

Agile Engineering: An Introduction for Managers

by Ryan Shriver

This article introduces the Agile Engineering principles and practices enabling some teams and their respective organizations build very high quality software, very quickly that pleases customers. Organizations embracing Agile Engineering practices, when used in conjunction with Agile and Lean management practices, can gain delivery advantages on their competitors while managing lower maintenance and support costs in the long term.

If you're a leader responsible for delivering value to your stakeholders and customers, you certainly need to know about Agile Engineering. The set of principles and practices, along with the wealth of knowledge and tools available, means teams can start seeing results immediately with a minimal investment. This is the type of positive return on investment that makes everyone happy, especially your boss!

What is Agile Engineering?

Agile Engineering is an umbrella term that refers to the technical principles and practices helping teams deliver high quality releases early and often. It covers both *principles*: those underlying truths that don't change over time or space and also *practices*: the application of principles to a particular situation.ⁱ

Those persons applying the practices in software they develop are considered *agile engineers*. Teams of agile engineers using these proven techniques, primarily around the design and development of the software, can often delight their customers by quickly delivering changes based on feedback. Side benefits are agile engineers typically have greater pride in their work, produce fewer defects can adapt to changing conditions quicker than their counterparts in the industry. And they're [much, much more productive](#).

Hitting the Scrum Wall

"All fine and good", you may be thinking. "But isn't adopting the agile management practices enough?"

While its true that teams adopting Scrum, Lean and other methods improve their performance simply through process changes, improvements can only go so far if there isn't a focus on quality at the most fundamental level - the *code*. The *code* refers to all the source code, configuration files, scripts, images and documentation necessary to ship a release to your customers.

Without a focus on the code, teams hit what's known as the [Scrum Wall](#) where velocity and productivity plateaus and even starts to degrade over time. Early process changes aren't sustainable because of the poor quality of the underlying

code. All the agile naysayers emerge saying "I told you agile didn't work" and your new Scrum team starts floundering. Without a focus on the code and the associated agile engineering principles and practices, your teams and organizations are only going to realize part of the benefits of agile and lean.

Business Benefits

When preparing the business case for making investments, these are the benefits most often cited when teams adopt the practices. I can attest from personal experience that there's a night-and-day difference between teams run using these practices and those not

Improving Quality

The use of automated testing, continuous integration and other practices have a dramatic impact on the quality of the software, resulting in fewer costly and embarrassing defects. The old axiom is true, "You can't go fast unless quality is very high". A focus on quality enables many of the other business benefits of interest to leadership and sets the right tone for the team that taking pride in your work is a fundamental practice we all should share.

Improving Time to Market

Automated build and deployments, along with a focus on simple design and not over-engineering solutions, enables agile engineers to deliver quality solutions to customers quicker and gain their valuable feedback on how to proceed next with confidence the changes they're making to the software are safe.

Improving Productivity

Quicker time to make changes in a safe manner and the automation of testing and repeatable processes allows agile engineers to spend more time on building value-focused solutions and less time on wrestling with the infrastructure. Agile engineering practices are at the center of what Jeff Sutherland coins Hyper-Productive teams; those teams 400% or more productive than their counterparts.

Reducing Maintenance Costs

Some estimate [upwards of 90% of software costs](#) are accrued in the maintenance phases where incremental changes and bug fixes are performed over the lifetime of the product. Sutherlandⁱⁱ and Edelsteinⁱⁱⁱ have put the cumulative figure at \$70 billion in the US alone spent on software maintenance. Changes to legacy code - code without unit tests - can be expensive and lead to long lead times for seemingly minor changes or two new bugs created for every one fixed. Teams I've spend as much as 1/3rd or 1/2 of their team capacity doing nothing but fixing and verifying defects and thereby driving up the maintenance costs! Agile engineering practices that keep the code less brittle and a continuous focus on reducing technical debt can go a long way to reducing maintenance costs.

Reducing Technical Debt

The topic of [Technical Debt](#) is getting the attention of business and technology leaders as an important metric to assess the quality of software and report it in financial terms familiar to executives including such *interest payments on technical debt* due to poor quality and design. The adoption of agile engineering practices is one of the best strategies for paying down technical debt and reducing the risk of expensive system re-writes.

