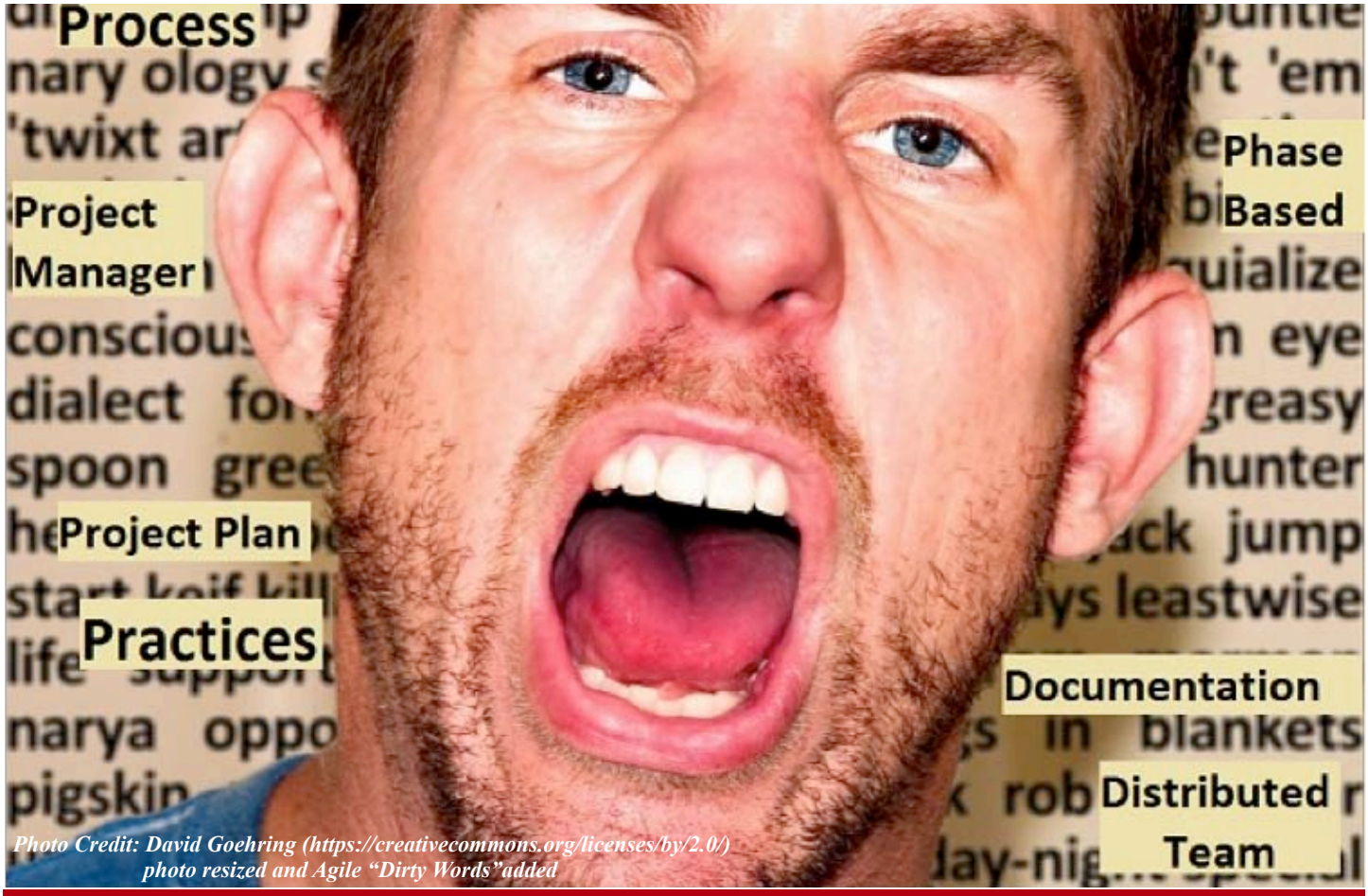


Photo Credit: David Goehring (https://creativecommons.org/licenses/by/2.0/) photo resized and Agile "Dirty Words" added

