Twice the Product in Half the Time™

The thinking that got us into this trouble is not going to get us out of it (Albert Einstein)

The Problem

With the title of the article as a generic goal for industry, what is going wrong in today's industrial environment? What stops or delays improvements in product delivery, despite embracing practices from Lean and Six Sigma? Why does it take years to get a new product out of the manufacturing plant? This article explores the "why" of this question and deep-dives into the solution.

What is the speed with which product launches really happen? The Tesla Model S was announced and prototypes were shown in 2007, and production started in 2012. Is 7 or 10 years a good estimate for the idea-to-launch time? Boeing's Dreamliner was announced in 2003 and entered production in 2011, estimating the full cycle at 12 to 16 years. Lockheed Martin's F35 started design in 1992 and entered production in 2018. Microsoft Xbox renews every 4 – 8 years. This long time between idea and production does not only affect new products like the Dreamliner, it also affects you when you respond to a competitor's move: for example, when your car brand wants to launch an all-electric vehicle, you find yourself 5 – 10 years behind your competitor who has already launched such a vehicle. Shortening this time span is what the Industrial Agile Framework is all about.

The Charter for Agile Product Development

In learning from our past and exploring the art of the possible, we embrace agile methods as the engine driving innovative solutions and collaboration to amplify economic, ecologic and social benefits around the world.

Through this work we have come to value:

Cross-functional team collaboration over specialization, process & tools.
Modularity over tightly coupled solutions.
Continuous customer collaboration over inflexible contracts.
Fast learning through iterative prototyping over comprehensive delivery.
Extending development through manufacturing over fixing problems in the field.
Useful continuous documentation over comprehensive documentation.

That is, while there is value in the items on the right, we value the items on the left more.[1]

In this article, we play back the commonly perceived causes of lost productivity and the common responses, which are often flawed, to demystify the myths surrounding agile product development.

We will create a running example of the principles and practices we introduce and add some real-life experiences. Our running example looks at a bicycle – high end, carbon fiber, aimed at the next Tour de France (see Figure 1). The bike is new in design with new materials and has to be produced in large batches for distribution in Europe and North America.

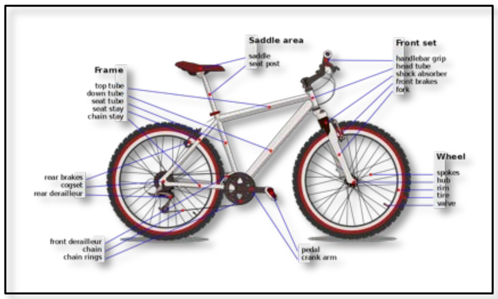

Figure 1: List of bicyle parts.

Perceived causes, flawed responses, and myths

Myth 1: Maximizing people utilization gives the most and cheapest output

Perceived cause: Our "resources", aka people, are underutilized

Flawed response: A "full resource utilization" approach

Skilled people are hard to get, and the process to bring them up to speed on your products and standards adds significant time for those people to become productive. As with equipment, you can optimize how the available people hours are spent and maximize the number of project hours. As you attempt to fill in every available hour with production work, people are allocated to multiple projects, performing specialist work on each project. The result is task-switching, with equipment that might work, although it takes an investment to create the "single minute exchange of dies.[1]"

An alternative to task-switching is outsourcing: doing parts of the project work in a place with lower wages, often many time zones away. The problem of bringing people up to speed gets worse, and the people already in place spend their time helping the outsourced people. It's quite common to see both outsourcing and task-switching occurring simultaneously.