Other Benefits

The benefits to technical teams adopting agile engineering practices are numerous. In addition to the business benefits listed above, other tangible benefits include:

Reducing Complexity

Designing and building only what's needed now, with the confidence the software can be refactored and improved evolutionarily over time, can have a significant impact on reducing software's complexity. This approach allows teams to deliver early releases quickly and then purposely evolve the solution over time as the collective team (business and technical people) gains knowledge of the problem domain. Following the principles of avoiding [Big Design Up Front](#) and deferring commitment until the last responsible moment helps keep complexity to a minimum and ensures complexity is only added when there's an absolute need.

Improving Team Experience

Aside from the benefits to the organization and code quality, agile engineering practices can also improve the working experience for the team members. In my experience, by automating the mundane parts of development and instilling a philosophy of continuous improvement and team camaraderie, teams aren't just more productive they also enjoy the work more. The high emphasis on quality can have a galvanizing effect on the team's work ethic and morale ensuring agile engineering practices stay in place long after first introduced because the team owns them.

Improving End-to-End Thinking

It's often easy to think of software development as just one step in a process of transforming ideas to shippable software. One important benefit Lean brings is the more holistic end-to-end focus on optimizing the whole - from [concept to cash](#). This broader perspective helps avoid sub-optimization of supporting processes and keeps teams and organizations focused on the delivering value to the customer through reduced cycle times (aka time to market) and improved responsiveness. Incorporating ideas such as the theory of constraints can benefit teams to better define and deliver value to their customers while avoiding non-value-add work (waste).

Principles and Practices

Now that you understand what agile engineering is and the core benefits, what are the specific principles and practices teams are using? While the term agile engineering is an umbrella term for the technical principles and practices, the most common ones come from XP, Lean and Evo.

Origins in XP

Much of the foundations of agile engineering come directly from the [eXtreme Programming \(XP\)](#) method. Many signatories of the [Agile Manifesto](#)^{iv} were learning, teaching and writing about ever-better ways to develop quality software using team-centric methods. These early agile evangelists help spread the use of agile engineering practices through publishing, presentations, training and consulting. As the agile community has broadened in the last decade, XP's early popularity has [given way to Scrum's emergence](#) as the leading Agile method and a growing influence from the Lean community.

XP Principles and Practices^v

XP's principles include rapid **Feedback** (at multiple levels), **Assuming Simplicity** in the design and evolution of the software and **Embracing Change** even late in the process in order to satisfy customers.

[XP's Practices](#) help put into practice the core principles. XP teams work as a **Whole Team** to estimate and plan their work through a collaborative **Planning Game** (a Sprint planning meeting in Scrum). When developers create code, they [Pair Program](#) to ensure there's always two people who understand every line of code. While this may seem less efficient, it actually improves productivity and maintainability while reducing coding errors that can lead to delays and cost overruns. Finally, the pair of developers writes a small set of unit tests before they write a line of production code, ensuring each new addition to the system has a corresponding unit test. This is known as [Test-Driven Development](#). All of these practices provide ever-finer levels of **Feedback** that enable teams to support a Continuous Process.

One of the most important practices found on all XP teams is the use of [Continuous Integration](#) to automatically compile, build, test and deploy the software on each change created by the team (on check-in to the source control repository). Automating these key processes has several advantages including improving quality and enabling **Small Releases** to the customer of working, tested software. This continuous process is reflected in the **Design Improvements** of the system that accommodates change for future adaptability and maintainability. Practices such as [Refactoring](#) can be used to incrementally improve the internal structure of the system without breaking customer-facing functionality. When used with unit testing, designs in XP can be modified and changes relatively safely and cost effectively.

XP is a team-centric set of practices and key among them is a **Shared Understanding** of how the system works, ideally through the use of a simpler metaphor everyone grasps. The practices of defining **Coding Standards** and sharing a sense of **Collective Code Ownership** (enabled by **Pair Programming**) ensures the entire team is working together and not going in divergent ways. Supporting this shared understand is the use of **Simple Designs** that only solve for today's problems and don't over-engineer unnecessary architecture and infrastructure when it won't be needed. This approach, when supported by **Design Improvement** practices, enables quicker time to market of smaller releases of functionality.

A team is a collection of people and XP doesn't forget about the individuals. The chief people-practice from XP is finding the team's **Sustainable Pace** - the output level (measured in stories or velocity) the team should be able to deliver indefinitely while working a reasonable schedule. This is the antithesis of *death marches* that characterize waterfall projects and lead to burnout, mistakes and ultimately turnover. XP recognizes that team members need a healthy work-life balance and although there may be short periods of intense activity, such as the Sprint before a release, this is not the norm for long period of time.