"7 Dirty Words" to Avoid in Agile/Scrum

By Bob Bretall
Enterprise Agile Coach

There has been a lot of talk about Agile and Scrum in recent years. It's the current darling of the software development world, lauded by some as the 'one true path' and by others with a bit more perspective as the latest evolutionary step in improving software development. There is no denying that when done properly Agile yields a number of wonderful benefits. However, there is no "one size fits all" implementation, nor is the adoption of Agile a silver bullet for all the myriad woes a software development organization may be facing at a given point in time. Agile software development has a lot in common with Lean manufacturing and the two are very complimentary to one another.

The more I learn about and use Agile, the more I come to realize that I've been "Agile aware" for close to 20 years. All the work I've done starting with the Rational Unified Process and on through various iterative and incremental process customizations have followed the majority of the core Agile principles. I started on the path as a consultant with Rational Software and had the great honor of being able to meet and on occasion talk to and interact with the likes of Ivar Jacobson, Grady Booch, Per Kroll, Dean Leffingwell, Kurt Bittner, and more.

As I learned about and implemented the core tenets of Agile over the years there were several concepts where a light bulb went on and I said "*Ah Ha! I've never thought of it that way before!*" For most of them I found that the underlying principles of Agile were things I was aware of and using to greater or lesser degree (depending on the organization) but there are some really useful ideas promoted by the Agile Community on how to technically approach topics from requirements gathering to development and test that when applied help deliver the benefits that Agile promises. In some cases I found that I would

Agile software development has a lot in common with Lean manufacturing and the two are very complimentary to one another.

sometimes prefer a bit of a shift in emphasis away from the pure 'Agile Way' to make things integrate more smoothly with a customer. This is not a knock on the Agile principles, but more about realizing that moving to Agile is a transformative activity that does not necessarily happen overnight like flipping on a light switch.

Even though many of the basic values and principles of software development made it through (mostly) intact, I have also found there to be certain words and phrases that I have seen used for years with no negative connotation in my mind that seem to have become hot-button terms with some members of the Agile/Scrum community. It is not that these terms are universally derided; people who have been around a bit longer and have a sense of software development history can generally do the translation in their head and get to the value of the underlying ideas without getting hung up on the surface terminology. But there are some in the community who have a more 'black & white' interpretation and go directly to the worst case interpretations associated with these words & phrases. In fact, I have seen a number of forum and community posts where they are actually used in a pejorative sense; they have actually become 'dirty words' for some.

