Leading Large Software Organizations

by Gary Gruver

Leading Large Software Organizations
(18 page Executive Summary)
Table of Contents

Intro:

What is the role of leadership in managing software projects? That is not an easy question because software differs from traditional projects. It also depends on whether small teams can work independently or the work needs to be coordinated across teams.

Why are software projects different? Jim Highsmith reminds us with 60 years of software development history in his book “The Wild West to Agile” [1] that software pioneers started by applying traditional project management approaches. Then, after years of attempting different versions of predictive methods, they discovered it wasn’t working as intended. While they did deliver large projects, improving on traditional approaches only led to increased frustration. Eventually, they realized the problem was, as Kent Beck pointed out, that software is a complex adaptive system. Software’s inherent complexity makes it hard to predict regardless of the level of rigor applied. But

[1] Highsmith, Jim. 2023. *Wild West to Agile: Adventures in Software Development Evolution and Revolution*. Addison-Wesley Professional[123]

software is adaptive and constantly evolving.. These unique software characteristics meant effective management would require a much more adaptive approach.

The pioneers came together and agreed on some guiding principles for this adaptive approach that became the Agile manifesto. They started working with small teams to develop techniques they could use to be more adaptive. As these changes started delivering better results, they worked on scaling these agile teams across larger, more complex organizations. The focus was on the teams, and the role of leadership was to empower these teams, remove roadblocks and get out of the way.

At HP, we took a very different approach to adaptively managing a very large (400-800 developers) complex (10 million lines of code rewrite) embedded software project that delivered 2X-3X improvements in productivity. Instead of focusing on how the teams worked, we essentially did Agile upside down with the leadership forming, for lack of a better term, the scrum team. This was a team of six to seven section managers, three technology leaders, a program manager and me. This team set the vision and direction for the organization. We also influenced how the complex system was evolving by setting the high-level objectives for the iterations, monitoring how the system was evolving, and then setting the priorities for the next iteration. We didn't do this by directive but in collaboration with the rest of the organization. Toward the end of each monthly iteration, we would create a draft of the priorities for the next iteration. This draft was then circulated for review and feedback in different management and technical forums for a week before coming back to my staff for finalization. It was all about aligning and agreeing on priorities, then spending time in the organization learning what was working, how the system was evolving and what was getting in the way so we knew what to prioritize next. If you had something on that list, you knew it was your top priority. If not, you would work on your team-level objectives. We weren't worried about how well we did Agile or DevOps. We just focused on adapting our processes and the product to deliver as much business value as possible.

The difference in our approach was that it didn't focus on the teams and how they worked. We let the teams choose how they wanted to work. Some chose to adapt agile processes, while some worked more traditionally. At the end of the three-year transformation, it delivered tremendous business value but, as we looked back, we didn't see dramatic differences in the effectiveness of the different teams. This led us to the conclusion that in a large, complex, tightly coupled system, the first order effect is the leadership team embracing an adaptive approach to management. The second order effect is how the teams work. This isn't to say you should stop all the current improvements at the team level. And with applications and architectures that enable small teams to work independently, empowering the teams with adaptive approaches and just removing roadblocks is appropriate. Instead, the lesson is that if you want to make dramatic improvements on large, complex software projects, you need the leadership team to play a much more active and adaptive role.

There are a few reasons it is important to have the leaders more deeply engaged. First, the leaders are much better positioned to understand the company's strategic direction and, thus, the vision for the organization, plus the objectives for large applications. An individual or a team could do this, but they are much more likely to focus on things within their sphere of influence. Second, the problems you can see and address when looking across teams are much more significant than the inefficiencies that exist within individual teams. These issues could be addressed by teams working with each other directly, but it is much more likely to happen if leaders are highlighting the issues and

getting the organization to work together on solutions. Third, leaders are better positioned to lead the more significant changes that have the biggest impact. Transforming how you develop software requires embracing new ways of working, and there will always be some level of resistance. For big changes, an engaged leadership team will be much more effective at influencing behaviors.

Since having the leadership team play a much more active and adaptive approach is very different from anything you are hearing in the industry. I will start with an example from HP. The goal of this example is not to recommend doing what we did, but instead show the power and types of changes best addressed by the leadership team. Then, I will review a more active role for leadership using basic common-sense principles so the approaches can be modified to address your unique situations.