As in manufacturing, the focus in projects is on up-time and utilization of the resource, with the resource being each person involved in a step in the product delivery. With knowledge workers, the opposite of equipment usage in manufacturing happens: more swapping lowers productivity dramatically. Gerald Weinberg[2] researched the topic and produced the numbers shown in Table 1. What his research shows is that a person working on 4 projects at the same time has only 40% of his available time adding value. The exact opposite of the intent. These numbers alone are already bad enough (we regularly witness people assigned to 4 or more deliveries), but a much larger effect is hidden. With a *team* of people working on two projects, the duration multiplier doesn't look awful: 1.25 * the duration of executing the two deliveries in sequence. Now look at the graph in Figure 2, which shows how the duration multiplier is not the only undesired effect from task-switching.

number of simultaneous deliveries	percent of working time available per delivery	loss to context switching	duration multiplier
1	100%	0%	1
2	40%	20%	1.25
3	20%	40%	1.66
4	10%	60%	2.5
5	5%	75%	4

Table 1: lost time vs. number of concurrent tasks

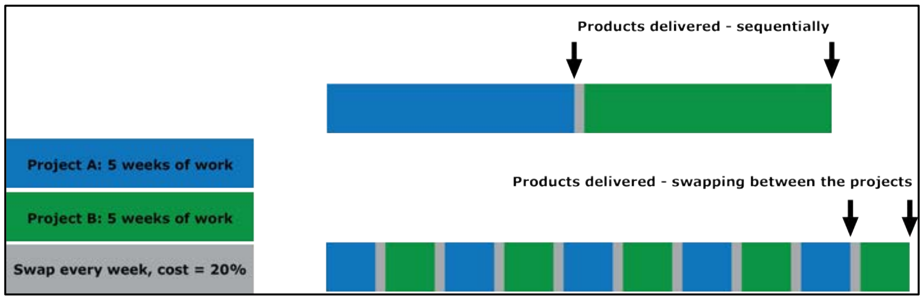
Products delivered - sequentially
Project A: 5 weeks of work
Project B: 5 weeks of work
Swap every week, cost = 20%
Products delivered - swapping between the projects

Figure 2: Go to market moments for sequential vs. swapped delivery

1 Shigeo Shingo, 1985, *A Revolution in Manufacturing: The SMED System*, Productivity Press
2 Gerald Weinbert, 1991, *Quality Software Management: Systems Thinking*, Dorset House

When the two products are delivered in sequence, product A goes to market and creates a return on investment at less than 50% of the time needed to deliver the same product when swapping between them every two weeks. And product B is on the market at less than 100% of the time needed when switching between deliveries. When switching tasks, it will take as long for product A to hit the market as both products together in the sequential delivery. Shortly thereafter the second product is on the market. Time to market for the first product has doubled!

***Industrial Agile Framework Principle*: Focus on one product at a time**, with people not switching tasks, and push this principle for every role and every person needed in the product delivery. In our example of bringing the new bicycle to market it means that the person who develops the disc brakes is always available to work with the team. Make the calculation with the graph above: what is the cost of taking double the time to get a product to market? Does that compensate for the cost of focusing the brake designer? And does a motivated brake designer really sit idle when she sees work that needs to be completed, even when it is not related to brakes? Cross-functional people with a focus on a single product delivery is what you aim for. More about that statement as we discuss Myth 2.

Figure 3 provides a graphical representation of skills requirements during the full product delivery into manufacturing:

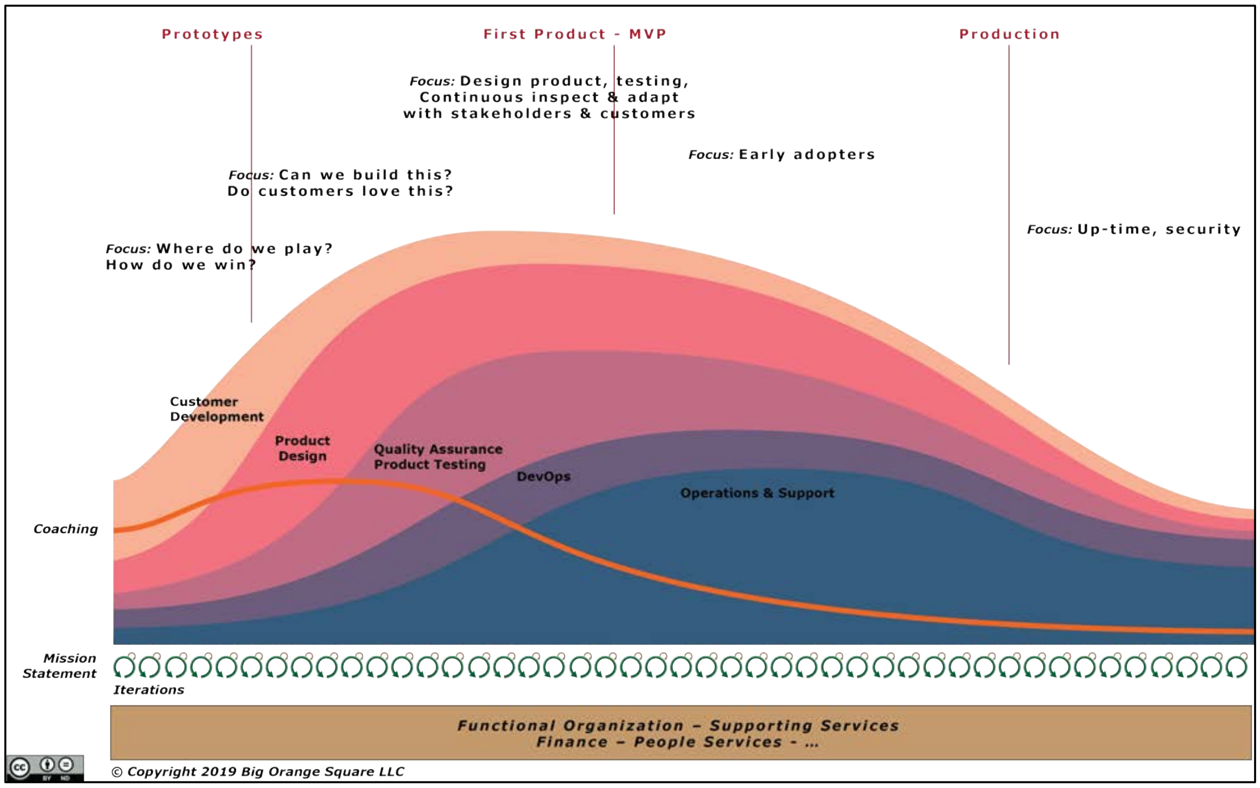

Figure 3: Involved roles and number of people over time

As the product delivery process begins, customer development[3] has the most people active, with designers, testers, supply chain, and manufacturing with fewer representatives. At the peak of the delivery all functions have the most people active; then the roles of customer development, design, and QA start to have fewer people involved. QA will remain in a larger role during production, with supply chain and manufacturing having the largest number of people involved.

Myth 2: Large batches are optimal

Perceived cause: We need to understand all the details before we can build anything

Flawed response: Create a phased delivery plan with large batches of work and long stretches of time

According to this myth, another way to optimize people's time is to have them work on large batches: design the whole product, build a full prototype, test a full prototype, and design manufacturing only when the prototype passes all tests. The thinking is that this will reduce task switching, eliminate mistakes, and achieve the desired high utilization of people. By planning the work in phases, we believe we can prevent problems in the following phases.

The phased approach (often called a waterfall approach) has effects that are undesirable: people not working on the current activity move on to other work (as we describe under Myth 1), which delays problem finding and problem solving. Moreover, the cost of solving problems late is much higher than that of issues that are found and resolved early.

Consider a worst-case scenario: product design and testing finish and manufacturing design starts. The manufacturing design process brings to light how the choice of certain materials or the design of a product feature makes manufacturing in large quantities hard or expensive. Back to the drawing board, with both redesign and retesting being the result.

A true story: In integrated circuit design, we have seen 8-12-month design phases that take the product to what is known as "tape-out," the moment where the very expensive first wafer is produced and sent to the testing phase. But tests haven't been designed yet since that's a later phase, and the testers have been spending their time on the previous chip delivery. You can guess what happens when the ICs arrive in the hands of the testers, and what the cost of a design improvement is.