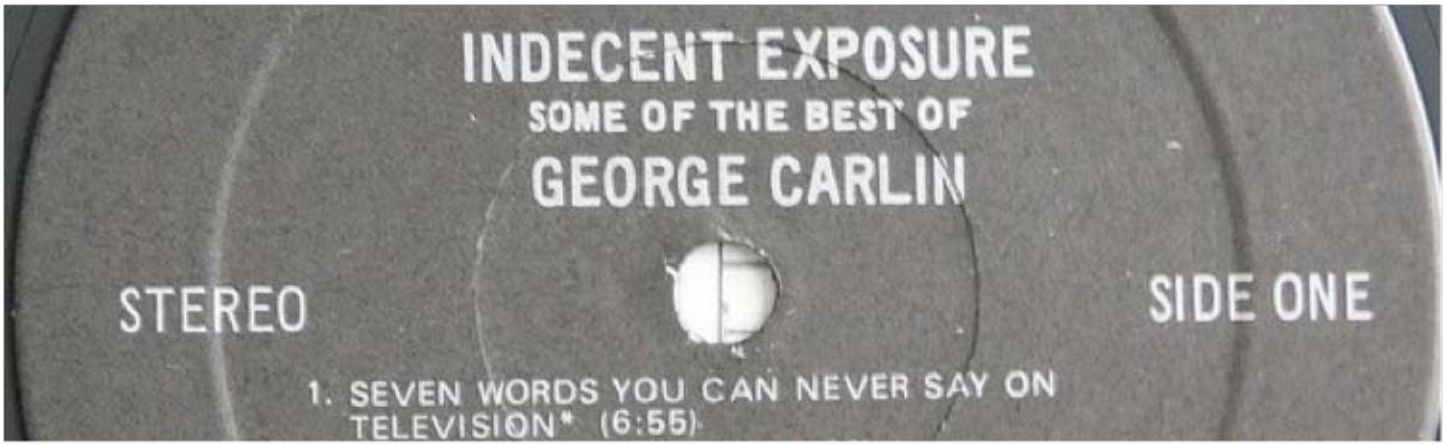
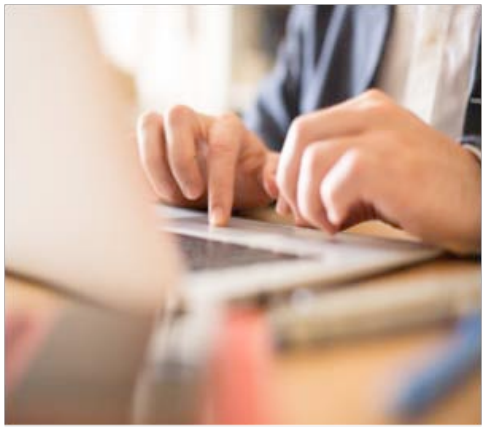


Photo Credit (cropped): Daniel Hartwig (https://creativecommons.org/licenses/by/2.0/)

This made me think of George Carlin's infamous routine "Seven Words You Can Never Say on Television" from 1972 because saying some of the "Agile/Scrum 7 Dirty Words" in meeting or on a forum can sometimes go over like a lead balloon in much the same way as saying one of Carlin's words would have received very negative attention from a TV censor.



Just like a performer avoids the "7 Dirty Words" if they want to be featured on (and asked back to) a television show, we should try to avoid our own techno-babble "7 Dirty Words" if we want to be asked back to help people develop software.

Worse, using these "7 Dirty Words" can lead to a closing off of a listener's objective faculties as they decide that you really don't know what you're talking about. I was initially a bit taken aback at how these terms describing useful core principles of software engineering could be seen by some in a negative light, but have come to realize that meaning is in the mind of the beholder.

Even though you may be well aware of the underlying software engineering concepts, if you aren't couching those concepts in generally accepted Agile terminology your words can be open to misinterpretation as a result. Just like a performer avoids the "7 Dirty Words" if they want to be featured on (and asked back to) a television show, we should try to avoid our own techno-babble "7 Dirty Words" if we want to be asked back to help people develop software better by adopting Agile principles.

While not everyone will consider the following to be "Dirty Words", there are people in the Agile/Scrum community who will go directly to the worst case interpretation of these, so it's my experience to try to avoid hot button words and substitute in alternate ways of getting the same concepts across that are less likely to misinterpretation.

Agile/Scrum "7 Dirty Words"

1. **Process** - This is the BIG one and also steeped in a healthy dollop of irony. Agile and Scrum are themselves software development processes (of a sort, though advocates may say they're lightweight frameworks, not processes). Whatever you call them, they help us understand how to develop software better, which is what a good process should do. The reason I shy away from the word Process is that I have seen its mere mention in certain Agile or Scrum groups received something like shouting out a string of profanities in church. The people who react negatively might say that it's right there in item #1 of the Agile Manifesto:
   Individuals and interactions *over processes and tools.*
   Process seems to be equated to **"high ceremony".** Needless steps that are done just to tick off steps on a process checklist and are not needed for any other reason. There also seems to be a fear that a process will prevent the development team from doing what they want exactly when they want to do it. Certainly there are processes that can be followed that are like this, but the point of customizing and right-sizing a process is to eliminate unnecessary steps that don't add value. Is it done right the first time every time? Certainly not, process definition is iterative and incremental just like software definition.
   By the way, the Agile Manifesto points out *"while there is value in the items on the right"* (processes and tools, in this case), *"we value the items on the left more"* (individuals and interactions).
   I get the feeling that some practitioners stopped reading the manifesto before they got to that final line. I have seen more times than I'd like where individuals throw the baby out with the bathwater and concentrate only on the higher value "items on the left" while to a greater or lesser extent demonizing the "items on the right".