Additions by Lean and Evo

In the past decade proven [Lean principles and practices](#) have been adapted from manufacturing processes to software development processes with positive results. These practices related to *kaizen* - Japanese for continuous improvement - are seen as complimentary to those in XP and have recently entered the lexicon of many software teams. Lean's focus on optimizing the flow through the removal of waste and continuous process improvement enables any team to improve their delivery of value to the customer.

Less well known but emergently important are the contributions of Evo - principally through the practice of quantifying qualities such as usability, reliability, performance and maintainability. Evo's focus on system design practices that deliver the maximum business value are complimentary to the code-focused practices of XP.

Lean Principles

Lean Principles are complimentary to those in XP but tend to focus more on the people and their processes and less on the software code and design. The most important Lean principles center on **Improving Flow** of value to the customer while **Eliminating Waste** in the form of non-value add work being done by the team (such as building unnecessary features).

Lean's approach, adapted from its origins as the Toyota Production System, is more holistic and applicable to paradigms outside software development. Similar to XP, Lean encourages team members to **Building Quality In** through the use of automated testing and continuous integration. During development teams **Create Knowledge** by incorporating feedback from stakeholders and creating standards for their work.

Delivering Fast is the Lean principle that encourages small batches and limiting work in process to deliver solutions to market early and often. Doing so requires engaged thinkers so Lean's **Respect for People** provides guidance around topics such as pride, trust, respect and commitment.

Lean's final two principles include **Defer Commitment** whereby irreversible decisions are made at the last responsible moment and multiple options are maintained to provide the maximum flexibility. **Optimize the Whole** recognizes that analyzing a product from the customer's perspective and learning where they derive value can help you produce a more complete solution - more than just working software.

Evo Principles

Evo is the least well-known but is important when emphasizing the *engineering in agile engineering*. Evo provides principles and practices that enable teams to make smart design decisions by **Quantifying Product Qualities** and then framing potential solutions as means to improve important qualities. Evo believes that delivering value to a customer should be done incrementally through the improvement to quality levels, such as Usability, Performance and Reliability. Stakeholders can choose to invest resources to improve these customer-facing qualities or more internal ones such as Maintainability and Adaptability.

Evo's approach encourages teams to iteratively deliver the **Highest Value Next** - the solution that returns the biggest benefit with respect to cost (aka bang-for-the-buck). Although Evo's management practices are outside the scope for most agile engineers, its proven approach on helping engineer solutions is worthy of inclusion in an agile engineer's tool kit.

Supporting Technology

People, process *and* technology are all important aspects of implementing Agile Engineering practices. Technology, in the form of tools, enable agile engineers to standardize core development practices and automate key elements like tests so they run after every change to the codebase.

The past few years has seen a large growth in the availability of new tools supporting agile engineering practices on a variety of platforms. Table 1 shows the most popular agile engineering tools for the [Java](#), [Ruby](#) and [.NET](#) platforms. All tools listed are open source (free) except for those noted with a (\$) sign. For more information on any one in particular, click the link.

Table 1 - Popular Agile Engineering Tools (as of Nov 2010)

Tools	Java	Ruby	.NET	Further Reading
Integrated Development Environment	Eclipse , NetBeans , JDeveloper , IntelliJIDEA (\$), JBuilder (\$)	TextMate (\$), NetBeans , Emacs	Visual Studio (\$), MonoDevelop	Comparison

(IDE)

<u>Source Control</u>	Subversion , Git , CVS , Perforce (\$)	Git , Subversion , Perforce (\$)	Team Foundation Server (TFS) (\$), Subversion , Mercurial , Git , Perforce (\$)	Comparison
<u>Continuous Integration</u>	Hudson , CruiseControl , Continuum , AnthillPro (\$), TeamCity (\$)	Hudson , CruiseControl.rb	CruiseControl.NET , TFS (\$), AnthillPro (\$), Team City (\$)	Comparison , Introduction (Fowler),
<u>Build & Deploy Automation</u>	Ant , Maven , AnthillPro (\$)	Rake , Capistrano , Passenger	NAnt , MSBuild	Comparison
<u>Unit Test Automation</u>	JUnit	Test::Unit , RSpec , Mocha	NUnit , MbUnit , xUnit	
<u>Acceptance Test Automation</u>	Fitnesse , Selenium	FitNesse , Cucumber , Selenium , Watir , Shoulda	FitSharp , Selenium , Visual Studio (\$)	
<u>Code Quality & Reporting</u>	PMD , Sonar , Emma , FindBugs , CPD , Clover (\$), Cobertura , JaCoCo , Simian , jcoverage	Metric-fu , Reek , Roodi , rcov , RailsBestPractices , Saikuro , Flog , Flay , Heckle , Churn	Sonar , NDepend (\$), NCover (\$), dotCover (\$), Visual Studio (\$), Simian , Resharper (\$)	

There are many additional tools that are likely to be in an agile engineer's toolkit. This may include web application frameworks, performance analyzers and data transfer tools. While this toolkit only represents a subset of what you'll likely need, these are the more important ones to include in our ecosystem.