Many people in industry are used to lean practices, and value stream mapping is an excellent way to create a real-life picture of the hand-offs. The reality of a waterfall process captured in a value stream map, as in Figure 4, shows how messy the flow really is.

[3] Eric Ries, 2011, *The Lean Startup*, Random House

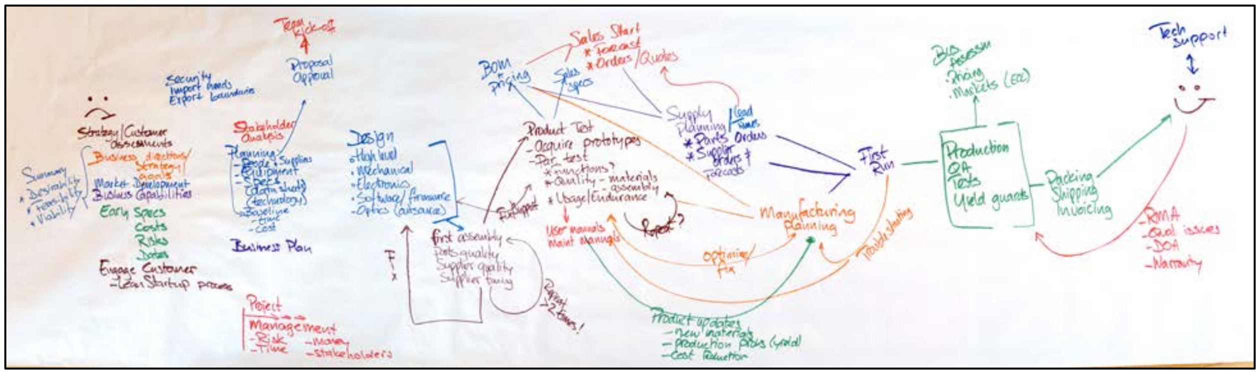

Figure 4: Sample Value Stream Map

When we ran this value stream mapping exercise with the managers of the involved departments, a discussion emerged about problems discovered late in the process. After completing a first prototype, the people responsible for testing often discovered problems with the availability of support materials and insufficient test points. Supply chain had never been informed of the need for those supplies, and the product had been designed neither for testing nor for manufacturing (see Myth 3).

An alternative to a waterfall is to deliver the product design *and* the manufacturing process *and* the supply chain instructions *and* the servicing guidelines in iterations, growing the product to the right set of capabilities, and inspecting the product increment at the end of every short 2-week iteration.

Back to the bicycle example. Most of the challenges in a bicycle are found in the wheels, and therefore these are designed in the first iteration. The rim is 3D printed with carbon fibers, the spokes are standard and available, and the hub is also 3D printed. It takes about two weeks to put all the parts together. Then the sample wheels can be handed out to other design functions and feedback requested. The rim might not fit standard tires (supply chain issue), the hub is expensive to manufacture due to mold complexity (manufacturing / low yield issue), fewer spokes can handle the required tensions if they are produced with a different kind of aluminum (design simplification), and a services rep sees a problem with replacing a full wheel on the road because the width of the rim vs. the width of the tires will cause a problem with a standard brake assembly. And one daredevil of an engineer puts the wheel on his bike and rides around the block. The learning will feed the next design steps, a principle known as empiricism.

The principle works well, but hinges on an upfront agreement between all the people in product delivery: what is the outcome of an iteration? Just the 3D printed parts are not enough for supply chain to give feedback. In addition, supply chain needs a bill of materials. Manufacturing needs the relevant work instructions. Testers need the list of required steps and required test equipment.

This description shows how all relevant people need to focus in a new way for successful product delivery and that people have got work to do throughout the delivery process. For example, while the rim for the bike is being designed, the manufacturing engineer can pair up with the CAD engineer and create the work instructions for manufacturing. And, as queried above, would professionals really sit idle while their teammates are struggling with a problem, even if that problem is outside their normal area of expertise?

