Agile Red Flags: The Problems they Signal and Mitigations [1, 2]

George Sammit

INTRODUCTION

Managing agile projects is becoming more prevalent and is escaping the software development realm. When executed properly, agile methods may be considered a risk reduction strategy, e.g. avoiding low end-user utility. They may also provide opportunities, e.g. reduced time-to-market. However, when poorly or improperly executed, the very risks that agility purports to reduce may manifest as problems. Sometimes these problems are subtle – perhaps mere inefficiencies or unreported technical debt – but they tend to snowball over time. Other times, they are forthright. Regardless, they have a negative effect on project performance. Therefore, it is important that project managers (PM) understand how to identify triggers of the fundamental risks associated with agile projects. To achieve that, this work presents a series of red flags – missteps, anti-patterns, warnings – across all phases of the project lifecycle. Grounded in both the author's personal experience and research, these red flags signify the forthcoming realization of agility problems or "not quite agile" agile projects.

The work is organized as follows. A brief introduction to agility is presented making it suitable for readers with little experience with this concept. Similarly, given the extreme popularity of Scrum (and Scrum hybrids) in the agile community, a Scrum overview is presented. The bulk of content enumerates the red flags, the problems that they may be signaling, and what can be done to control/avoid them – in essence an agile risk register. The problems presented are related to fundamental concepts of agility and agile practices, and not specific project types or domains.

BRIEF INTRODUCTION TO AGILITY

Agility is arguably an adaption incremental commitment spiral model [1] for software development which takes a value-based approach to building a system gradually. It relies on high-accountability and concurrent development. Perhaps most important are the risk-based decision gates that drive progression. Around the turn of the 20th century, the software development world was shifting from a traditional, highly structured, sequential delivery model. Why? Perhaps, this

[1] *Editor's note: Second Editions are previously published papers that have continued relevance in today's project management world, or which were originally published in conference proceedings or in a language other than English. Original publication acknowledged; authors retain copyright. This paper was originally prepared for the 15th UT Dallas PM Symposium in May 2023, which the author was unable to attend. It is published here with the permission of the author and conference organizers.*

[2] How to cite this paper: Sammit, G. (2023). Agile Red Flags: The Problems they Signal and Mitigations; prepared for the 15th University of Texas at Dallas Project Management Symposium in Richardson, TX, USA in May 2023; *PM World Journal*, Vol. XII, Issue VIII, August.

was a direct result of a malleability curse on software. Given that software was not the predominant factor in systems development, combine with its perceived easy and lower cost to change, it became the resolution to system development problems – just fix it via software. It also became more prevalent in day-to-day life of many people. During this evolution, practitioners around the world joined and created the Manifesto for Agile Software Development [2] which remains a guiding principle of agility today. There are four key points to this proclamation which form the basis for the 12 principles of agile development (not shown due to page limitations) [2].

1. Human interactions are more fruitful than bureaucratic processes/procedures.
2. Working products are more meaningful and illustrative than documents about them.
3. Customer feedback is superior to interpretations of specifications.
4. Change is inevitable and should be acknowledged and embraced, but planning is important.

One may structure a project around these concepts and achieve high levels of success as has been proven over the years [3]. Agile projects are iterative and incremental refinements of a product. They employ a lean mindset and err on the side of delaying decisions as long as responsibly possible. They rely on high levels of automation to achieve rapid product releases and rely on end-user feedback to help shape the product under development.

SCRUM

Scrum is the most popular of the agile methods [3] although it is technically a framework [4]. The theory is simple – iteratively and incrementally move toward a vision, learning, reflecting, and adapting as you go. In addition to the utmost transparency, it relies on:

- A product owner that prioritizes the work and acts as an interface between the various stakeholders and the Scrum team. A product is implied here, and that is what is being built which could be a component of a product or a sub-system of a system. The work required to achieve the vision for the product is kept as a prioritized list called a backlog.
- The Scrum team is a group of individuals (usually around 7 [5]) that have the skills to build the product and are empowered to do so. They work in increments, often 2–4-week increments [5].
- The Scrum team has no leader, they are self-governing. However, it does have a Scrum master who is responsible for ensuring the team is aligned with the framework, adhering to their agreed upon working rules, are focused on project work, and not blocked or distracted by external entities.
- A product increment is a fully functional feature of the product that provides utility to the end-user. It may be very minor utility, but notion of usability must be present.