HP example:
When I took on the enterprise embedded firmware group at HP in 2007, it had been the bottleneck for the business for over two decades. We had a hard time supporting defect fixes for released products because the code was in different branches based on when they were released. In new product development, we couldn't add a new printer to our plans or new features without checking with firmware, and usually the answer was no. The current architecture had been patched together over the years, starting with monochrome then adding color, advanced control panels and scanners. It was a tangled mess.

HP printing was growing at the time, so we tried spending our way out of the problem, but that wasn't working. While this situation was frustrating, it was tolerable until adding scanners required integration with enterprise email and security systems. We tried to write our own code for this integration, but we quickly discovered that we could not keep up with all the changes Microsoft was making.

At the time, HP was making a big investment into a new copier, and wanted to make sure firmware could fully support all the needs of this new product. The original team was separated into two groups. One to support the existing printer line on the old architecture and the other worked on advancing the firmware to support the new copier line. The manager before me led the release for the first new trial copier with the new firmware and realized it was not working very well. Before I arrived, they set the expectation that the firmware was going to need to be rewritten from the ground up. Additionally, instead of fighting to keep up with Microsoft, we were going to join them by writing the code to run on their Windows XP embedded (XPe) platform, which included integration into the enterprise systems. This rewrite would include all the current features, the new security integrations and would be done in time for the new copier release.

When I was hired, they were not looking for a different plan. Instead, they were clearly looking for someone to make the current plan happen. I understood this, but also realized we would need to add a couple more objectives if this effort was going to be successful. First, I knew the costs for XPe would be too high ever to be adopted by the printer line, so we needed to ensure the new architecture could also run on the cheaper Windows CE embedded (CE) platform. Second, we could no longer afford the complexity and costs of all the different code branches, so we needed to ensure we could support both released products and new products with the same branch.

When the leadership team came together, we had our objectives but lacked a vision. After a bit of work, we agreed our vision was: “To no longer be the bottleneck for the business and to free up capacity for innovation.” They also felt we should add an increased focus on test automation to our objectives. Efforts to include test automation were made in the past, but everything was mostly manual. The combination of these objectives and the vision drove our priorities over the next few years.

At this point, the traditional approach would have been to pull the development team into a few months-long detailed planning sessions. This typically led to a plan that showed we didn’t have enough resources to do all the “must-have” features. This would result in an additional couple of months of arguing with the business that we needed more resources and still couldn’t do everything. The leadership team had seen this play out in the past and didn’t feel it was very productive. We weren’t going to get more resources and the expectation was that if the new architecture didn’t at least match the current functionality, it wasn’t going to be accepted by the customers. Instead of focusing on creating an inaccurate plan given all the uncertainties, we decided to focus on making it happen. The program manager suggested we write down our top priorities for the month and then schedule a meeting at the end to see how we did and decide what to focus on next. This was a huge fundamental shift from anything that had ever been done before, but really set the foundation for the leadership team shifting from managing to a plan to taking a much more adaptive approach.

The first step in any big rewrite is agreeing on the architectural approach. This is typically led by a team of senior architects that would draw up diagrams to argue the pros and cons of different approaches, which someone else would implement. Our technical lead convinced us that was not a good idea. Yes, the goal of the first few iterations should be an architecture reference specification for each component. But instead of having a senior architecture team do this work, he approached each technical team (print, fax, UI, copy etc.) and said, “We are being given this opportunity to clean up our architecture to fulfill this vision and achieve these objectives.” Given that mission, what would they recommend changing and why? He would spend all day meeting with different teams and reviewing their recommendations and providing feedback. Each team came back once or twice a week to receive rapid feedback and adjust their plans. Frequently, these sessions led to quick agreements, but occasionally the alignment was more difficult. In these situations, instead of arguing the pros and cons of the different approaches and escalating to management for a decision, he would launch competing prototypes to see which would work better. It wasn’t someone’s opinion that drove the decision, but working code that demonstrated which approach was easiest to scale and had the best performance.