***Industrial Agile Framework Principle*: Many small steps are better than a single big bang.** Iterate work every two weeks. Deliver an inspectable result each iteration and inspect with all involved disciplines. Agree between those disciplines what will be contained in an inspectable result: what product components, and at what quality level (is 3D printed good enough?). What documentation, what test results, parts lists, assembly and manufacturing instructions, service notes, etc. are necessary?

Myth 3: Iterating physical product delivery is not possible

Perceived cause: Departments are not optimized for rapid delivery

Flawed response: Local optimization (not seeing "the whole")

With three departments working on design contributions, each department strives to optimize its own work. The waterfall approach from Myth 2 encourages this local optimization. Yet a "good" product is only "good" when all contributions are optimal: the best design, with the right parts, delivered with consistent quality by the manufacturing process – the whole needs to be optimized; a good design alone does not guarantee a good product in the hands of the user. Nor does it create the shortest time from concept to user.

A true story: While we were coaching people working on cash processing equipment, the PCB (printed circuit board) lay-out engineer argued that a proper PCB lay-out would take him 6-8 weeks. "You just have to wait," was his position, focused as he was on what was needed to deliver the best PCB design. In parallel, the mechanical designer was making a lay-out for other components – power supply, scanner, display. Without the PCB design, the mechanical engineer needed to make assumptions about the size of the PCB and the places where the other components would connect. You can see the problem coming: when the physical PCB was ready, the size assumptions that the mechanical designer had made were wrong and the connectors had been relocated. Who will change what, who takes responsibility for the mess, and how will this impact the final cost and delivery date?

When the two engineers talked during our coaching, the mechanical engineer said he would really appreciate a first physical board with just powerlines on it, then later a next version with the data lines, and he didn't care much for thermal provisions at all until late in the design process. "No problem," said the PCB designer, "I'll produce cheap PCBs that you can work with."

The conversation yielded two important results: First, the mechanical designer can give early feedback; for example, "The power supply is located close to the fan – does that generate interference?" The second and crucial result is that the two engineers are communicating early and often. When a next PCB iteration is produced every few days, the feedback and related discussions also happen every few days. Smaller batches are created, people work in parallel, and the whole of the product is seen during the process.

This problem is interesting in its cause: why does local optimization happen? Often local optimization is driven by budgets or rewards. Why would the PCB department look at the supply chain impact when deciding on a better PCB solution? It isn't in that department's interest to solve problems for another team. In some cases, a department is even financially rewarded for its localized work! A team wins team sports games, not just a good offense or defense or a single star player. All team members need to be good and be well-tuned to each other to win the game.

***Industrial Agile Framework Principle*: Optimize the whole.** Returning to our bicycle example: with a first wheel developed in two weeks, the quality inspection starts as soon as the first rim and hub prototypes come out of the 3D printer. Problems are now found as soon as two weeks into the bicycle design. It's easier and cheaper to fix in the next iteration, rather than at the end of the project. The next iteration results in a better wheel, which gets inspected again. Now the tire is on the wheel, which also gets inspected. Feedback is provided, and the next iteration results in two wheels – front and rear, a better fitting tire, and a first printed cassette mounted on the rear wheel. This all gets inspected and tested again. These inspections are all focused on the product and don't create a better process or better collaboration. However, using the rhythm of the bi-weekly product inspections, the teamwork can also be examined and optimized. Should the product inspection show tires not fitting on the wheels, that problem can be resolved. The next questions are: How can the problem be prevented? Should the CAD engineer be sitting in front of the screen with the purchaser? Should the library of basic CAD parts include only available tires in the set? There is not one single solution that works for all products and all teams. Which is why we speak of the Industrial Agile *Framework*: the solution isn't prescribed; the inspection that is leading to a solution that works for this team and this product becomes part of the process that creates a practice that is then inserted into the framework.