At the beginning of each increment (sprint), the product owner prioritizes the work from all of the known work (product backlog), including any new work just discovered. The Scrum team negotiates with the product owner as to how much of that work can be accomplished during the

sprint. The work is elaborated, and the Scrum team commits to completing it. The product owner commits to not changing scope during the increment. That list of work to be performed during the time-bound increment work cycle (sprint) is moved into an increment backlog. Work begins, and each day the Scrum team replans how they will achieve their goal. The product owner begins elaborating the work for the next increment. At the end of the sprint the product owner and Scrum team inspect the product increment and invite stakeholders to provide feedback. This feedback informs the work that the product owner again prioritizes. This loop repeats.

PROJECT INITIATION RED FLAGS

Starting a project off on the right foot is important regardless of development methodology chosen. However, there are a few traps to avoid that are particularly pertinent to agile projects.

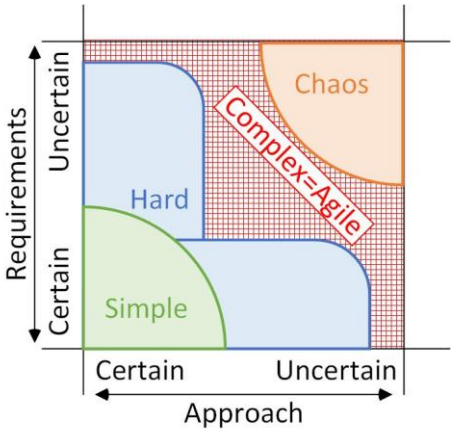
PROJECT MANAGEMENT

One misconception about agility is that it removes the need for project management – that the project magically manages itself every couple of weeks. True, the development team is self-managing. However, they are concerned solely with product development. Agility does leverage a lean and just-in-time mindset, so frivolous actions are indeed removed as they should be on any project. One of the biggest red flags -- one sure to doom a project -- is to forego key components project management. This is particularly the case when a project involves multiple development teams.

Red Flag: Reduced PM. ***Mitigation***: Follow proven PM practices.

THE AGILITY DECISION

The methodology with which the project will be executed is often made during the initiation phase. In fact, it is commonly included in RFPs. In those cases, it is important, regardless of the choice, to make careful, calculated decisions. Far too often is the case that a project sets down the agile path simply for the sake of agility. A very popular and reputable statistic is the Standish Group CHAOS Report for software projects. The latest report purports that agile projects are 3-times more likely to succeed than waterfall projects [6]. Touting such statistics is not sound rationale for deciding a development methodology.



Adapted from [8]

Most proponents of agility agree that it is best used on complex projects, particularly those that have a significant degree of uncertainty and need the capability to adjust course [7] [8]. The key

is that the project and its stakeholders (do not forget the stakeholders) are evaluated prior to deciding to execute in an agile manner. One recommendation is to assess on of the dimensions of: 1) personnel skill level; 2) anticipated dynamisms of the requirements/scope; 3) organizational culture; 4) project team size; 5) criticality of the project [9].

Red Flag: Casually choosing agility. ***Mitigation***: Structured decision analysis.

PROJECT PLANNING RED FLAGS

Agility is a planning-intensive approach where the bulk of it is deferred to the last responsible moment – usually during execution. Regardless, certain up-front planning is necessary for project success.

DOCUMENTATION

As mentioned earlier, an agile approach does not remove the need for pertinent PM practices nor their documentation. For example, one may manage stakeholders differently in an agile project, e.g. ensuring that a single empowered person (product owner) will make all decisions for a particular component of the system. Nonetheless, that individual and other stakeholders still need to be managed and it remains a good practice to document how that will be done in a stakeholder management plan. The same applies to the myriad of other documents common to projects. However, this documentation should be done with a lean mindset which is sound advice on any project. In many cases, an agile project operates in extremes in terms of documentation – either too much (waste) or too little (uncontrolled).

Red Flag: Extremeness in documentation. ***Mitigation***: Right-size and be explicit about the documentation required.