As a management team, we started monitoring how this complex system was evolving by tracking how well we were completing our monthly objectives and demonstrations of working code. The first few iterations focused on getting the tools and architectural specifications in place. Then, we shifted to reviewing working code. The first thing we focused on was the simplest thing we could do to prove out a thin slice of our new architecture, which scanned an image and sent it to a folder. The goal was to do a demo of this code in one of our iteration checkpoints. Then, over time, these demos expanded to print a job, copy an image (combining scan and print), copy an image through the control panel, etc. This is how we monitored how our complex system was coming together, by reviewing more and more complex thin slices through our architecture over time. We also quickly realized that if we were going to get people to embrace new ways of working, we needed to reinforce

these expectations in the checkpoints. We set the expectation that you couldn't demo any code that wasn't on trunk and we expected the demo not to be run by a person but an automated test. This was initially met with resistance, but after a few demos were canceled, everyone got the message and started embracing the new ways of working.

This approach was working well, and we could see the progress we were making with XPe, which was the primary objective. We also realized that if the system couldn't scale to CE, it wouldn't be a viable option for the broader business. So, about nine months in, the leadership team decided we needed to slow down and ensure everything could also run on CE. This took about a month and some significant architectural adjustments, but when we were done, we felt that our complex system was better evolving to meet the needs of the broader business.

One of the biggest cultural challenges we faced was getting people to accept the idea that we could support defect fixing for released products and all the new products under development on a common branch of the code. People would argue with the leadership team to convince them it would never work, but we held firm to that objective. We didn't know exactly *how* it was going to work, but we knew it was required if the platform was going to meet the needs of the business. Then, one day, a technical lead approached us with a solution. There would be one file that would document all the capabilities of each product (printer, scanner, stapler, paper sizes etc.) that all the code would reference to determine how to behave on a specific product. The codebase would be able to run and test all the capabilities on a simulator, but when deployed, it would configure at runtime to match the capabilities of the specific product. As usual, we monitored this with a prototype of working code and then prioritized converting everything.

We had a solution for working on trunk that met the needs of the business, and this worked well until we got into crunch time. The technical lead who created the solution realized that some of the teams had started taking shortcuts and breaking this capability. He approached us with this concern and recommended inserting code to break their functionality if it wasn't architected correctly. We agreed, and could quickly see the impact because teams would suddenly see their automated test passing rates drop. When we asked what happened, they would humbly admit they got caught taking shortcuts and they would prioritize getting it fixed. For a change this big to take hold, it took more than the vision and solution; it took monitoring the progress and a leadership team that understood they would have to work with the teams to ensure they understood the priorities, even if it required slowing down to do it right.

We were about 18 months along and making good progress, when the economy took a big downturn in 2008. The printer business went soft, and we had a new senior vice president who needed to dramatically reduce costs. There was a bakeoff between the new and old architectures that the team we were leading won, primarily because it scaled to CE, was architected for no branches and included lots of automated testing. The decision was made to kill the old architecture and not release another product until the new architecture was ready. I was given 30 days to combine the two teams and reduce my spending from $100 million per year to ~$55 million per year. This brought a whole new focus on understanding our cost drivers and eliminating the need for manual testers and replacing them with automation. It also required supporting a bunch of other products under development.

The leadership team was focused on how we were going to deliver on this expanded charter with half the resources. At this point, the architecture and approaches were clear–it was just a matter of getting everything done on time and managing the priorities across all the different products. We knew the developers were our bottleneck, so we spent time monitoring progress and trying to understand anything that was slowing them down. We watched the build success rate, test passing rates, feature development and completion of monthly objectives closely. When things weren't going as expected, we spent time in the organization talking to people to better understand the issues. As a leadership team, we were in a constant state of learning how the system was evolving and adjusting high-level priorities as appropriate.

About two years in, we started realizing we had stability issues with our automated testing due to our common test framework. When I asked the technical leads who reviewed the design, I got a bunch of people staring at their feet; it was just testing, after all. Instead of getting angry that we didn't have a solid grip on a foundational piece of our strategy, I just asked if we could find someone willing to do a deep dive and help us with this issue. We had a technical leader step up and develop the common test framework 2.0 to address the stability issues. Then, once again, we had to slow down to port all the existing test automation to the new framework. This was a hard tradeoff to make, given all the schedule pressure, but we knew if the developers were responding to the unstable tests, they were going to end up wasting a lot of time.