Myth 4: Only the most complete feature set will do

Perceived cause: It is too expensive to build product versions

Flawed response: Every possible product feature will be implemented before the first launch

Product development is challenging for both business people and engineers, and one of the challenges is knowing which features to add and when to stop adding more features. Iterative development solves this problem. With short and repeated development cycles, the product grows – the bicycle example starts with wheels, then a frame, pedals, handlebars, water bottle clip, brakes, saddle, and a derailleur. Since the wheels are likely to cause the biggest design challenges they are developed first, then other parts follow. In other words, the wheels have the highest development priority. After every iteration, the business people can inspect the product for completeness and fit-to-market. When the product is deemed complete, the development work stops, and manufacturing starts production. This iterative process needs a few supporting techniques: prioritization, and delivering an integrated working product at the end of every iteration.

The list in the previous paragraph (wheels, frame, pedals, handlebars, water bottle clip, brakes, saddle, and a derailleur) doesn't lend itself to an early fit-to-market: with the saddle late on the list, the bicycle won't be usable until that feature is completed. When the order of the list is changed to wheels, frame, pedals, handlebars, brakes, saddle, derailleur, and water bottle clip, an option appears to define a minimum viable product and a couple of next versions. A first release-to-market is possible when an iteration delivers an accepted, integrated, and tested saddle. Without a derailleur and water bottle clip, the bike won't do well in the Tour de France, but it is a working bike, and the business people can choose to put it on the market. Another few iterations deliver an integrated and tested derailleur and Bicycle Mk. II is on the market. Iterative delivery in the Industrial Agile Framework provides the business people with options.

Different techniques can be used to prioritize the product features (Six Sigma has excellent tools), and when the prior feature is good enough, the next one is taken into development. In the above list of features for a bicycle, how important is the water bottle clip? Or the derailleur? If the water bottle clip isn't too important, then develop it late, or possibly purchase it aftermarket. If time or budget runs out, the bike can still go into manufacturing, and sales can start.

This minimum viable product option only works when every iteration result is an integrated, tested working product. That requires a good ordering of the feature set, and the discipline to complete all the parts that make a product "complete". We explained earlier that all disciplines involved in delivering the bicycle to market are always represented in an iteration. Each discipline completes in the iteration what is needed to satisfy its part of the process: designers deliver the design and necessary documentation, QA people create the necessary tests and validations and execute those, and when problems are found, then these are solved during the iteration. Supply chain people update the bill of materials and start working with suppliers as early as possible; manufacturing processes are designed simultaneously with the product.

A true story: The first versions of an air and noise pollution measurement instrument that we developed were built with cheap sensors, and measurements were shared in the user community. One remark was, "with VOC (volatile organic compounds) levels this high in our neighborhood, we sould all be in the hospital." Tuning was needed, and a next version was shown to the head of atmospheric research at Boulder University. He wrinkled his nose, and explained that the sensors weren't accurate, and too prone to influences like humidity or temperature. The next model (shown in Figure 5) had a calibrated VOC and noise sensor on it. It took only weeks to work with off-the-shelf products and get essential feedback. The product in the above state is still too big and too heavy, and negotiations with suppliers are needed. But all of this is known early in the delivery process, not when a completed product is entering the market.