Supporting Ecosystems

Now that you know the essential tools, how do you arrange them together to support agile engineering practices? While a full description is beyond the scope of this article, there are some common and ecosystems found in most teams using agile engineering practices. Table 2 lists these while Figure A shows how they are connected to form a supporting ecosystem for the team.

Table 2 - Common environments for agile teams

Environments	Purpose	Criticality
Turnover	Each iteration's delivery is deployed for use by invited stakeholders	Mandatory
Nightly Integration	Daily stable build deployed for use by limited set of stakeholders	Nice to have for many solutions, mandatory for mission critical ones
Daily Regression	Environment for running daily automated acceptance tests against system	Only needed if you have automated acceptance testing

Staging	Last stop before production	Nice to have for many solutions, mandatory for mission critical ones
Production	Live to your customers	Mandatory
Performance	For doing performance analysis of your software	Nice to have for many solutions, mandatory for mission critical ones

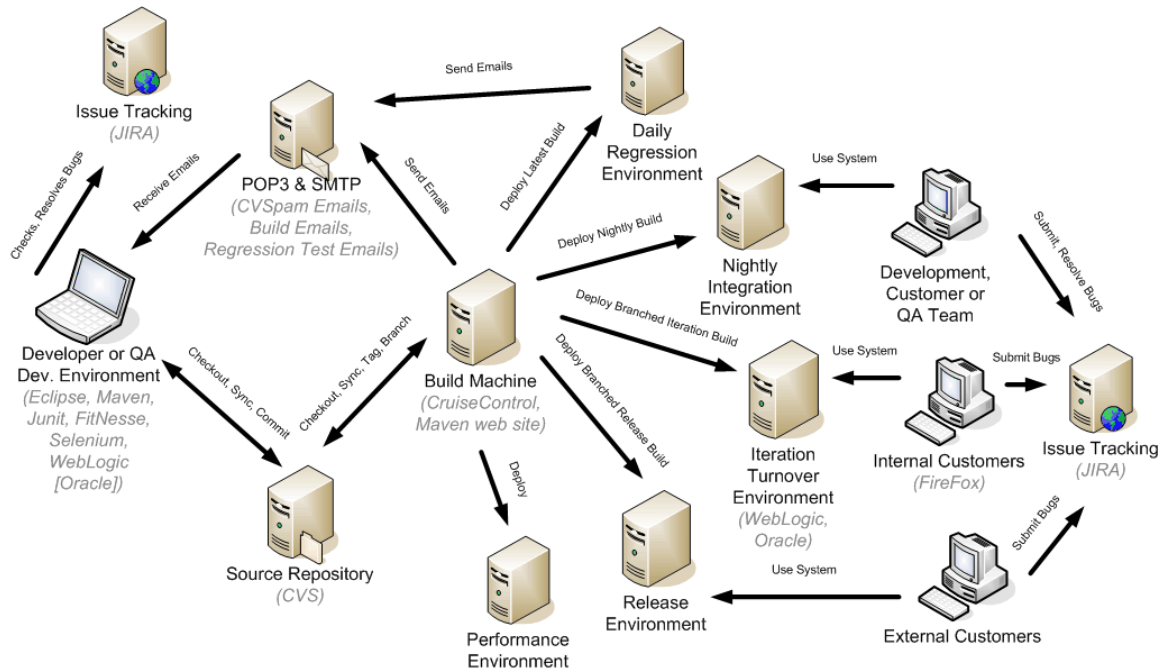


Figure A - Environment used from 2004 to 2007 by the author to develop an enterprise banking product built using Java. Practically all actions resulting from check-ins including customer deployments and testing were automated. This environment supported a team of ~60 analysts, developers and testers doing agile development. Source: [Scaling Agility](#)

Getting Started

There's no right or wrong way to go about adopting agile engineering practices on your teams. With that said, I have seen some proven techniques teams can use to get started including:

Get Educated

First and foremost, get educated on the topic. The fact that you're reading this article is a good start - congratulations. Next, explore some of the links in this article and visit your favorite bookstore or web site. There are typically agile engineering sessions at popular agile and lean conferences and more recently you can even pursue certification in agile engineering practices. While some organizations benefit from training and consulting from thought leaders, others may leverage brown-bag lunches or less formal ways of learning.