We were constantly looking across the teams focused on how things were going and what we could do to help the productivity of developers. We realized when the build was broken, it brought the progress to a standstill. This led us to create quality gates the code had to go through before getting to trunk so the codebase stayed stable. We also realized when developers broke the build, they went through some standard steps like looking for error logs and finding the test to reproduce the failure. We decided we could make that process easier by automating that work. When there was a build failure, everyone in the build would get a triage bundle email with a consolidated list of the error logs, a link to the automated test for easy reproduction and a chat room for discussion. We prioritized anything we could find to fix or automate to make the developers more productive because we knew we needed to optimize the throughput of the bottleneck.

The leadership team was constantly monitoring how the system was evolving to meet the needs of the business. We were tracking feature throughput to ensure we would have backward compatibility in place by release time. We also had lots of automated tests running all the time that we could use to measure the system's quality. We were developing features faster than we thought possible while keeping all the automated tests passing around 90%. It was working well, but for some reason I thought it would probably work even better if we focused on getting the passing rates up to 98% even though we were not close to a release. This turned out to be a big mistake on my part, because it caused our feature throughput to drop. We realized this quickly because we were monitoring how the system was evolving and could see that when we dropped the passing rate expectations down to 90%, the feature throughput came back up.

When we got ready to launch our first products, marketing realized that the architectural and process changes we made to reduce development costs had changed the value proposition for the printer line. The fact that released products and new products were on the same branch of code meant that firmware updates for released products would include all the features of the newest products. This

meant, as a purchasing manager, any product you bought today would be as easy to manage and have the same features as any new product you bought in the future. In essence, your purchase was future-proofed, so for the first time ever, the marketing team decided not just to market the printers but also the new "FutureSmart Firmware." This was a huge change that showed the value that strategic architectural decisions can have on both products and processes. In fact, this was so powerful, it is still part of the HP marketing messaging over a decade later. [2]

This is a brief list of some of the high-level improvements influenced by the leadership team to help the reader understand the power of having an active and adaptive approach to management. A more complete story can be found in my first book I co-wrote with members of the team "A Practical Approach to Large Scale Agile Development." [3] The reason I pulled these examples out of the book is that they are some of the changes that had the biggest impact. They are also the type of things that I can't ever imagine being fixed by empowering the teams and getting out of the way.

Understanding that we needed to scale to CE, architect for a common trunk and invest heavily in test automation required a good understanding of our cost structure and a more strategic view of the needs of the business than would typically occur at the team level. Influencing the prioritization of the common test framework 2.0, introducing quality gates and automating the triage bundles required looking across the teams to identify the biggest opportunities for improvement. Shifting to an iterative and adaptive approach to managing, developing the architectural plans with the teams and getting people to embrace trunk-based development were big organizational change management challenges that never could have happened from the bottom up. These were changes that had the biggest impacts and were best led by the leaders because of their unique perspective and influence.

The interesting part of this story is that, at the end of the journey, we were doing things we never thought possible. We were no longer the bottleneck for the business even though we cut our development costs by almost half. We were supporting 140% more products and increased our capacity for innovation by 8X. We were keeping 15,000 hours of automated testing passing at 90% on a rack of servers while bringing in 75,000 lines of code turmoil a day. If you told anyone on the leadership team this was possible in the beginning, we would have said that you were crazy. So how do you put together and execute a plan for something you don't believe is possible? It just doesn't work. Instead, it took a constant focus on the vision and objectives, plus an adaptive approach to learning and adjusting to discover what was possible. It also required working in collaboration with the entire organization to understand how the complex system was evolving so we knew what to prioritize next.

Leading large complex software projects

Hopefully, this example has convinced you that, for large complex software systems where lots of teams must work together, there is a powerful and active role for the leadership team. If you own deployment and have small applications teams that can be managed separately or large applications that have been architected so small teams can independently release their components, then empowering the team is a great approach. But let's be honest–not everything we do with software is

[2] https://www.hp.com/us-en/printers/futuresmart-firmware.html
[3] Gruver, Gary, Mike Young, and Pat Fulghum. 2012. *A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware*. Addison-Wesley

a mobile application, website or software-as-a-service. There are also lots of large applications where the deployment is done by someone else, so we need to create packages of updates for the customer that require coordinated releases, if nothing else. Software is used in applications like satellite systems, airplanes, automobiles and other things that require lots of teams working together to create complex systems. There are also many software applications that have yet to be rearchitected to enable small teams to work independently.

