スクラムマスターチェックリスト

Michael James

(mj@seattlescrum.com)

Revised 8 Feb 2022

<https://scrummasterchecklist.org>

翻訳：清水 弘毅(しみず こうき)

# あなたはチームのフルタイムのファシリテーター（促進者）ですか？

An adequate Scrum Master 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. Probably nothing catastrophic will happen.

典型的なスクラムマスターは、１度に２〜３チームを担当します。

もしあなたがスクラムマスターの役割を、ミーティングの運営やタイムボックスの意識の促進、周囲の人たちから共有される明らかな障害物を取り除いたりする、ということで十分だと判断するのであれば、パートタイム感覚でもこなすことができます。おそらく致命的なことにはならないでしょう。

But if you envision a team that succeeds at things they didn’t previously realize they could do, consider being a *great* Scrum Master.

一方で、誰もが不可能と思うようなことをチームが成し遂げ、組織変革までを見据えられるなら、あなたは”卓越した”スクラムマスターであると考えられます。

A great Scrum Master can handle *one* team at a time.

“卓越した”スクラムマスターは、一度に一つのチームを担当します。

We recommend one dedicated Scrum Master per team of about six when starting out.

１人の専任スクラムマスターにつき、チーム7名程度から始めることをお薦めします。

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 typical things I've seen Scrum Masters overlook.

もしどのようなことをしたらいいのかイメージがもてない場合は、プロダクトオーナーやチーム、チーム外の組織がどのような考えをもっているか、今どのようなことが起きているのか、チームの技術や開発方法はどうなっているか、理解することから始めましょう。 万能な処方箋ではありませんが、私がこれまで出会ってきたスクラムマスター達が見落としがちなことをまとめました。

Please mark each box with √, ∆, ?, or N/A, as described on the last page.

以下のそれぞれのボックスに、 √, ∆, ?, もしくは N/A をつけてみてください。

チェックの付け方は一 番 後のページに記載しています。

# Part I -- How Is My Product Owner Doing? プロダクトオーナーはどのように過ごしていますか？

Scrum Masters improve Product Owner effectiveness in many ways.