DELIVERABLES

Another issue to be careful of is assumed deliverables. Agility employs a lean mindset and empowers team to operate as best they see fit. If the people delivering the product believe that certain artifacts may not be useful, they may not incorporate them into their work. Therefore, it becomes important to establish the project deliverables. If a design document is needed, it is a project deliverable. More importantly, that deliverable needs to be tied to each product increment. This would mean that the design document (in this example) is kept current with each product increment and reviewed at the end of the increment. Recall that at the end of each increment, the product should be releasable.

Red Flag: Missing deliverables. ***Mitigation***: Tie deliverables to product increments.

FIXED SCOPE V. FIXED TIME

The triple constraint does not evaporate with agility. Arguably, it becomes more of an issue due to the evolving nature on an agile project. There are effectively two options to PMs, and this decision should be made early and rigorously managed.

1. Fix scope and then execute until that scope is achieved or there is no longer benefit from the project. Often the product's features are categorized as must-have, desirable, etc. The must-have features naturally for what is referred to as minimum viable product (MVP). In this mode, you know *what*, the MVP, just not *when*.
2. Fix time. This is the chase where you know *when*, just not *what*. Scope must, therefore, be flexible. The concept of an MVP remains, and hopefully it is achieved before time runs out. A more favorable scenario is that desirable features in addition to the MVP are achieved.

Regardless of choice, the key is to make a choice and manage to that choice. Projects that neglect to do so tend to spiral out of control and neither time nor scope converge.

Red Flag: Not fixing time nor scope. ***Mitigation***: Pick one before execution begins.

HYBRID APPROACHES

A purely agile project is a rarity, particularly in a large organization or when working across organizations. More often, some hybrid approach is taken.

One such common approach is select agility for certain aspects of the project. In fact, this is the approach proposed by Boehm & Turner [9]. The classic example is developing the user interface with an agile methodology to sit on top of a system developed using traditional methods. There are certain subtleties to this approach that must be considered. First, it adds an additional layer of complexity and integration to the project that must be managed, i.e. extra time and cost. Second, it tends to divide teams along the lines of methodology. Particularly when things are not going well, you may end up with camps pitted against one another instead of a unified team. Finally, it relies on the notion that a full-blown plan can coexist with an iterative and incremental plan.

Another interesting approach is to cherry-pick aspects of a certain method. For example, the decision may be made to sprint using Scrum, but not have a releasable product with each sprint. When making such decisions, it is best to weight the cost of agility against the perceived benefit of circumventing it. In this example, the opportunity for feedback is lost as is integration with other components at the increased cost of daily planning. Be prepared for surprises when the decision to put all of the pieces together is made at the end.

The decision to take a hybrid approach is often erroneously considered as a risk mitigation tactic. Ironically, it adds risk for the previously stated reasons. If this is necessary, plan for extra time managing the complexities of a hybrid. Frequent integration on short cycles is also strongly recommended.

Red Flag: Hybrids, in general ***Mitigation***: Think hard about why to choose a hybrid. Integrate frequently if you do.

CONSULTANTS & VENDORS

For organizations transitioning from more traditional methods to agility, it is common to hire consultants and use vendors. There are certain red flags arising from that process that are important to address. When working with vendors, the ability to identify what the Department of Defense calls Agile BS [10] (operating under the guise of agility) is important. In many cases, vendors will inundate the client with buzzwords and capitalize on unfamiliarity with agility to their advantage. They will also tout the use of many popular tools and will typically have a solution before even understanding your problem. Agile consultants are great resource for identifying this. So are asking probing questions going back to the agile principles such as, "Can you explain how that tool fits facilitates the continuous delivery of value?"

Also, be caution of vendors that guarantee delivery of story points. Story points (often just points) are a mechanism for estimating work relative to other work and are unique to individual development teams. Thus, an agreement based on points is effectively meaningless – you will get the number of points in the contract with each increment, guaranteed. In most cases, a time & materials contract is the best bet.

Red Flag: Agile BS from vendors. ***Mitigation***: Ask many questions.

TRAINING

Training is relevant for teams that are considering transitioning to agile methods as well as those that have been doing it for a while. In the case of the former, the team will undoubtedly need training. Simply reading the 14 pages of The Scrum Guide [4] is insufficient to create a high-performing efficient Scrum team that adds value starting with the first increment. All too often, that is the case and PMs are surprised to see the team struggle increment after increment. Formal training and hiring a seasoned coach is a smart idea. Similarly, after choosing an agile method because of the complexity of the project, many PMs simply assume that the team will have all of the knowledge then need to solve that complex problem as it evolves in front of them. It would be rare that the team will not need any training on a complex project.

