An Example Checklist for ScrumMasters

Michael James September 14, 2007 (Revised November 13, 2009)

A FULL TIME FACILITATOR

An adequate ScrumMaster can handle two or three teams at a time. If you're content to limit your role to organizing meetings, enforcing timeboxes, and responding to the impediments people explicitly report, you can get by with part-time attention to this role. The team will probably still exceed the baseline, pre-Scrum expectation at your organization and probably nothing catastrophic will happen.

But if you can envision a team that has a great time accomplishing things no one previously thought possible, within a transformed organization -- consider being a *great* ScrumMaster.

A great ScrumMaster can handle *one* team at a time.

We recommend one dedicated ScrumMaster per team of about seven, especially when starting out.

If you haven't discovered all the work there is to do, tune in to your Product Owner, your team, your team's engineering practices, and the organization outside your team. While there's no single prescription for everyone, I've outlined some things I've seen ScrumMasters overlook.

1. HOW IS MY PRODUCT OWNER DOING?

You improve the Product Owner's effectiveness by helping maintain the Product Backlog and release plan. (Note that only the Product Owner may prioritize the backlog.)

- Is the Product Backlog prioritized according to his/her latest thinking?
- Are all the stakeholder requirements and desirements for the product captured in the backlog? Remember: The backlog is *emergent*.
- Is the Product Backlog a manageable size? To maintain a manageable number of items, keep things more granular towards the top, with general epics at the bottom. It's counterproductive to overanalyze too far past the top of the Product Backlog. Your requirements will change in an ongoing conversation between the developing product and the stakeholders/customers.
- Could any requirements (especially those near the top of the Product Backlog) be better expressed as independent, negotiable, valuable, estimable, small, and testable user stories?
- Have you educated your Product Owner about technical debt and how to avoid it? One piece of the puzzle may be adding automated testing and refactoring to the definition of "done" for each backlog item.
- Is the backlog an *information radiator*, highly visible to all stakeholders?
- If you're using an automated tool for backlog management, does everyone know how to use it easily? Automated management tools introduce the danger of becoming information refrigerators without active radiation from the ScrumMaster.
- Can you help radiate by showing everyone printouts?
- Can you help radiate by creating big visible charts[1]?
- Have you helped your Product Owner organize backlog items into appropriate releases or priority groups?

[1] http://xp123.com/xplor/xp0308/index.shtml

- Do all stakeholders (including the team) know whether the release plan still matches reality, based on the current velocity (story points per Sprint)? You might try showing everyone Product/Release Burndown Charts[2] after the items have been acknowledged as "done" during every Spring Review Meeting. Charts showing both the rate of PBIs completed and new ones added allow early discovery of scope/schedule drift.
- Did your Product Owner adjust the release plan after the last Sprint Review Meeting? The minority of Product Owners who ship adequately tested products on time re-plan the release every Sprint. This probably requires deferring some work for future releases as more important work is discovered.

2. HOW IS MY TEAM DOING?

You are a member of the team, and you're encouraged to set an example by collaborating with them on their work. Before getting too lost in technical tasks, consider your primary responsibilities:

- Is your team in the state of flow? Some characteristics of this state[3]:
    - Clear goals (expectations and rules are discernible and goals are attainable and align appropriately with one's skill set and abilities).
    - Concentration and focus, a high degree of concentration on a limited field of attention.
    - A loss of the feeling of self-consciousness, the merging of action and awareness.
    - Direct and immediate feedback (successes and failures in the course of the activity are apparent, so that behavior can be adjusted as needed).
    - Balance between ability level and challenge (the activity is neither too easy nor too difficult).
    - A sense of personal control over the situation or activity.
    - The activity is intrinsically rewarding, so there is an effortlessness of action.
- Do team members seem to like each other, goof off together, and celebrate each other's success?
- Do team members hold each other to high standards, and challenge each other to grow?
- Are there issues/opportunities the team isn't discussing because they're too uncomfortable?[4]
- Have you tried a variety of formats and locations for Sprint Retrospective Meetings?[5]
- Has the team kept focus on acceptance criteria? Perhaps you should conduct a mid-Sprint checkup to re-review the acceptance criteria of the product backlog items committed for this Sprint.
- Does the Sprint Task list reflect what the team is actually doing?
- Does your team have 5-9 people with a mix of skills?
- Are your team's task estimates and/or your taskboard up to date?
- Are the team self-management artifacts (taskboard, Sprint Burndown Chart, impediments list, etc.) visible to the team, convenient for the team to use?
- Are these artifacts adequately protected from meddlers? Excess scrutiny of daily activity by people outside the team may impede team internal transparency and self management.

---

[2] Mike Cohn, *Agile Estimation and Planning* (2005)

[3] Mihaly Csikszentmihalyi, *Flow: The Psychology of Optimal Experience* (1990).

[4] Kerry Patterson, *Crucial Conversations: Tools for Talking When Stakes are High* (2002). Also consider enlisting a professional facilitator who can make uncomfortable conversations more comfortable.

[5] Derby/Larson *Agile Retrospectives: Making Good Teams Great* (2006).

- Do team members *volunteer* for tasks?
- Are technical debt repayment items (sapping your team's velocity) captured in the backlog, or otherwise communicated with the Product Owner?
- Are team members checking their job titles at the door of the team room, collectively responsible for all aspects of agreed work (testing, user documentation, etc.)?
- Is management measuring the team by collective success?