スクラムマスターは、様々な方法でプロダクトオーナーの有効性を高めます。

* Regardless of the number of teams, is there one (and only one[[1]](#footnote-1)) Product Owner for the product?
* チーム数に関係なく、プロダクトに対してプロダクトオーナーは一人（だけ[[2]](#footnote-2)）ですか？
* Is the Product Backlog prioritized according to his/her latest thinking?
* プロダクトバックログは、プロダクトオーナーの最新の考えで優先順位付けされていますか？
* Are newly discovered requirements and desirements being captured in the Product Backlog? Remember: the backlog is *emergent*.
* 新しく発見された要求や要望は、プロダクトバックログに取り込まれていますか？プロダクトバックログは”創発的”であることを忘れないでください。
* 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. Plans will change in an ongoing conversation between the developing product and the users/customers.
* プロダクトバックログは管理しやすいサイズですか？管理可能なプロダクトバックログアイテム数を維持するためには、優先順位が高いほど粒度を細かくし、優先順位が低いほどエピックと呼ばれる大きく一般的なエピソードが入ります。また、優先順位が低いプロダクトバックログアイテムを詳細に分析しすぎるのは逆効果です。計画は、開発中のプロダクトとユーザー/顧客との継続的な会話の中で変化していくものです。
* 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[[3]](#footnote-3)?
* プロダクトバックログの（特に）優先順位が高いアイテムは、独立した、交渉可能な、価値のある、見積もり可能な、小規模な、そしてテスト可能な（INVESTな）ユーザーストーリー[[4]](#footnote-4)として表現できていますか？
* Have you educated your Product Owner about *technical debt* and how to avoid it? One piece of the puzzle may be to write automated test and refactoring into the definition of "done" for each backlog item.
* 技術的負債とそれを回避する方法について、プロダクトオーナーを教育したことがありますか？自動テストとリファクタリングを、各プロダクトバックログアイテムの完成(Done)の定義に書き込むことが、難しい問題を解く一つの鍵になるかもしれません。
* Is the backlog an *information radiator*, immediately visible to all concerned?
* プロダクトバックログは情報発信の起点であり、スクラムチーム、チーム外のステークホルダーを含めて関係者全員がすぐに見ることができる状態になっていますか？
* If you're using an automated tool for backlog management, does everyone know how to use it easily? Sadly, automated management tools usually become anti-collaborative *information refrigerators*.
* プロダクトバックログの管理に自動化されたデジタルツールを使っている場合、誰もがその使い方を簡単に把握できているでしょうか？悲しいかな、自動化されたデジタルツールはたいてい、チームの協調を阻害する情報冷蔵庫になってしまいます。
* Can you help radiate information by teaching how to create big visible charts?
* 大きくて見やすいチャートの作り方を教えることで、情報発信を手助けできていますか？
* Have you helped your Product Owner organize backlog items into appropriate releases or priority groups?
* あなたはこれまで、プロダクトオーナーが適切なタイミングでリリースしたり、優先順位をつけるためにプロダクトバックログアイテムを調整するのを支援したことはありますか？
* Does everyone know whether the release plan still matches reality? You might try showing everyone Product/Release Burndown Charts[[5]](#footnote-5) after the items have been acknowledged as “done” during every Sprint Review Meeting. Charts showing both the rate of PBIs actually completed and new ones added allow early discovery of scope/schedule drift.
* リリース計画が最新の状態を保てているかどうか、スクラムチームやステークホルダーは把握していますか？スプリントレビューミーティングでプロダクトバックログアイテムが「完了」とされた後に、プロダクトまたはリリースバーンダウンチャート[[6]](#footnote-6)を更新して、全員に見せるのはどうでしょう？完了したPBIと新たに追加されたPBIの割合をグラフにすることで、スケジュールとスコープのずれを早期に発見できるかもしれません。
* Did your Product Owner adjust any release forecasts after the last Sprint Review Meeting? If you want to ship a properly tested product by a fixed date, you must constantly reconsider scope. This probably requires deferring some work for future releases as more important work is discovered. Some products can be released every Sprint, or multiple times per Sprint.
* スプリントレビューの後に、プロダクトオーナーはリリース予測を調整しましたか？もし、十分にテストされたプロダクトを決まった日までに出荷したいのであれば、常にスコープを考え直す必要があります。そのため、より重要な作業が途中で発見された場合、一部のプロダクトバックログアイテムを将来のリリースに延期する必要があるかもしれません。 プロダクトによっては、スプリントごとにリリースすることもできますし、スプリントの中で複数回リリースすることもできます。

**Part II -- How Is My Team Doing?**

**Part II --**開発チームの状況はどうでしょうか？

While the Scrum Master are encouraged to lead by the example of collaborating with team members on their work, there is a risk of getting lost in technical tasks. Consider your primary responsibilities to the team:

スクラムマスターは、チームメンバーと協力して仕事を進めるという模範を示すことが求められますが、技術的なタスクに没頭してしまう危険性があります。チームに対する主な責任について考えてみましょう。

* Is your team in the state of *flow*? Some characteristics of this state[[7]](#footnote-7):