Fix the Biggest Pain Now

A good place to start applying practices is to fix the biggest pain in your current situation. *Necessity is the mother of invention* and nothing feels better than to dive right into something's that a sore point for the team and make it better. After an analysis of the problem, see which of the agile engineering practices and supporting tools could improve the situation. Either create a technical story and get it prioritized or have the team allocate some free time to work in the problem. If you are able to make improvements team members will be happy when

Pick a Practice

If tackling the biggest pain isn't possible, simply choose a practice that a team member is especially motivated around and have them lead the team through adoption. This may include developing some training and documentation, but building adoption around motivated individuals creates great opportunities for members of your team to step into leadership positions and be recognized by their peers.

Setup Continuous Integration (CI)

This one may seem too specific but this is by design. Today there's absolutely no excuse for not running CI for all your software solutions. It's no longer a matter of "*should we*", it's a matter of *why shouldn't we*. Given the availability of simple tools for a wide variety of platforms and the enormous benefits for quality and the team, setting up CI is a no-brainer and a good place to start when adopting agile engineering practices.

One Thing at a Time

With a plethora of principles, practices and tools, where to start may seem daunting. Just remember to introduce one or two practices at a time and focus on mastering these before moving on to new ones. Adopting most of the agile engineering practices requires a mental leap for developers and overloading too many changes can have the opposite of the desired effect. Tools like [Sonar](#) can show quality improvements over time so don't try and fix everything at once and just focus on sustainable continuous improvement.

Take a Step Back

While most of these starting points are code-centric, Lean reminds us that it's always important to take a step back and look at the whole - the end-to-end process for how we turn an idea into a release. Exercises like [value stream mapping](#) can be eye opening for opportunities for improving flow and reducing waste, benefits that all team members can get behind.

Summary

It turns out productivity, quality, team morale and costs are all interrelated. Teams leveraging agile engineering practices enjoy a multitude of benefits including faster

time to market and avoiding costly technical debt. Moreover, based on my experience, the teams where these practices were followed were also more fun to work on due to the camaraderie we built around creating a product of such high quality. We had pride in our work and it showed.

The biggest challenges with adopting these practices are both the education and discipline at the individual, team and organization levels to see them through. Knowing *what to do* and *doing it when nobody is looking* are very different things. Getting developers to *say they'll write unit tests* and *seeing test driven development in action* are very different things. Sadly, while only a minority of agile teams utilize many of these practices, those that do help their organizations and themselves while continuing the push for ever-higher quality and more agile solutions.

Author's Note: This article was originally published in [two parts](#) on [gantthead.com](#), but I've decided to publish the combined version here.

ⁱ From [Concept to Cash](#) by Mary and Tom Poppendieck

ⁱⁱ Sutherland, J. (1995). "Business objects in corporate information systems". *ACM Computing Surveys* **27** (2), 274-276

ⁱⁱⁱ Edelstein, D. (1993). "Report on the IEEE STD 1219 – 1993 – Standard for Software Maintenance". *ACM SIGSOFT Software Engineering Notes* **18** (4), p. 94.

^{iv} Chief amongst the technical signatories include Kent Beck, Robert Martin, Martin Fowler, Alistair Cockburn, Ward Cunningham, James Grenning, Andrew Hunt, Ron Jeffries, Brian Marick and Dave Thomas

^v Source: [An Introduction to Agile Engineering Practices](#) by Kane Mar

Ryan Shriver is a Managing Consultant with [Dominion Digital](#), a Virginia-based process and technology-consulting firm. Based in Richmond, he leads the [IT Performance Improvement Solution](#), which includes [Agile Adoption](#), [Agile Engineering](#), [IT Process Improvement](#) and [IT Services Management](#). With a background in systems architecture and large-scale agile development, Ryan currently focuses on measurable business value and systems engineering. He writes and speaks on these topics in the US and Europe, posting his current thoughts at [theagileengineer.com](#). Ryan can be reached at rshriver@dominiondigital.com.