JOIN OVER 100,000 PROS! Join Log In | Member Benefits

STICKYMINDS

Agile Testing DevOps Security AI

Survival Rules and the Lamp Lighter

ARTICLE

By Steve Berczuk | May 23, 2012

SUMMARY

By understanding the context in which their existing practices were meant to work, teams new to agile can more easily decide which of those practices still make sense and which are simply security blankets.

One of the great things about being a parent is that you have an excuse to re-read some classic books. My five-year-old and I recently read The Little Prince. In one part of the book, the little prince talks about various characters he meets on his journey from his planet to Earth. The story of one of these characters, the lamp lighter, reminds me of the organizational inertia some teams encounter when trying to transition to agile software development.

The Story of the Lamp Lighter

On one planet, the little prince encounters a lamp lighter who lights a lamp at dusk and extinguishes it at sunrise. But, at the time of the little prince's visit, the rotation speed of the planet has increased such that the length of a day has shortened from many hours to about a minute. The lamp lighter's formerly low-stress task has become quite stressful.

When the little prince asks how this absurd situation came to be, the lamp lighter explains, "Orders are orders … It's a terrible job I have. It used to be reasonable enough. I put the lamp out mornings, and lit it after dark. I had the rest of the day for my own affairs and the rest of the night for sleeping." [1]

At this explanation, the little prince asks, "And since then orders have changed?"

The Lamplighter replies, "Orders haven't changed. That's just the trouble! Year by year the planet is turning faster and faster, and the orders haven't changed!"

This fanciful story has many similarities to situations that teams encounter when they try to adopt agile practices to fit in with existing ways of working without understanding the

We use cookies to ensure you get the best experience on our website.

Accept

Read More

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Processes and tools, documentation, contract negotiation, and plans are part of the agile process, but they only important to the extent that they aid the things that help teams be agile: interactions and collaborations that help teams deliver working software that meets customers' changing needs.

The Neo-agile Lamp Lighters
Many teams new to agile resemble the lamp lighter in that they try to follow standards and practices that no longer make sense in the context of an agile process. For example, the team may use good technical practices, but the requirements management policies are too heavyweight to keep the backlog populated or the release management policies discourage frequent deployments, leading to code-line policies that place a drag on the team. In addition to organizational resistance to change, some team members might also use development tools and practices that make it hard for other agile practice to work as well as they could.

While it's possible to use some agile practices without fully embracing agile values, agile is really a system-wide change. Agile development practices work best in the context of agile planning practices, and applying agile methods successfully in one part of the lifecycle will highlight roadblocks in another. The improved visibility of problems and roadblocks can be a challenge, causing some to revert to practices that support documentation and justification rather than embracing the failures as an opportunity to improve. But, the basic context and values of agile methods are different. In agile projects, the working software and the degree to which the software meets customers' needs matters. Documentation and specification matters only to the extent that it helps to deliver software. So, those old practices that have a sole goal of traceability no longer make sense.

Traditional software development practices focus on reducing uncertainty early. Often, this isn't possible, so agile practices manage uncertainty by embracing the inevitability of change. Rather than requiring detailed specifications that will guide teams, agile teams err on the side of lightweight requirements and frequent iterations where product managers review the working software. Rather than extensive integration cycles, agile teams spread testing and integration throughout the development process so that problems are detected early.

By clinging to practices that "worked" (or at least mitigated risk) in the past, the organization makes it harder to adopt agile by making it more difficult to iterate and deliver incrementally. Why do organizations do this? While it might seem that someone wants to sabotage agile adoption, this usually isn't the case. Agile approaches require working differently and that the stakeholders trust each other. Change and trust are both hard, especially when you are on the line for meeting a commitment.

Survival
Standards and conventions, or "orders," can be helpful to working effectively, but it's

We use cookies to ensure you get the best
experience on our website.

For example, detailed specification might give us a certain level of comfort, even though much of the specification is speculative—or even fictional—and won't represent the final product. Having that specification might give us the security that, even if we can't meet a goal, we can explain why we missed it in a way that stakeholders will accept. It's hard not to want to continue to use these survival rules as they are. As Jerry Weinberg writes in *Quality Software Management: First-Order Measurement* [3]:

> Survival rules are not stupid; they are simply over-generalizations of rules we once needed for survival. We don't want to simply throw them away. Survival rules can be transformed into less powerful forms, so that we can still use their wisdom without becoming incongruent.

"Survival" sounds a bit strong until you consider that the motivation behind working a certain way is often a desire not to fail, and sometimes failing is scarier when you are not following established practices. The context behind an agile approach may be different from the context behind your current approach, and the when the context changes, the rules need to change as well.

If you are considering adopting a new approach to software development, you probably believe that your current way is not helping you reach your goals, such as time to market or quality. A key part of adopting an agile approach is to acknowledge successes and failures, analyze then, and understand how to continuously improve.

Adaptable Change
It's important to remember three things when you are trying to be more agile:

- Agile is a system-wide change. Changing practices in one part of the development lifecycle will quickly reveal roadblocks in other parts.
- It's important to periodically examine how well your team is doing. Iteration retrospectives are an important part of improving how you work.
- Practices that seem problematic often are not "bad," per se; they just don't apply to your current situation.

When you encounter your first failure, it will be very tempting to revert to old practices. But, to succeed in agile, it's important to review your results periodically and decide what you can change to improve them. It's difficult for people to look objectively on results without assigning blame or determining the root cause of a problem, so using lightweight, structured techniques such as those in Agile Retrospectives: Making Good Teams Great [4] can make it easier for teams to consider how they can improve while quelling the instinct to revert to prior work habits.

Change the Method, Change the Orders
Changing from a waterfall method to an agile method places a number of stresses on team members. During these times of stress, it's not uncommon for teams or managers to enforce standard policies in an attempt to improve results in the short term. But, doing so will prevent the team from seeing the benefits an agile approach can bring and may cause the agile adoption to fail.

References

We use cookies to ensure you get the best experience on our website.

2. The Agile Manifesto
3. Weinberg, Gerald M. *Quality Software Management, Vol. 1* (New York: Dorset House, 1991).
4. Derby, Esther and Diana Larsen. *Agile Retrospectives: Making Good Teams Great* (Raleigh, NC: Pragmatic Bookshelf, 2006).

Topics: agile, development lifecycles,

ABOUT THE AUTHOR

Steve Berczuk

Steve Berczuk is a Principal Software Engineer with experience as a manager, Scrum Master and technical lead in Boston, MA. The author of *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*, he is a recognized expert in software configuration management and agile software development. Steve is passionate about helping teams work effectively to produce quality software. He has an M.S. in operations research from Stanford University and an S.B. in Electrical Engineering from MIT, and is a Certified ScrumMaster. Contact Steve at steve@berczuk.com or visit berczuk.com and follow his blog at blog.berczuk.com.

UPCOMING WEBINAR

LETS HANG!

We use cookies to ensure you get the best experience on our website.

SECURITY

Shifting Security Left: DevSecOps eGuide

White Paper | StickyMinds

Log In or Register to post comments

USER COMMENTS

0 comments

YOU MAY ALSO LIKE

SECURITY

Agentic AI Opens Up New Security Flaws

Nahla Davies | August 27, 2025

SECURITY

Countering the Threat of Ransomware-as-a-Service (RaaS)

Nahla Davies | August 22, 2025

SECURITY

Building Trust in Blockchain: The Role of Smart Contract Testing

Ekaterina Egorova | November 18, 2024

Join the crew for FREE! Access exclusive content,

We use cookies to ensure you get the best experience on our website.

See All Member Benefits

We use cookies to ensure you get the best experience on our website.