あなたのチームはフロー状態[[8]](#footnote-8)になっていますか？この状態の特徴は以下の通りです。

* Clear goals (expectations and rules are discernible and goals are attainable, aligning appropriately with one's skill set and abilities).
* 明確な目標（期待やルールが明確であり、自分のスキルや能力に合った達成可能な目標である）
  + Concentration and focus, a high degree of concentration on a limited field of attention.
  + 選択と集中ができ、限られたこと・もの・場所に注意を向けることができる状態
* A loss of the feeling of self-consciousness, the merging of action and awareness.
* 迷いがなく、一心不乱な状態。
* Direct and immediate feedback (successes and failures in the course of the activity are apparent, so that behavior can be adjusted as needed).
* 直接かつ即時的なフィードバックを受けれる状態（活動の過程での成功や失敗が明らかで、必要に応じて行動を適応することができる）
* Balance between ability level and challenge (the activity is neither too easy nor too difficult).
* 現在の能力レベルとチャレンジのバランスが適切な状態。（アクティビティは簡単すぎず、難しすぎず）
* A sense of personal control over the situation or activity.
* 状況や活動を自分自身で選択し制御できているという感覚がある状態。
* The activity is intrinsically rewarding, so there is an effortlessness of action
* 本質的にやりがいのある活動であり、楽しさがある状態
* Do team members seem to like each other, goof off together, and celebrate each other's success?
* チームメンバーはお互い気が置けない仲で、一緒にふざけたり、お互いに成功を祝ったりするような状態ですか？
* Do team members hold each other accountable to high standards, and challenge each other to grow?
* チームメンバーはお互いに高い責任を持ち、成長するために挑戦し合っていますか？
* Are there issues and opportunities the team isn't discussing because they're too uncomfortable?[[9]](#footnote-9)
* 本来話合うべきなのに、チームが背を向けて、話し合われていない問題はないでしょうか？[[10]](#footnote-10)
* Have you tried a variety of formats and locations for Sprint Retrospectives?[[11]](#footnote-11)
* スプリントレトロスペクティブのいろいろな手法や（気分転換できる）開催場所等を試しましたか？[[12]](#footnote-12)
* Has the team kept focus on Sprint goals? Consider a mid-Sprint checkup to reassess the Sprint Plan.
* チームはスプリントゴールに集中し続けていますか？スプリント計画を見直すための、スプリント中間確認ポイントを設けてもいいかもしれません。
* Does the Sprint taskboard reflect what the team is actually doing? Beware the “dark matter” of undisclosed tasks and tasks bigger than one day’s work. Work not related to the Sprint is an impediment to the Sprint.
* スプリントバックログは、チームが実際に行う予定のもの、行ったものを反映されていますか？誰も知らない未公開のタスクや1日分の仕事より大きなタスクなど、「ダークマター[[13]](#footnote-13)」に注意しましょう。 スプリントに関係のない仕事は、スプリントの妨げになります。
* Does your team have 3-9 people with a sufficient mix of skills to build a potentially shippable product increment?
* あなたのチームは、出荷可能なプロダクトを構築するのに十分なスキルを持つ3〜9人のメンバーで構成されていますか？
* Is your team's taskboard up to date?
* スプリントバックログは最新の状態になっていますか？
* Are the team self-management artifacts visible to the team, convenient for the team to use?
* チームが自己管理している作成物(artifacts)は、チームから見えるようになっていますか？チームにとって使いやすいものになっていますか？
* 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.
* これらの作成物は、チーム外のお節介な人々から守られていますか？チーム外の人による過度なチェックは、チーム内部の透明性と自己管理を妨げる可能性があります。
* Do team members volunteer for tasks? On my first Scrum team we felt like volunteers who also got paid. If it doesn’t feel like that, something’s gone wrong somewhere.
* チームメンバーは自発的に（ボランティアのように）仕事をしていますか？私が担当した最初のスクラムチームは、まるでお給料をもらっているボランティアのような感じがました。あなたのチームにそうした感じがないのであれば、何かが誤っているかもしれません。
* Has the need for technical debt repayment been made explicit in the definition of *done*, gradually making the code a more pleasant place to work?
* 技術的負債返済の必要性がdoneの定義に明示され、コードを書くことがより楽しく心地よい感じになっていますか？
* Are team members leaving their job titles at the door of the team room, collectively responsible for all aspects of agreed work (testing, user documentation, etc.)?
* チームメンバーは、自分の職務範囲にとらわれず、合意した業務（テスト、ユーザードキュメントなど）のすべての側面にチームとして責任を持って取り組んでいますか？