What do we do for these applications? Do we hire a bunch of Agile coaches to teach the teams to be adaptive and hope for the best? Do we set the expectation that we want everyone to do DevOps and start measuring their maturity on this path to force them to improve? Do we create and drive KPIs that we believe represent the behaviors we think we want? Or do we actively engage with the teams to understand how these complex systems are evolving so we can influence the direction to deliver the most business value?

I would argue it is the latter, but I most frequently see the former. Leaders believe that if they just hire a bunch of consultants to come in and teach their team(s) how to be adaptive, they will get the predictive results they expect. If we just figure out how to measure these teams correctly, we can get the right behavior. If they just push hard enough, the team will accomplish more. If we just get good at implementing this new complex framework, the business will get what they need. My experience is that, yes, you can take this approach, and that is what I frequently see. You can deliver large, complex software projects that way, just as the software pioneers did with predictive methods. However, I don't think you will deliver the types of breakthroughs we were able to accomplish at HP. That requires not just that the teams be adaptive, but everyone in the organization up through the leaders. The organization needs to learn and adjust to your unique challenges as a team. This starts with the leaders understanding the need to embrace an adaptive mindset if they are going to be effective managing a complex adaptive system. They need to clarify the vision and objectives so the goals are clear. It requires leveraging some common sense principles for adaptive management to help overcome the resistance to change and lead the transformation. Finally, it involves aligning on where to start and leveraging proven frameworks as appropriate.

Adaptive mindset for leaders

The first and hardest step is getting the leadership to understand that optimizing outcomes for large software projects requires an adaptive management approach. If the leaders aren't willing to embrace new ways of working and come together as a team with a common set of priorities, they shouldn't expect anything different than their current results. They need to move past seeing their roles solely as motivating and measuring how the teams are doing. Instead, they need to see their roles as part of the team trying to understand how this complex system is evolving so they can guide the priorities to optimize business outcomes.

This requires them to have a much better understanding of the current state, which can be hard because software development is difficult to see and measure. It is even harder if there is not a culture of trust. If the metrics have measured teams to challenge them to improve, then the metrics will tell them what they expect to hear. A classic example is when a senior leader went through my "Engineering the Digital Transformation White Belt Training" [4] with a group of managers. The leader

[4] https://bit.ly/EDT-Cert

had no idea people in the organization were dealing with so many issues, and was surprised that these issues never came up at checkpoints. As a leader, when you expect to hear good news and expect people to show you progress, that is what you are going to hear. If you really want to understand what is going on and what is possible, you need to seek a more detailed understanding and not get upset when you find issues.

Leaders have to change their perspective on the role of metrics. Instead of a tool to measure and motivate people to work harder/better, they need to think more about how they use metrics to monitor how the complex system is evolving so they can influence improvements to optimize outcomes. Or, as Deming says, "The role of leadership is to create systems and processes that enable employees to be successful." [5] This requires a shift in thinking from how we measure people and teams to how we monitor and improve this complex adaptive system so people can be successful. Instead of measuring based on accuracy of plan, we need to focus on discovering what is possible and improving those capabilities over time.

Setting the vision/direction

The leadership team needs to set out the vision and business objectives for the organization. Your customers, business partners and shareholders don't care how well you do Agile, DevOps, etc., so that shouldn't be your primary objective. And just because you do a really good job of adapting that complex development framework, you shouldn't expect business success. You're creating the software system and improving your processes for a reason. If you are not clear about those reasons and what you expect to accomplish, you shouldn't expect to see those results.

At HP, our vision was "to no longer be the bottleneck for the organization and to free up capacity for innovation," and our objectives were clear. We accomplished those things and delivered 2X-3X improvements in productivity because that drove everything we did. We were constantly monitoring and adjusting our plans to meet those goals. If you don't have your leadership aligned in a common direction using an adaptive approach, you shouldn't expect these types of breakthrough improvements.