Red Flag: A project plan that does not include training. ***Mitigation***: Plan for training.

PROJECT EXECUTION RED FLAGS

SPRINT 0 OR WARMUP INCREMENTS

It is not uncommon for a forming team to create a pre-increment increment, often called sprint 0 on Scrum projects. The idea is that this is a ramp-up period where the team does not necessarily produce a working, demonstratable product increment. It often consists of tasks such as setup,

configuration, planning, etc. The problem is that it is counter to the notion of sprinting – delivering product increments on a standard reliable cadence. More problematic is effect on the punctuated-equilibrium model [11] which states that the first group meeting sets the tone for approximately the first half the project. In essence, sprint 0 negates the one of the core tenants of agility, incremental working products, through about half of the first product release. In such cases, expect a significant amount of foundational rather than user-facing product increments.

Red Flag: Sprint 0. ***Mitigation***: Plan for setup type work so the first increment if focused on a useable product.

FOUNDATIONAL V. FUNCTIONAL CODE

"Foundational" product increments do not produce user-facing functionality and often include satisfying non-functional requirements. They often result in a "demonstration" of code, documentation, tests, etc. during the product review at the end of the increment. A three-tier software system commonly consists of a data layer, business logic layer, and a presentation layer. Traditional logic would dictate that the database is foundational to the business services which are foundational to the user interface, and implement them in that order for the given problem. An agile mindset would break the problem down as much as possible such that all three layers are implemented in an increment. "Can't be done" or "It will be more efficient to do it all at once" are likely to be the rationale behind a foundation increment. It can be done, and efficiency is superseded by working products from which feedback may be generated.

Red Flag: An increment ending without a functional product. ***Mitigation***: Demand to see a working product.

PRODUCT HARDENING

Hardening refers to taking a step back and reviewing the product holistically for weaknesses. Those weaknesses are then addressed or hardened. Often these weaknesses manifest themselves as: 1) non-functional requirements; 2) accumulated technical debt; 3) inefficient processes and procedures. Product hardening is the one exception to the rule of producing new user-facing functionality each increment.

There are two common approaches that should be considered. First is the notion of a dedicated hardening product increments which should occur on a predetermined schedule. The alternative is to set aside time in each increment to address the aforementioned and tackle them in small chunks. Regardless of choice, the development team, not the product owner, chooses the work to be done. Furthermore, this should not be seen as an opportunity to "catch-up" on product features. Failure to allow for hardening increments is red flag that indicates an unrealistic notion that agility equates to perfection.

Red Flag: Lack of planning for hardening. ***Mitigation***: Plan for hardening in periodic whole increments or a portion of each increment.

PRIORITY OF WORK

For any given product increment, there are different levels of priority spread across the tasks. They simply cannot be equal, and they should be executed in priority order. A red flag is when a team divvies up the work at the beginning of the product increment and assigns it to individuals, forcing individual responsibility instead of accepting team responsibility. In all cases, the entire team should attack the highest priority task, and no one should work on anything else until either that task is complete or additional team member become detrimental to its completion. Very rare is it that a task may only be worked by one person, although that is often the pushback that the team will provide. Another idea for not operating in the manner is efficiency – it is more efficient to have one person do something. However, it is better to complete a priority #1 task slightly less efficiently than to not complete it at all.

Red Flag: Divvying up all work at the onset. ***Mitigation***: Enforce priority.

OVERTIME

Short increments can be exhausting, particularly if the team feels that they need to work overtime to meet their commitments. There are two issues with overtime that should be considered.

The first is burn-out. Working overtime is not uncommon on projects, but if it is the norm on an agile project developers tend to burn-out rather quickly and a potentially high-performing team can become completely demoralized. Burn-out on agile projects is no different than traditional project, it just happens quicker if not addressed.

The second issue is that it makes it impossible to gauge how much work the team is capable of performing (velocity). When a small task requires the effort of a medium task, the answer should not be "make it up over the weekend." The answer should be questioning why it was considered small in the first place. What was missed? What led to a convergence on that estimate?