**Part III -- How Are Our Engineering Practices Doing?**

**Part III —** エンジニアリングプラクティスはどうなっていますか？

* Does your system in development have a "push to test" button allowing anyone (same team or different team) to conveniently detect when they've caused a regression failure (broken previously-working functionality)? Typically this is achieved through the xUnit framework (JUnit, NUnit, etc.).
* あなたたちの開発しているシステムには、回帰テストの失敗（以前動いていた機能が壊れてしまう）を見抜きやすい環境はありますか？通常、これはxUnitフレー ムワーク（JUnit、NUnitなど）や、E2Eテスト（Cucumberなど）を活用して見抜きやすい環境づくりを行います。
* Do you have an appropriate balance of automated end-to-end system tests (a.k.a. "functional tests") and automated unit tests?
* 自動エンドツーエンドシステムテスト（機能テストなど）と自動化されたユニットテストのバランスは適切ですか？
* Is the team writing both system tests and unit tests in the same language as the system they're developing? Collaboration is not enhanced by proprietary scripting languages or capture playback tools that only a subset of the team knows how to maintain.
* チームは、システムテストとユニットテストの両方を、開発中のシステムと同じプログラミング言語で書いていますか？チームの一部メンバーだけがメンテナンス方法を知っている独自のスクリプト言語やキャプチャ再生ツールでは、チーム内の協働は促進されません。
* Has your team discovered the useful gray area between system tests and unit tests?
* あなたのチームは、システムテストとユニットテストの間の有用なグレーゾーン*[[14]](#footnote-14)*を発見しましたか？
* Does a continuous integration[[15]](#footnote-15) 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)
* 回帰テストが失敗した時、CI(継続的インテグレーション)[[16]](#footnote-16)サーバーは自動的に警告を通知しますか？ またこのフィードバックループを数時間または数分間に短縮できますか？ （「Daily builds are for wimps.（日に１度のビルドは弱虫のためのものです）」 - Kent Beck）
* Do *all* tests roll up into the continuous integration server result?
* すべてのテスト結果をCIサーバに上げていますか？
* Have team members discovered the joy of continuous design and constant refactoring[[17]](#footnote-17), as an alternative to Big Up Front Design? Refactoring has a strict definition: changing internal structure without changing external behavior. Refactoring should occur continuously, or at least several times per hour. Refactor 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. Neglecting refactoring makes it hard to change the product in the future, especially since it’s hard to find good developers willing to work on bad code.
* チームメンバーは、Big Up Front Designに代わる、継続的な設計と絶え間ないリファクタリングの楽しさを発見しましたか。リファクタリングには厳密な定義があり、「外部の動作を変えずに内部の構造を変更すること」です。リファクタリングは継続的に、あるいは少なくとも1時間に数回行う必要があります。 重複したコード、複雑な条件ロジック（過剰なインデントや長いメソッドでわかる）、不適切な名前の識別子、オブジェクト間の過剰な結合などがあれば、いつでもリファクタリングします。自信を持ってリファクタリングするためには、自動化されたテストカバレッジがあればこそです。リファクタリングを怠ると、将来的に製品を変更することが難しくなります。特に、悪いコードに取り組んでくれる優秀な開発者を見つけることは難しいからです。
* Does your definition of "done" for each Product Backlog Item include full automated test coverage and refactoring? Learning Test Driven Development (TDD) increases the probability of achieving this.
* "Done"の定義には、完全自動化されたテストカバレッジとリファクタリングが含まれ、各PBI毎に実現できるようになっていますか？ TDD（テスト駆動開発）を学習することで、これを実現しやすくなるでしょう。
* Are team members pair programming or mob programming most of the time? This can dramatically increase code maintainability and reduce defect rates. It challenges people's boundaries and sometimes seems to take longer (if we measure by lines of code rather than shippable functionality). Lead by example by initiating paired workdays or mobbing sessions with team members. Some of them will start to prefer working this way.
* チームメンバーは開発時間の多くをペアプログラミングやモブプログラミングに充てられていますか？ペアプログラミングにより、コードの保守性が大幅に向上し、 欠陥率を減らすことができます。 この手法は、チームメンバーの境界に挑戦し、時には時間がかかるように見えます（出荷可能な機能ではなく、コード行数で測定した場合）。スクラムマスターは、ペアワークやモブセッションを率先して始め、模範を示しましょう。そのうちこの手法で仕事をすることを好む人も出てくるでしょう。

