your role now. But you do need to remind the team members of the new expectations whenever you observe a problem, and you and I need to talk more to make sure we are coaching consistently and in a way that fits within your company constraints."

This was a style change for the project manager, and it required some time for the project manager and myself to work through a number of issues related to constraints on the team's operation.

This team had competent team members committed to start the mission, but they weren't yet collaborating given the new agile-Scrum practices.

Once we worked through these issues and communicated practice clarifications to the team, it didn't take long before the team was collaborating in a self-directed way.

Questions Teams Leaders and Process Improvement Professionals Should Be Asking:

- If you are using multiple coaches, such as an external coach and an internal coach, have you worked through any issues that might exist with your agreed to way of working (e.g. organizational policy constraints) to ensure consistent coaching?

Tip to Team Leaders:

- Consider using a head-coach at least at the start of the endeavor to coach the coaches ensuring key points— especially those associated with likely transition trouble spots—are understood and coached consistently.

Examples of typical areas where issues might arise when transitioning to an agile-Scrum approach include [13]:

- Constraints on who can sign up for certain task types
- Capacity planning expectations
- Risk assessment expectations
- Expectations on basis of task estimates
- Expectations on use of team velocity
- Expectations on use of burn-down charts

It is not uncommon in organizations moving from a traditional command and control approach to a self-directed team approach to need reminders and clarifications on basic expectations during the initial transition. This is a culture change in many organizations and the degree of agility that makes sense in one organization often differs substantially from another due to project specific conditions. Therefore it is critical for each team and their coaches to agree on their way of working and then coach consistently to those agreements.

Warning:

Keep an eye out for coaches who just coach to their own experiences and aren't attune to the needs of your organization.

Pattern Five: The "Non-Engaged Stakeholder"

In Pattern One we saw an example where a stakeholder does not know their own requirements. In this scenario we examine a different type of problem with a stakeholder. This is the situation when your stakeholder knows the requirements, but isn't providing the information you need in a timely fashion. This scenario should be of interest to all software practitioners, project managers, and subcontract managers.

Scenario start:

On the LEM project Don's company was subcontracting component testing. Critical test data was required to be supplied by a customer agency, but the agency kept putting off Don's repeated requests for the data. Don was the project manager and this situation put his subcontractor over budget and behind schedule. Eventually the data arrived and Don approved additional budget for his subcontractor to complete the testing. However, since this project was at a fixed cost to the prime contractor, it resulted in significant lost profit for Don's company.

Tip:

When you face this kind of dilemma consider first digging to see if you can uncover the root cause.

Questions the project manager and subcontract manager should be asking:

- Has a stakeholder representative for the customer agency been appointed and has he agreed to take on the responsibility to provide the required data?
- If so, what is causing the lack of responsiveness?

Tip:

- Often in these cases a stakeholder representative has not been appointed within the responsible organization, or the representative doesn't realize the impact caused by their lack of responsiveness.

There are many other possible reasons why dependent stakeholders fail to respond in a timely way. Examples include:

- Too much work on their plates
- Lack of communication
- Lack of understanding of the exact need
- Poor processes within their own organization
- Lack of authorization

With today's move to more agile approaches there is increased demand for stakeholder representative resources. Often dependent organizations, unless they are already working in an agile way, do not have adequate resources to respond in a timely way. Your first goal should always be to figure out the root of the problem so proper corrective action can be taken.

Warning:

Keep on the lookout for stakeholder representatives who may lack authorization to perform their responsibilities.

Pattern Six: "Are We Really Getting Better?"

This pattern demonstrates how a team can easily fall into the "going through the motions" trap by not making value-added improvements that can aid their performance. This scenario should be of interest to all software practitioners, coaches, Scrum Masters, and process improvement professionals.

Scenario start:

I was asked to help a company that was having trouble hitting their schedule commitments. The company was using an agile-Scrum approach on most of their projects. I started my investigation by sitting in on a retrospective [12] listening to the team discussion.

The improvements they were planning to put in place during the next sprint surprised me because I didn't hear anything about schedule problems being addressed. So I challenged the team by asking the following question: "Why aren't you hitting your schedule commitments?"

One developer immediately responded: "Because our product backlog is never properly prepared. We always end up doing more work than planned because the requirements are always vague and incomplete."