One of the goals of agility that many practitioners overlook is the notion of a constant cadence. In Scrum, the increments are called sprints which is a horrible name. Agile projects are marathons, not a series of sprints. Overtime is usually the result of incorrect estimation. Estimation is beyond the scope of this paper, but it should be mentioned that lightweight approached should be preferred and progressively heavyweight approaches applied as the team come closer to performing the work.

Red Flag: Systemic overtime. ***Mitigation***: Ensure understanding of the scope of the task and use an effective means to estimate it

PRODUCT BACKLOG

Given the iterative and incremental nature of agile projects, there is always something in the product backlog. However, that backlog must necessarily continually shrink. A red flag that you may not be heading toward the finish line is a stabilized or growing backlog. There are a couple of reasons that this may occur. When the product is released to users and feedback is provided, it

is easy to get into the trap of, "Yeah, that's a great idea. Let's add it to the backlog." PMs recognize this as scope creep, and it is a difficult problem on agile projects because on one-hand, there is value in innovation, but on the other hand the project is constrained in terms of resources. A clear vision and frequent reviews & prioritization of the backlog may address this issue. The other case is more troublesome and manifests itself when the development team is constantly adding work to the backlog. This may be an indication that what was claimed to have been completed is not really being completed. This is a form of carryover where new tasks are created for the other ones that could not be completed which points to a poor-quality finished product, schedule delays, or both.

Another issue related to the size of the product backlog is the amount of elaborated requirements often referred to as user stories or simply stories. As a rule of thumb, the backlog should have about two increments worth of work ready to be executed at all times. Whatever number you settle on, be leery of extreme shifts from it. An excessively large number of fleshed out requirements is wasteful. A low number indicated lack of direction the inability to pivot should the need arise. The product owner, with help from the development team, should continuously review and update the backlog – a process called grooming.

Red Flag: Non-shrinking product backlog. ***Mitigation***: Align on product vision and definition of done.

Red Flag: Extremes in the number of fleshed out requirements. ***Mitigation***: Continuously groom the product backlog

REQUIREMENTS

Requirements, or stories, are identified at a high-level during release plannings. They are estimated according – at a high-level. As the release progresses, they are elaborated in a just-it-time manner as was described above and re-estimated as more information becomes available. One red flag to watch for is extremes in the level of detail contained in these requirements. Overly detailed requirements are contrary to the notion that interactions are preferred to documentation and may point to wasted effort. They may also point to a lack of trust. Single sentences are likely to be too open ended and not provide enough information to allow the development team to decompose the requirement into workable tasks. What does a good level of detail look like? Prior to electronic requirement management tools, agelists used 3x5 index cards. The requirement was stated on the front and the verification steps on the back. That remains a good rule of thumb. If the requirement cannot be stated in that amount of space, it should probably be broken down more.

Red Flag: Extremes in the detail of stories. ***Mitigation***: Agree on a definition of ready and resist extreme documentation.

CARRYOVER

As a product increment proceeds more information becomes available, and that information may indicate that work committed to cannot be achieved without sacrificing quality. In those cases, the team and product owner need to have a discussion. Note that this may happen on occasion, but should not be the norm.

The product owner may agree to reduced scope noting that he/she agreed not to change scope during the increment. If this occurs, the team should receive credit for only the work accomplished. The unaccomplished work goes into the product backlog for reprioritization. If the development team has not achieved sufficient value, they should not receive any credit. In either case above, many times, the incomplete work is immediately scoped into the following increment. Indeed, it may be the highest priority, but that decision should not be automatic.

Even more concerning is the case where the same task carries over multiple times. This is a good indication that it was either misunderstood or that it was not ready to be worked in the first place.

Red Flag: Systemic carryover. ***Mitigation***: Enforce commitment. Ensure realistic estimates. Do not work on unready tasks.

Red Flag: Carryover automatically appearing in the next increment. ***Mitigation***: Reprioritize incomplete work based on value, not remining effort.

AUTOMATION

As stressed several times, the culmination of a product increment is a releasable product. Most products require varying levels of testing before they are released. Products such as software may require compilation, building, and deployment. Requisites for a product release need to be incorporated in every increment. Such repetition naturally leads to automation, and agile projects without high levels of automation are likely to struggle. At the very least, they are operating inefficiently.