Common Sense Principles

The hardest part of any large-scale transformation is getting the organization aligned and working together in a common direction. Since software is so hard to see, different people have different perspectives and ideas about the changes that will help the most. The first step in getting people aligned is working to gain a common understanding. This requires making the current state visible so people have a common perspective. The second step is having some simple common-sense principles for discussing the types of improvements that will help the most. The third step is agreeing on where to start. Getting people to invest in improvements and embrace new ways of working is one of the biggest challenges, in fact it is almost more important than where you start. We want to help reduce the resistance to change by letting people pick improvements they are comfortable with and believe will most help the business.

[5] Deming, W. Edwards. 2018. *Out of the Crisis*. The MIT Press. Cambridge, Mass.: MIT Press

Once there is agreement about where to start, leaders need to embrace an adaptive approach. This is a complex adaptive system, so it is hard to predict how long improvements will take or what issues will emerge as we start making changes. Therefore, the focus should be on making progress on priorities for the current iteration and learning as much as possible about how the system is evolving so we know what to prioritize next. We need a good understanding of how we are going to monitor the complex system so we maintain a common understanding. The focus needs to be on learning and adapting, which is why it is more important to align on where to start than starting in the right place. If, during the first iteration, you find other more important things getting in the way, you can always adjust the priorities based on what you are learning if you are aligned as a team.

The leadership team needs to make sure they are doing this transformation *with* the organization and not *to* them. Transparency is important, along with ensuring there are channels for feedback. At HP, we had monthly all-hands meetings where we reviewed progress and plans for each iteration. We also started with rough drafts of objectives for the next iteration and allowed a week for discussion and alignment to help include as many people as possible in the journey. The key is not just what you are doing but how you take the organization with you. Finally, the leadership team needs to quantify the impact of the improvements and market the benefits. It is easy to get bogged down in the day-to-day challenges and not take the time to appreciate the benefits of all the hard work. This is a mistake for two reasons. First, we miss an opportunity for the organization to build credibility with business partners that enable us to use capacity for further improvements. Second, we miss an opportunity for the team to feel good about how all their different improvements impact the overall system.

As we work to get the organization aligned and started on this continuous improvement journey, we need to understand that organizational change management is our biggest challenge. For this to work, we need to get people to invest in improvements and embrace new ways of working.

Overcoming their resistance to change requires learning from organizational change management experts like Jonah Berger. [6] He points out that the harder you push people to do something, the harder they resist. Instead of pushing harder, we should instead focus on overcoming their barriers to change.

One of the things I have found works well is moving away from telling people to do things like Agile, CI, CD, TDD, etc., especially when they have preconceived notions about what those mean and prepared arguments for why they won't work. Instead, I have moved to trying to explain the basic principles in very simple, common-sense terms that are harder to refute. This approach lowers the resistance to change, helps get the leaders aligned and can be used to explain the plans to the broader organization.

These common-sense principles start with understanding that software development and delivery contain different types of activities that require different approaches for improvement. First is determining what to develop. Second is determining the architectural approach for how the software is going to be developed. Third is the very creative process of working with the business to develop software that solves problems or adds unique value. Fourth are the repetitive tasks like builds, testing, creating environments and deploying code. Each of these different activities requires

[6] Berger, Jonah. 2020. *The Catalyst: How to Change Anyone's Mind*. Simon & Schuster

different approaches for improvement. In addition to looking at each of these different types of activities independently, software can leverage techniques refined over the years from manufacturing to look across these different activities to improve the flow of value to the customer. And lastly, the adaptive aspect of software enables some unique advantages for managing uncertainty that also merit consideration.

The following sections will review each of these types of activities in more detail. There will be ideas for making those activities more visible to help everyone align on a common understanding of the current state of the system. It will include principles for improving the activity, ideas for monitoring progress and approaches for quantifying the impact to help market the improvements. The goal is to provide some basic principles the leadership team can leverage to address their unique challenges.