Question Scrum Masters and Software Practitioners Should Be Asking:

- If your team is using a Scrum approach, are they seriously addressing the real problems on their project in their retrospective discussions?

Scenario continued:

I challenged the team by suggesting that they should reject backlog entries that are not properly prepared. However, a number of team members objected to my suggestion saying that it would never work in their organization. They said that management expected them to do whatever it takes to get the job done and rejecting requirements is not an option.

I then suggested that they should start keeping personal data on the time they spent addressing poorly prepared backlog items, as well as the time they spent designing, coding and test-

ing. I also suggested that when they estimate tasks in the future they should add in the backlog preparation time, and this would help them hit their estimates more accurately in the future.

I further suggested that they use the personal data they collected as rationale in case anyone objected to their estimate, and that they should not reduce their estimate until they had more current data that indicated their schedule performance had improved.

Tip/Warning to Coaches, Scrum Masters and Team Leaders:

- Scrum teams can easily fall into a "going through the motions" syndrome especially during their retrospectives. Scrum teams must continually tackle the hard problems they have seeking out high value performance improvements. Coaches should constantly be on the look-out for the "going through the motions" sign and continually challenge the team to seek higher performance.

Conclusion

Patterns don't replace your company processes, or what your team is doing today. They are an aid that can help your team's critical-thinking and decision-making. They can help team's make better decisions by reminding them of common scenarios, questions to ask, conditions to be aware of and options they have that can ultimately lead your organization to higher sustainable performance.

I encourage you to develop your own patterns to help guide your software developers in making better decisions. I also encourage you to involve your own experienced practitioners in developing your own scenarios, relevant questions, tips and warnings based on your specific context, organizational constraints, and culture. If you are unsure whether patterns can help your practitioners beyond whatever you are doing today, consider polling your practitioners to get the opinion of those you seek to help.

ABOUT THE AUTHOR

Paul E. McMahon, Principal, PEM Systems (www.pemsystems.com) has been an independent consultant since 1997. He has published more than 45 articles and multiple books including "15 Fundamentals for Higher Performance in Software Development." Paul is a Certified Scrum Master and a Certified Lean Six Sigma Black Belt. His insights reflect 24 years of industry experience, and 17 years of consulting/coaching experience. Paul has been a leader in the SEMAT initiative since 2010.

Phone: 607-798-7740
E-mail: pemcmahon1@gmail.com

NOTES

1. For more information on patterns and pattern examples refer to the book, "15 Fundamentals for Higher Performance in Software Development."

REFERENCES

1. IDC Study: How Many Software Developers Are Out There? <http://www.infoq.com/news/2014/01/IDC-software-developers>
2. Gamma, Erich, Helm, Richard, Johnson, Ralph, Vlissides, John, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995
3. For every hour cutting code, how many hours are spent doing something else? <http://pm.stackexchange.com/questions/10116/for-every-hour-cutting-code-how-many-project-hours-are-spent-doing-something-el>
4. Ambler, Scott, An Introduction to Process Patterns, <http://www.ambysoft.com/downloads/processPatterns.pdf>
5. McMahon, Paul E, 15 Fundamentals for Higher Performance in Software Development, PEM Systems, 2014,
6. Cohn, Mike, User Stories Applied, Addison-Wesley, 2004
7. Jacobson, Ivar, NG, Pan-Wei, McMahon, Paul E, Spence, Ian, Lidman, Svante, The Essence of Software Engineering: Applying the SEMAT Kernel, Addison-Wesley, 2013
8. Jacobson, Ivar, Ng Pan-Wei, McMahon, Paul, Spence, Ian, Lidman, Svante, The Essence of Software Engineering: The SEMAT kernel, Oct, 2012, ACMQueue, <http://queue.acm.org/detail.cfm?id=2389616>
9. OMG Essence Specification, <http://www.omg.org/spec/Essence/Current>
10. SEMAT web site, <www.semat.org>
11. McMahon, Paul E, Integrating CMMI and Agile Development: Case Studies and Proven Techniques for Faster Performance Improvement, Addison-Wesley, 2011
12. Sutherland, Jeff, Schwaber, Ken, "Scrum Guide– The Definitive Guide to Scrum: The Rules of the Game", July 2013
13. Cohn, Mike, Agile Estimating and Planning, Pearson Education, 2006.