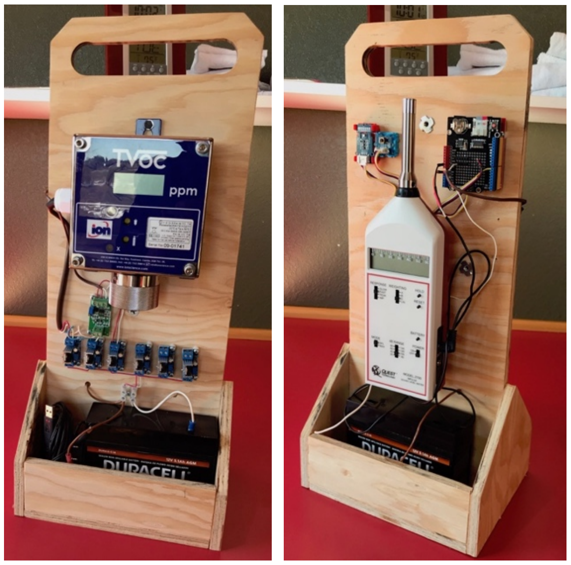

Figure 5: Prototype VOC sensor

Agile development is best known from the software world (although its roots are in industry, dating back to the early 1990s). It's usually thought that software is easy to iterate--a website can be developed page by page--but that for an industrial product, like a brake caliper, that is not possible. Both statements are wrong: it is just as hard to iterate software in a complex environment as it is to produce an industrial product. It took time, thinking, experimentation, and daring over many years to learn how to produce software for life-critical environments using an agile framework. Yet that barrier has been crossed, and software for any purpose is now developed with agile frameworks. Physical products can also be produced using agile, with the same challenges: it will take time for you to think and to experiment and a good amount of daring to develop a brake caliper in two-week iterations.

Industrial Agile Framework Principle: **Iterations are the only way to enable empiricism in product deliveries.** For empirical process control to work, transparency, inspection, and adaptation are needed. Hence, building a real product part will require integrating it in with already completed parts, inspecting it with representatives of the full value chain, and relentlessly improving the product when inspections show what improvements are needed.

Myth 5: Detailed management is needed to guarantee early delivery

Perceived cause: Lack of process and management will result in chaos

Flawed response: More process, more reports, and more checklists

Your experience in your own organization will likely contradict the assertion in this myth: despite the heavy processes that are put in place in the organization, new products are not delivered on time, projects do overrun, and customers are not happy with the delivered product features. Harsh words, and in our interactions with industrial organizations, they are the most often heard opening statements. Albert Einstein is credited with exclaiming, “The definition of insanity is doing the same thing over and over again, but expecting different results.” More process, more controls, more reports are not going to affect product delivery enough to increase the chances of successful delivery and improved performance.

The core of product development is to create something new--a new solution, a new machine – product development is creative in nature, creating something that did not previously exist. Remember that painting by numbers doesn’t make a Picasso out of you, nor does following process steps make you into a Steve Jobs. In the Industrial Agile Framework, all the people involved in attaining a goal have total freedom to reach that goal. Certain parameters definitely must be met in reaching a given goal: safety requirements for a car, say, or cost, or the required use of existing technology. Progress has to be made measurable and visible; otherwise, there is no control of the project. The Industrial Agile Framework provides the freedom to create the best solution, whilst producing quality and progress indicators throughout.

Using our bicycle as a simple example, we can show how progress can be tracked. The bicycle, as shown in Figure 1, has 7 parts, and if we over-simplify the work, each part takes as much work as every other part. Delivery started with the wheels, the part with the most challenges. After 5 two-week iterations, you and the team agree this is the right wheel for the bike. Project duration can now be predicted: 1 part completed over 10 weeks, 6 parts to go, so the total work will take 70 weeks from start to finish. Too simple? Yes, we agree; the complexity and sheer amount of work in each part needs to be better estimated. Integration may get harder and harder, so that the last part may take twice as many weeks. Yet progress and the delays can be measured after every iteration. And every time a slowdown is measured at the end of an iteration, questions can be asked: "Why?," "Will we bring it to market on time?" "What solutions are available?" Lowered productivity is visible very early, and tough choices can be made.

Industrial Agile Framework Principle: **Iterate, and involve the whole team.**

- Iterate, take small steps to deliver the product (the design, the purchasing requirements, the manufacturing processes & tools, the servicing needs).
- Every iteration starts with representatives from each discipline agreeing what the iteration needs to deliver, what product features will be included in the iteration, and who delivers what and how.
- Every iteration ends with representatives from the whole value chain inspecting the delivered increment. This team decides whether there are problems, the importance of the problems, and how problem solving will merge into the work of delivering other parts of the product.

The control structure is a classic plan-do-inspect-adapt structure from Lean, but with requirements around the components of the delivered increment. It has to be as real a product as possible, it has to be integrated with other product components, and it has to include the prototype and the materials needed for testing, purchasing, manufacturing, and service.