**Part IV -- How Is The Organization Doing?**

**Part IV —** 組織の状況はどうでしょうか？

* Is the appropriate amount of inter-team communication happening? “Scrum of Scrums” is only one way to achieve this, and rarely the best.[[18]](#footnote-18)
* チーム間でのコミュニケーションは適切に行われていますか？"スクラム・オブ・スクラム “は、これを実現するための一つの方法に過ぎず、最善の方法ではありません[[19]](#footnote-19)。
* Are teams able to produce working features, even spanning architectural boundaries?[[20]](#footnote-20)
* チーム内で動くソフトウェア（顧客が判断できる結果）を作ることができていますか？アーキテクチャや技術領域をまたいだチームになっていますか？[[21]](#footnote-21)
* Are you conducting an *Overall Retrospective[[22]](#footnote-22)* to resolve cross-team, organizational and systemic problems within the organization?
* 組織内の複数チームが、横断的、組織的、システム的な問題を解決するために、全体ふりかえり（オーバーオールレトロスペクティブ）を実施していますか？
* 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 Scrum Master is a useless Scrum Master.”[[23]](#footnote-23))
* 組織上の問題は、マネージャーのオフィスの壁に貼り付けられ、透明性が高い状態になっていますか？市場投入までに浪費した時間や、 失った品質や、顧客機会の喪失などのコストを、お金で数値化することはできますか？ （Ken Schwaberの 失敗から学びましょう：「Adead ScrumMaster is a useless ScrumMaster.（死んだScrumMasterは役に立たないScrumMasterです）[[24]](#footnote-24)
* 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[[25]](#footnote-25) to do programming or architecture work at the expense of testing, test automation, or user documentation.
* あなたの組織は、チーム全体の目標と、組織の人事評価基準が二軸の両輪として存在していますか？テスト、テスト自動化、ユーザー ドキュメントを犠牲にして、プログラミングやアーキテクチャーの仕事をすることが評価[[26]](#footnote-26)されるような組織であるなら、答えは「いいえ」でしょう。
* 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?
* あなたの組織は、働くのに最高な場所、あるいは素晴らしいリーダーがいると、業界紙や独立機関などに認知されていますか？
* Are you creating a *learning organization*?
* 学習する組織を創造していますか？

**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.

上記項目にほとんどのチェックがつけられても、まだ時間が残っている場合は、ぜひあなたのお話を聞かせてください。 人間の創意工夫を生み出す決まった公式はどこにもありません。 このチェックリストに列挙したポイントは、あなたの役立つかもしれないし、役に立たないかもしれません。一度あなたがチームや組織に変化を起こすために、何が出来るかを考え始めると、実行に移すのが怖くなってくるかもしれませ ん。その気持ちは、あなたがスクラムマスターとして正しい道を歩んでいる証拠なのです。

Organizational Impediment Form

Surface issue:

Root cause (Use five times “Why?”):

Business Impact:

Emotional Impact:

Clear, actionable request:

Organizational Impediment Form

Surface issue:

Root cause (Use five times “Why?”):

Business Impact:

Emotional Impact:

Clear, actionable request:

Organizational Impediment Form

Surface issue:

Root cause (Use five times “Why?”):

Business Impact:

Emotional Impact:

Clear, actionable request:

Organizational Impediment Form

Surface issue:

Root cause (Use five times “Why?”):

Business Impact:

Emotional Impact:

Clear, actionable request:

Organizational Impediment Form

Surface issue:

Root cause (Use five times “Why?”):

Business Impact:

Emotional Impact:

Clear, actionable request:

Organizational Impediment Form

Surface issue:

Root cause (Use five times “Why?”):

Business Impact:

Emotional Impact:

Clear, actionable request:

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”)

∆ (for “could be improved and I know how to start”)

? (for “could be improved, but how?”)

N/A (for “not applicable” or “would provide no benefit”)

Or, 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”)

∆ (for “would be a challenge and I know how to start”)

? (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, declare 2-6 organizational impediments on the attached Organizational Impediment Forms, whether or not they’re derived from this checklist. Choose impediments you have at least 1% hope of changing.