3. HOW ARE OUR ENGINEERING PRACTICES DOING?

- Does your system in development have a "push to test" button so that anyone (same team or different team) can conveniently detect when they've caused a regression failure (broken previously working functionality)? Typically, this is achieved through the xUnit framework (JUnit[6], NUnit, etc.).
- Do you have an appropriate balance between automated end-to-end system tests (a.k.a., "functional tests") and automated unit tests?
- Is the team writing both system "functional" tests and unit tests in the same language as the system they're developing (rather than using proprietary scripting languages or capture playback tools only a subset of the team knows how to maintain)?
- Has your team discovered the useful gray area between system tests and unit tests?
- Does a continuous integration[7] server automatically sound an alarm when someone causes a regression failure? Can this feedback loop be reduced to hours or minutes? ("Daily builds are for wimps." -- Kent Beck)
- Do *all* tests roll up into the continuous integration server result?
- Have team members discovered the joy of continuous design and constant refactoring as an alternative to Big Up Front Design? Refactoring has a strict definition: changing the internal structure of a system without changing its external behavior. Refactoring[8] should occur several times per hour, whenever there is duplicate code, complex conditional logic (visible by excess indenting or long methods), poorly named identifiers, excessive coupling between objects, etc. Refactoring with confidence is only possible with automated test coverage. Holes in test coverage and deferred refactoring are *technical debt*[9].
- Does your definition of "done" (acceptance criteria) for each functional Product Backlog Item include full automated test coverage and refactoring? You're unlikely to build a potentially shippable products every Sprint without learning eXtreme Programming[10] practices.
- Are team members pair programming most of the time? Pair programming dramatically increases code maintainability and reduces bug rates[11]. It challenges people's boundaries and sometimes seems to take longer (if we put quantity over quality). Lead by example by initiating paired workdays with team members. Some of them will start to prefer working this way.

4. HOW IS THE ORGANIZATION DOING?

- Is the appropriate amount of inter-team communication happening? Scrum-of-Scrums is only one way to achieve this.

---

[6] http:://danube.com/blog/michaeljames/junit_is_not_just_for_unit_testing_anymore

[7] http://www.martinfowler.com/articles/continuousIntegration.html

[8] Martin Fowler, *Refactoring: Improving the Design of Existing Code* (1999).

[9] Michael Feathers, *Working Effectively with Legacy Code* (2004).

[10] http://www.extremeprogramming.org

[11] http://www.agilealliance.org/article/articles_by_category/35

- Are teams independently able to produce working features, even spanning architectural boundaries?[12]
- Are your ScrumMasters meeting with each other, working the organizational impediments list?
- When appropriate, are the organizational impediments pasted to the wall of the development director's office? Can the cost be quantified in dollars, lost time to market, lost quality, or lost customer opportunities? (But remember Ken Schwaber's discovery: "A dead ScrumMaster is a useless ScrumMaster.")
- When appropriate, are the organizational impediments pasted to the wall of the development director's office? Can the cost be quantified in dollars, lost time to market, lost quality, or lost customer opportunities? (but learn from Ken Schwaber's mistakes: "A dead ScrumMaster is a useless ScrumMaster."[13])
- Is your organization one of the few with career paths compatible with the collective goals of your teams? Answer "no" if there's a career incentive[14] to do programming or architecture work at the expense of testing, test automation, or user documentation.
- Has your organization been recognized by the trade press or other independent sources as one of the best places to work or a leader in your industry?
- Are you creating a *learning organization*?

CONCLUSION

If you can check off most of these items and still have time left during the day, I'd like to hear from you.

There's no canned formula for creating human ingenuity. This paper lists points which may, or may not, help in your situation.

Once you start to realize what you could do to make a difference, you may find yourself afraid to do it. This is a sign you're on the right track.

ABOUT THE AUTHOR

Michael James, Software Process Mentor, CollabNet, is a software process mentor and Certified Scrum Trainer, focusing on the engineering practices that enable Agile project management. Having worked in the software industry for more than 20 years as a software developer (formerly "architect"), he has experience in: automated testing that predates the Extreme Programming movement; formal, phased, high-ceremony processes based on DOD-STD-2167A; chaotic non-processes of the dot-com era; and Agile processes including Scrum and XP. Michael maintains a blog at http://blogs.danube.com/.

[12] Craig Larman/Bas Vodde, *Scaling Lean & Agile Development* (2007)

[13] Ken Schwaber, *Agile Project management with Scrum* (2004)

[14] Alfie Kohn, *Punished By Rewards: The Trouble with Gold Stars, Incentive Plans, A's, Praise, and Other Bribes* (1999)

ABOUT COLLABNET

CollabNet leads the industry in Agile application lifecycle management (Agile ALM) in the Cloud. The CollabNet TeamForge ALM platform, CollabNet Subversion software configuration management (SCM) solution, and ScrumWorks project management software enable teams using any environment, development method, and technology to improve productivity up to 50% and to reduce the cost of software development by up to 80%. Millions of users at more than 2,500 organizations, including Applied Biosystems, Capgemini, Deutsche Bank, Oracle, Reuters, and the U.S. Department of Defense, have transformed the way they develop software with CollabNet. For more information, visit www.collab.net.

Instructions

If you have received this checklist as a training assignment and your current (or most recent) employer has been attempting anything like Scrum, please apply this to what you've seen there. Mark each item with one of the following

+	(for "doing well")
Δ	(for "could be improved and I know how to start")
Δ?	(for "could be improved, but how?")
N/A	(for "not applicable" or "would provide no benefit")

If your current (or most recent) employer has not been attempting anything like Scrum, mark each item with one of the following:

+	(for "doing well" "or "would be easy to do well")
Δ	(for "would be a challenge and I know how to start")
Δ?	(for "would be a challenge and I don't know how to start")
N/A	(for "not applicable" or "would provide no benefit")

When all items are marked, list your top 3-5 organizational impediments, whether or not they're derived from this checklist.