Automation presents a bit of a conundrum. The goal of an increment is a working product, not automation. In many cases automation takes time, and it would be irresponsible to setup a fully automated pipeline before developing any of the product. Therefore, it is important to instill a mindset of progressive automation among the project stakeholders. Using hardening increments to bolster automation is a good use of the team's time.

Red Flag: Lack of automation. ***Mitigation***: Plan for automation early and use hardening increments to beef it up.

PROJECT MONITORING RED FLAGS

The job of monitoring project progress does not disappear with the choice of agility. While the following are red flags in general, they tend to become exacerbated on agile projects.

EXTREMES IN PERFORMANCE

Performance is often measured in velocity which is a quantification of how much work the team can perform in a product increment. There are different ways to measure it that are beyond the scope of this paper, but a common approach is points accomplished. The goal, in terms of output, should be average consistency because without it the ability to plan into the future evaporates quickly. Two extremes to be weary of are teams that always accomplish all the tasks and those that never accomplish all of the tasks.

In the case of the former, either the team is under committing or sand bagging or is reducing quality by accumulating technical debt to meet a deadline. In the case of the latter, the team likely did not fully understand the problem at the onset of the product increment or did not break the work down enough to be achievable. They could also be gold-plating.

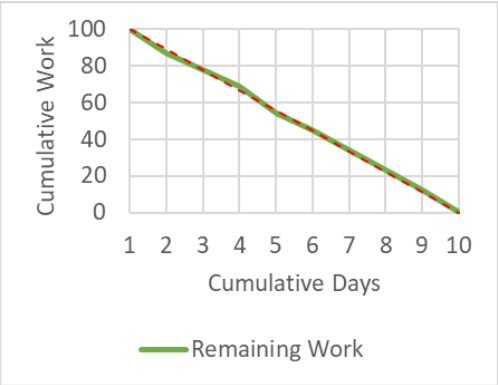
The stakeholders need to agree on when a unit of work is ready for the team to execute (definition of ready). Likewise, the stakeholders need to agree on the level of quality that the system requires (definition of quality) and what constitutes completion (definition of done). These three tools help to address extremes in performance.

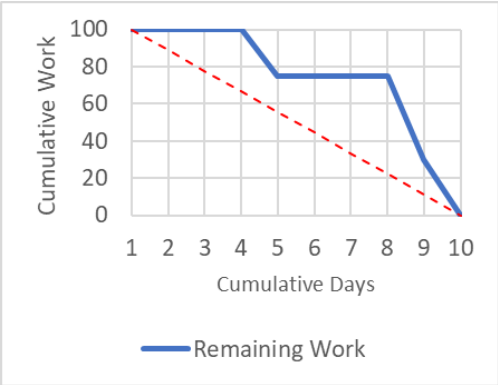
Red Flag: Always perfect or never perfect performance. ***Mitigation***: A complete and concise definition of ready, quality, done.

INCREMENTAL MOVEMENT TOWARD GOALS

With each increment, the product owner sets a goal and identifies, with the help of the development team, the work required to achieve that goal. If the development team, agrees with the scope of work, they commit to it and decompose it into manageable chunks in the increment backlog.

There are many types of agile reports, and one of the most popular is a burndown chart. This chart shows the amount of incomplete work compared to time remaining in an increment. Both of the following charts are severe red flags.





The chart on the left shows an ideal burn-down. That is, the total work was equally divided and equal amounts of work are being completed each day. This is virtually impossible. Greater

deviations around this hypothetical line (almost invisible in chart) are expected. An also alarming situation occurs when true performance runs parallel but offset higher or lower from the ideal burndown. Be skeptical of perfection when operating in a complex domain.

The chart to the right shows an equally alarming situation. Here daily progress is illusive, but somehow the team is able to pull everything together at the last moment. This a highly unlikely feat. Yet burndown charts of this nature are not uncommon across the duration of a project.

So, what does a burndown look like. First, work should be decomposed to the point that tasks are able to be completed daily. When this happens, it is much easier to track progress. Second expect to see the progress line cross the ideal burndown several times. Some tasks will turn out to be easier than anticipated. Others will be harder than anticipated.