Myth 6: Agile product delivery doesn't scale

Perceived cause: Only small teams can work in an iterative way

Flawed response: Doing more of the same process

Building a complex product can have many aspects, requiring many skills to deal with all the various aspects. This often leads to large and distributed groups of people. This scale need is often seen as the reason why agile solutions are not applicable in an organization developing complex products. Let's take a look at three often heard statements about complex deliveries to demystify the scaling issues.

We often hear, "Complexity requires many people to get to a working solution." Is this really true? And if so, why? How many people are truly at the core of developing a car, or a satellite, or an airplane? And can the group be made smaller by developing or choosing people with multiple skills? Referring back to Myth 5, using the same solution and expecting a different result is not going to work. So, start the next project with a small team, and expand that team when needed. With an inspection taking place every two weeks, the need, when and if there is one, will become clear. For example, if needed, a core team member could work with her own sub-team, or a vendor team, but only if other options have failed. Use iterations to learn what fails, and then address the failure, not the other way around; do not simply assume that a sub-team is needed and put one in place. Assumptions are not proof of need.

Another frequently heard statement is, "Many people in a team are hard to manage." That is true, so don't do it. Probably the hardest aspect of the Industrial Agile Framework is ceasing to manage people and instead creating and supporting teams that manage themselves. Management is responsible for clearly defining a mission for the team, establishing goals, and always being available to explain, inspect, and steer. Research shows that people who are autonomous in choosing their own solutions are better motivated and produce more and better results.[4] The iterative aspect of the framework provides early and frequent inspection to steer the mission, or, in the worst case, to make the decision to cancel the project.

The final statement that we often hear is, "Work from multiple contributors doesn't integrate well." True, so integrate early and often to find the problems early on. In the Scrum framework, the result of an iteration is defined as "a shippable product increment" – something that is real and integrated, and over time the sum of the increments produces the full product, which is then released to the market.[5] We often hear that this approach is feasible in software, where Scrum is mostly used, but not in industry. We demystified this objection earlier (in Myth 4) and again emphasize that incremental integration is not easy but is possible.

Industrial Agile Framework Principle: **Only bring more people in when inspections show the need to do so.** Don't make assumptions, don't rely on your previous practices, let the team decide when it is time to scale, and, before you make that happen, relentlessly question that need.

Conclusion: The Industrial Agile Framework summarized.

To summarize how the thinking behind the Industrial Agile Framework is different from that behind classic project management, we reverse the key project management practices:

- No resource utilization focus, but availability focus: Each person works on a single product delivery, completely focused and available up to the delivery.
- No large projects, but product delivery in small steps: Batches of work on the delivered product are extremely small. What is accomplished is as much as the focused group of people can deliver in two weeks.
- No local optimization by department, but product delivery optimization: All relevant people for the delivery of the product into the hands of the customer are always engaged and only focused on the delivery.
- No local optimization or step optimization, but delivering the small batch from each iteration ready for manufacturing: The outcome of each two-week iteration includes design, testing, materials, manufacturing steps, service manuals, etc.
- No prescribed processes, but a light framework and freedom for the team to fill in the framework with the best practices: The delivery team decides what activities are necessary to deliver each small batch.
- No adding people to teams beyond representatives of the value chain, but only when inspections show the need to add a person.

In our network of industrial agile companies, product owners in the agricultural machining domain went from one or two new products every two years to 15 new products every two years.

[4] Dan Pink, 2011, *Drive: The Surprising Truth About What Motivates Us*, Riverhead Books

[5] Ken Schwaber and Jeff Sutherland, 2017, *The Scrum Guide*, http://scrumguides.org

And they did this with fewer people, less process, higher quality, and happier customers. The framework pushes our opening Einstein quote (“The thinking that got us into this trouble is not going to get us out of it”) very far: the thinking to get you out of trouble must produce approaches that are radically different from what has been practiced for decades. Your own organization provides enough proof that doing more of the same is not working. The first companies courageous enough to make radical change see amazing results.