   Stay safe: Avoid saying the "P Word". Talk about the framework and activities that add value to the things you do. To some extent the word process in and of itself adds no value, the value comes from what we do and how we do it.

2. **Documentation** – Nobody doing Agile would really say to totally skip documenting things when you really get down to it. Here I refer to item #2 of the Agile Manifesto:
   Working software *over comprehensive documentation*
   The key concept here is staying away from "comprehensive" (e.g., needlessly overdone) documentation. Right-sizing documentation is just like right-sizing process. Do what needs to be done, what adds value and no more.
   Vision documents are useful high-level documents created by the Product owner. User Stories are documentation of requirements and really help flesh out the details bridging the gap between what is needed and what is going to be built by the team. Saying "Documentation" as a standalone term conjures up visions of heavy-weight forest-depleting reams of paper that are seen as adding no value to the members of the development team. If it smacks of extra work that is not contributing to advancing the product in some way, it's considered superfluous and rightly so. If a 10 item bullet list will get the job done, no need to turn it into a long document with title pages, tables of contents and all other kinds of extraneous material.

   Stay safe: Refer to the things you are creating (user stories, backlogs, etc.) and skip the generic term documentation. If we're following Agile the right way, the things we create along the way should be adequately documenting what is being done without being transformed into Documentation.

3. **Project Plan –** Another 'baby with the bathwater' term. Having a plan is not a bad thing and following a plan does not mean being unresponsive to change. But some have taken item #4 on the Agile Manifesto too far:
   Responding to change *over following a plan.*
   We're right back to focusing too hard on the left part while assuming the right part can be nothing but bad. Certainly there are project plans that go too far and attempt to lock things down tight early. Those are bad project plans. We should not make the assumption that just because there are bad project plans (even if there are a lot of bad project plans) that all project plans must then be bad. User stories, backlogs, the adjustments that come out of a daily meeting and many other artifacts are certainly plans, just not called out by that name. Having a project plan that maps out sprints at a high level and keeps track of how they are tracking against the Vision and Roadmap over time should not be seen as bad thing. Tracking burndown of backlog on a sprint by sprint basis is not a bad thing. Practice good project planning and don't allow yourself to be over-driven by a desire for false precision by parts of the organization who are not 1st-order members of the Agile team and you should be all right (in some organizations, upper management or finance people may insist on seeing a project plan). As with anything, think of your project plan as an iterative artifact that evolves and changes over time. It is possible to have a project plan that is by its nature responsive to change.

   Stay safe: Avoid "P Word #2". If Agile is going to actually deliver on the benefits it promises there are quite a number of specific engineering practices that need to be followed. A process by another name can yield the same benefits. So let's discard the P-Word but still understand that there are things that when not done will lead to non-optimal results. Talk about the Roadmap, Release Plan, Backlog, Sprint Planning, etc. These refer to what you're trying to accomplish and don't evoke images of overly detailed Gantt charts in people's minds.

Even though many of the basic values and principles of software development made it through (mostly) intact, I have also found there to be certain words and phrases that I have seen used for years with no negative connotation in my mind that seem to have become hot-button terms with some members of the Agile/Scrum community.