Red Flag: Unbelievable burndowns. ***Mitigation***: Monitor daily and determine root cause.

PROJECT CLOSURE RED FLAGS

All projects must end, and agile projects are no exception. If you have made it to this point, congratulations. There should be no red flags to watch for during closure, but remember to retrospect on the project and take a well-deserved break.

CONCLUSION

This paper illustrated a laundry list of about 20 red flags that PMs running agile projects should watch for along with tips to address them. Although all project phases were covered, the bulk of the not-quite-right situation were discussed in the planning, executing, and monitoring project phases. Arguably, each of these red flags deserve their own paper complete with case studies.

Recall that agile methods are a risk reduction strategy, and when poorly or improperly executed, those may unexpectedly arise as issues and problems. The goal of this paper is to assist PMs in running agile project efficiently and correctly thereby realizing those risk reduction strategies and achieving the success with their project that are so frequently thrown around by the agile community.

REFERENCES

[1] B. Boehm, J. A. Lane, S. Koolmanojwong and R. Turner, Incremental Commitment Spiral Model, The: Principles and Practices for Successful Systems and Software: Principles and Practices for Successful Systems and Software, Upper Saddle River, NJ: Addison-Wesley Professional, 2014.

[2] K. Beck, M. Beedle, A. v. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland and D. Thomas, "Manifesto for Agile Software Development," 2001. [Online]. Available: https://agilemanifesto.org/ . [Accessed 5 April 2023].

[3] CollabNet VersionOne, "16th Annual State of Agile Report," digital.ai, 2020.

[4] K. Schwaber and J. Sutherland, "The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game," 2020.

[5] "State of Scrum 2017-2018 Scaling and Agile Transformation," Scrum Alliance, 2019.

[6] J. Johnson, "CHAOS Report: Beyond Infinity," The Shandish Group, 2020.

[7] K. Rubin, "Essential Scrum: A Practical Guide to the Most Popular Agile Process," Addison-Wesley Professional, 2012.

[8] K. Schwaber, Agile Project Management with Scrum (Developer Best Practices), Microsoft Press, 2004.

[9] B. Boehm and R. Turner, Balancing Agility and Discipline: A Guide for the Perplexed, Boston: Addison-Wesley/Pearson Education, 2003.

[10] Department of Defense, "DIB Guide: Detecting Agile BS," Department of Defense Office of Republication and Security Review, Washington, D.C., USA, 2018.

[11] S. P. Robbins and T. A. Judge, Organizational Behavior, 17th ed., Boston, 2015.

[12] Sauce Labs, "Adoption of agile development and continuous integration (CI) in software development worldwide from 2015 to 2018," 28 February 2018. [Online]. Available: https://www-statista-com.proxy.libraries.smu.edu/statistics/673786/worldwide-software-development-survey-agile-development-continuous-integration-adoption/ . [Accessed 6 January 2021].

[13] S. W. Ambler and M. Lines, Choose Your WoW! A Disciplined Agile Approach to Optimizing Your Way of Working, Project Management Institute, Inc., 2022.

[14] M. Cohn, User Stories Applied: For Agile Software Development, Addison-Wesley Professional, 2004.

[15] J. Shore and S. Warden, The Art of Agile Development, O'Reilly Media, 2021.

[16] J. Gido and J. Clements, Successful Project Management, Cengage Learning, 2014.

[17] Project Management Institute, A Guide to the Project Management Body of Knowledge (PMBOK® Guide), Project Management Institute, 2021.

About the Author

George Sammit

San Antonio, Texas, USA

George Sammit has been developing commercial software applications for nearly thirty years and leading such efforts for over twenty. He is a Principal Computer Scientist at the Southwest Research Institute, an independent non-profit research organization which has been advancing science through applied technology for 75 years. There, he serves as an advisor in the Intelligent Systems Division which executes programs and projects that will improve human life. He holds both a B.S. in Computer Science and an M.S. in Program and Project Management from The University of Michigan and is a doctoral candidate at The Southern Methodist University studying Software Engineering. George is an active PMI member holding the PMP credential and an active Scrum Alliance member holding the Certified Scrum Professional credential among others. ■ george.sammit@swri.org ■ https://www.linkedin.com/in/george-sammit/