1. <https://seattlescrum.com/Why-Scrum-Isnt-Making-Your-Company-Very-Agile/> [↑](#footnote-ref-1)
2. <https://seattlescrum.com/Why-Scrum-Isnt-Making-Your-Company-Very-Agile/> [↑](#footnote-ref-2)
3. [https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/](https://xp123.com/xplor/xp0308/index.shtml) [↑](#footnote-ref-3)
4. [https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/](https://xp123.com/xplor/xp0308/index.shtml) [↑](#footnote-ref-4)
5. Mike Cohn, *Agile Estimation and Planning*. (2005). [↑](#footnote-ref-5)
6. Mike Cohn, *Agile Estimation and Planning*. (2005). [↑](#footnote-ref-6)
7. Mihaly Csikszentmihalyi, *Flow: The Psychology of Optimal Experience* (1990). [↑](#footnote-ref-7)
8. Mihaly Csikszentmihalyi, *Flow: The Psychology of Optimal Experience* (1990). [↑](#footnote-ref-8)
9. Marshall Rosenberg, *Nonviolent Communication: A Language of Life: Life-Changing Tools for Healthy Relationships* (2003). Also consider enlisting a professional facilitator who can make uncomfortable conversations more comfortable. [↑](#footnote-ref-9)
10. Marshall Rosenberg, *Nonviolent Communication: A Language of Life: Life-Changing Tools for Healthy Relationships* (2003). Also consider enlisting a professional facilitator who can make uncomfortable conversations more comfortable. [↑](#footnote-ref-10)
11. Derby/Larson *Agile Retrospectives: Making Good Teams Great* (2006). [↑](#footnote-ref-11)
12. Derby/Larson *Agile Retrospectives: Making Good Teams Great* (2006). [↑](#footnote-ref-12)
13. [https://ja.wikipedia.org/wiki/暗黒物質](https://ja.wikipedia.org/wiki/%E6%9A%97%E9%BB%92%E7%89%A9%E8%B3%AA)  
    暗黒物質とは： 質量は持つが光学的に直接観測できないもののこと。例えば、「テストを行った」と他のメンバーは言っていても、実際に中身を確認して皆が「確かに行っている」と認識するわけではないので、実際のテスト範囲や書き方について過不足があるかどうか明らかではない状態の比喩として書かれています。 [↑](#footnote-ref-13)
14. グレーゾーンとは：ユニットテストとシステムテストの担当者が別々にいる場合（例えば、ユニットテストは開発チームが行い、システムテストはQA部が行うなど）、ソフトウェアコードの問題を見つけることは難しくなります。なぜならテスト範囲等はお互いに”任せてしまっている”からです。テストの範囲や内容について双方で理解しあわない限りは、「本当に期待通りにソフトウェアが動いているかどうか。問題はどこにあるか。」について知ることはできません。 [↑](#footnote-ref-14)
15. <http://www.martinfowler.com/articles/continuousIntegration.html> [↑](#footnote-ref-15)
16. <http://www.martinfowler.com/articles/continuousIntegration.html> [↑](#footnote-ref-16)
17. Martin Fowler, *Refactoring: Improving the Design of Existing Code* (1999). [↑](#footnote-ref-17)
18. See <https://less.works/less/framework/coordination-and-integration/> for alternatives. [↑](#footnote-ref-18)
19. See <https://less.works/less/framework/coordination-and-integration/> for alternatives. [↑](#footnote-ref-19)
20. https://less.works/less/structure/feature-teams [↑](#footnote-ref-20)
21. https://less.works/less/structure/feature-teams [↑](#footnote-ref-21)
22. <https://less.works/less/framework/overall-retrospective/> [↑](#footnote-ref-22)
23. Ken Schwaber, *Agile Project Management with Scrum* (2004) [↑](#footnote-ref-23)
24. Ken Schwaber, *Agile Project Management with Scrum* (2004) [↑](#footnote-ref-24)
25. Alfie Kohn, *Punished By Rewards: The Trouble with Gold Stars, Incentive Plans, A's, Praise, and Other Bribes* (1999) [↑](#footnote-ref-25)
26. Alfie Kohn, *Punished By Rewards: The Trouble with Gold Stars, Incentive Plans, A's, Praise, and Other Bribes* (1999) [↑](#footnote-ref-26)