4. **Project Manager** - The genesis of this as a “dirty word” seems to be rooted in Scrum, not in the broader Agile world. There is not a role for the Project Manager in Scrum, except as a pure people manager or 'traffic cop' kind of role and it does not seem to be seen as a value added role by many in the Scrum community. Based on the spin I'm seeing when the term gets used conversationally, the most common negative connotation associated with it is **"command and control"**, *which is itself a dirty word,* but not one that I have typically see used outside aerospace/defense. As a result, C&C is not a term I’ve used myself very often or at all on the job in the past 10 years. The idea here is that project managers don't add value to the Scrum team but rather come in and tell people what to do and when to do it, really just cramping their style. I won't deny having seen this style of project management in my career, probably more often than I’d have liked to. It's not a style most of the people I'd consider good project managers would ever aspire to. In fact, I had my own term for this kind of project manager back when I was with Rational Software; I'd call these no-value-added managers "spreadsheet managers" who were mostly good for setting up meetings and tracking metrics. Fortunately for me, I had the pleasure of working with many Project Managers in my career who were definitely NOT “spreadsheet managers”. They rolled up their sleeves and worked directly with the teams to help solve problems. Many project managers I have worked with were fairly close to, but not exactly in all aspects, performing the Scrum Master role, and could easily transition to the Scrum Master role in a full-blown Scrum implementation. The point here is that it is not worth getting into a fight over having a role called “Project Manager”, because when moving to Agile, what you really want is someone who is not in a command and control role anyway.

   Stay safe: Avoid “P Word #3” as a descriptor for the attendant role. Embrace the change and go for the underlying value that can be added to the team by looking at this traditional role and tweaking it or distributing the functions among other roles to ensure everything is getting done that needs to be done while not adding extraneous overhead activities.

5. **Phase-Based** - Strongly equated to Waterfall process. The concept of a phase that is not tied to focusing on a class of activities (for instance requirements definition all up front) does not seem to be in the vocabulary. There are phase-based processes that are certainly not Waterfall phases, the emphasis is on the very same core concepts espoused by Agile for ensuring key discovery is taken care of early, embodied in the Agile principles:
   - Balance predictive up-front work with adaptive just-in-time work
   - Validate important assumptions fast

   Stay safe: Avoid “P Word #4” and just concentrate on doing the right things at the right time. A lot of this will be handled if you’re doing proper prioritization of the features and user stories. Ultimately it's not worth beating our head against overloading a new meaning onto a term with a negative connotation, let's focus on the value delivered and go with the flow.

6. **Practices** – ‘Best Practices’ and its close cousin 'Engineering Best Practices' or worse yet 'Software Engineering Best Practices'. This one was puzzling to me. While reading up on things all discussing and espousing their own takes on Best Practices embodied in Scrum and Agile, the actual term "Best Practices" seemed to be treated like some kind of expletive that was synonymous with '***Bureaucracy***'. Certainly not a universal attitude, but I saw negative reactions to the term often enough that I didn’t want to ignore it. I think this stems from historic talk of Software Engineering Best Practices linked with pre-Agile methodologies. This ties in with the interesting concept that a lot of times when I read about Agile the authors jump directly from Waterfall to Agile, as though there were not 10-15 years of other things in between Waterfall’s heyday and the Agile Manifesto. It’s as though anything pre-Agile is considered (incorrectly) to be Waterfall. Instead of accepting that Best Practices are by their very nature not a static thing capturing a single point in time but must, by necessity, continually grow to reflect the current state of the craft of software development.

   Stay safe: Avoid “P Word #5”. While I find the negative reaction by some people to ‘best practices’ to be a bit silly, it’s really not a battle worth fighting. I don't think this is a 100% universal 'gotcha' but it is good to be aware that this is not a universally well-regarded term in some Agile circles.

7. **Distributed Team** – It would be hard to argue with the concept that co-located teams are more efficient, but ignoring the business necessity to sometimes have to work in a distributed environment is naïve. Most seasoned practitioners don't really have a problem with making the best of a bad situation and figuring out ways to deal with the hand that has been dealt to them. People who rail against this concept usually are generally not used to coming to grips with the fact that just **wanting** something to be a certain way does not mean that it will **actually** be that way.

   Stay safe: This one isn't as much of a word to be avoided. It will be what it's going to be based on the realities of the business environment you find yourself developing software in. This 'Dirty Word' is here mostly to call attention to the fact that if you find yourself in an environment that necessitates distributed teams you will need to be particularly wary about making sure things are set up in such a way that the physical separation of team members is causing as little disruption as possible.

Those are the "7 Dirty Words". Are they really **Dirty**? Certainly not to everyone; it appears that the more software development experience a person has, the less likely these will seem particularly dirty. I listed them out this way to have a bit of fun with the ways that different people will perceive the same things and I hope you had some fun reading along with me. The important thing is understanding your audience when communicating and tailoring your message to that audience.

Some Terms Survived Without Change!

While noticing the things that could be misinterpreted by some, I also noticed a variety of concepts that survived largely intact from the pre-Agile (but post-Waterfall) days of software development. These are concepts I've held near and dear for many years that have moved into the Agile world relatively unscathed:

- **Iterative and Incremental** - A core tenet of Agile and Scrum. The main concept here is accepting that you cannot get everything right up front and need to use multiple sprints to help grow understanding and expand our requirements & design in a just-in-time fashion as much as is appropriate. Another trick is getting organizations to accept that requirements are just as iterative and incremental as the software being built and will not all be detailed out at the beginning.

- **Deliver Working Software Frequently –** This is the core intent of the #1, #3, and #7 principles behind the Agile Manifesto**.**
  #1 - Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
  #3 – Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
  #7 – Working software is the primary measure of progress.
  In Scrum the **Potentially Shippable Product Increment** is produced as an artifact of every Sprint and is exactly the same thing that is the goal of providing an executable/testable incremental deliverable as an outcome of every sprint in earlier iterative and incremental development methods. Don't forget the most important part; working and **tested** software as a result of every sprint. A good yardstick to see if a team is practicing "lip-service Agile" is to check if testing is happening in lock-step with development every day of every sprint, or if it is waterfalled in at the end of the sprint.

Looking at your organization and deciding if it is time for some change will often benefit from a new set of eyes helping assess the situation, looking for gaps that can bring value to your organization if they are bridged.

- **Sprints** - While I have always been partial to the word iteration (since iteration is inherent in iterative development), I'm tired of being on the wrong side of history. My definition of iteration is functionally equivalent to a sprint, so let's just call it a sprint. The sprint/iteration is the core concept in iterative and incremental software development. The concept can be corrupted and it does not come for free. There are as many ways to do “code and fix” development in sprints as there are ways to do properly Agile development. We’ll take the sprint as a core foundation principle and build sound software development techniques on top of it.

- **Early Focus on Identifying and Eliminating Risk** - I'll give this one a "kinda sorta". Risk is not an explicit focus in Agile it’s just one of several factors used to prioritize Product Backlog Items. While there isn’t an over-arching theme for identifying and attacking risk early it can easily be a focus as part of grooming the backlog every sprint; risk is something we can easily bring to the table as an emphasis point as appropriate.

As I’ve looked at Agile and seen the value it provides, I believe that a right-sized transition to Agile is a huge value add for almost any organization. The important thing is embracing the underlying principles of agility, avoiding the hot button “7 Dirty Words” while embracing the Agile values that can and should be embedded within them. Some organizations may find changing to Agile is a big deal, but with great change can come great benefits. Other organizations may find that Agile is not a complete paradigm shift and some (or most) of these principles may have been practiced to a greater or lesser extent for years, just under different names. Others may be almost optimized but able to deliver greater value with a bit of a tune up.

Looking at your organization and deciding if it is time for some change will often benefit from a new set of eyes helping assess the situation, looking for gaps that can bring value to your organization if they are bridged. Adding agility to your software development future may not be that far away.

THE DESARA GROUP

Bob Bretall

Enterprise Agile Coach

Bob Bretall has spent over 30 years in the software development field with hands-on experience in all aspects of the software development lifecycle. He has been a developer, designer, team lead, manager, trainer, mentor and consultant.

Read More **Bob.Bretall@DesaraGroup.com**