What to develop

The first step in any software development program is determining what to create. This can be done through a business plan, working with marketing, the business and/or customers. The output is usually a list of requirements that need to be prioritized for development. The priorities are based on a combination of how hard it will be to implement and how much business value it will create. This process can have a few challenges. First, with complex systems like software, it is hard to tell how the customers will use it and if it will deliver the expected value. Second, software organizations tend to overcommit which sets the business up for disappointment and teams for burnout. Third, since it is usually easier to come up with ideas for software than to implement them, organizations frequently end up with a large backlog of ideas that start taking on a life of their own.

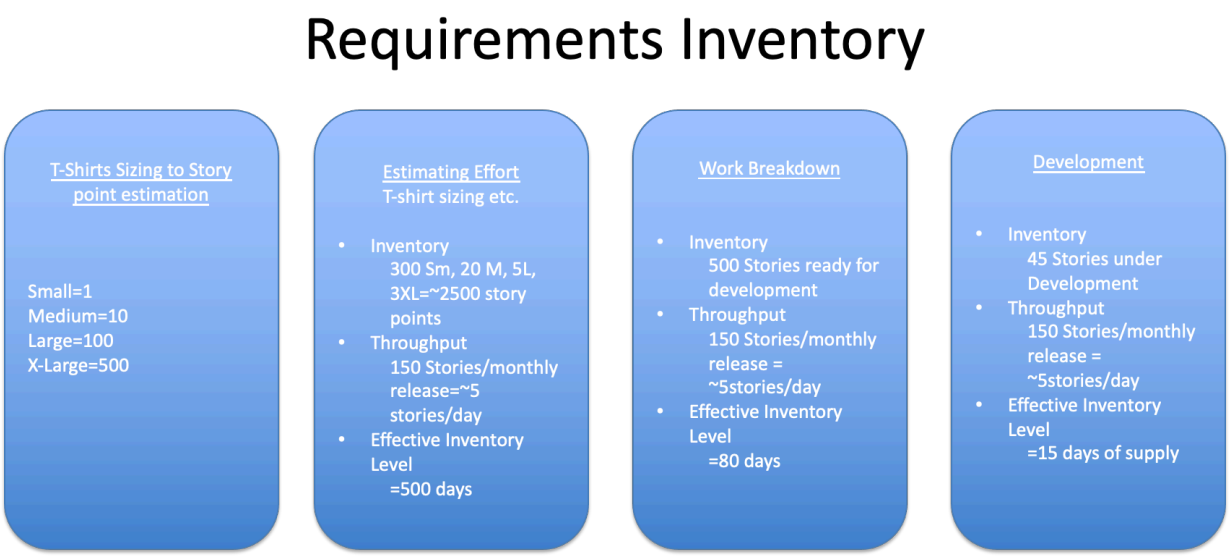
One of the biggest sources of waste in software is developing new features that are not used or don't deliver business improvements. This issue can be made visible by measuring feature usage and outcomes. As Jez Humble, Joanne Molesky and Barry O'Reilly point out in the book "Lean Enterprise," [7] every new idea should be treated as a hypothesis that needs to be validated as quickly as possible. This problem is addressed ideally by getting working software in front of users–or, if that is not possible, then representatives of users–as quickly as possible for feedback and impact monitoring. The faster and better the feedback, the less time wasted on development that doesn't add value. Marketing the impact of improvements in this area will require documenting work saved by not completing and supporting features that were not fruitful. There is also the potential to show how the feature was improved based on the feedback so that it did deliver positive outcomes.

Software organizations tend to overcommit and under-deliver. This sets the business up with unrealistic expectations, which causes them to lose confidence in the development teams. It also overwhelms the development team, leading to burnout and the accumulation of technical debt. This push for overcommitment can come from anywhere in the organization down to the team level as this 10-year-plus example [8] from a senior program manager officer over hundreds of developers demonstrates. Visibility of this overcommitment is characterized by how often the business' expectations are missed and how often it requires heroic efforts from the development teams to deliver the bare minimum. Leaders are best positioned to address this issue. As the example shows,

[7] Humble, J., Molesky, J., & O'Reilly, B. (2015). *Lean Enterprise: How High Performance Organizations Innovate at Scale*. O'Reilly Media
[8] https://bit.ly/flowwhitepaper

they need to create metrics for monitoring the historical capacity of the organization. Not as a metric for measuring team's productivity–because then it will be gamed like most software metrics–but as a measure of historical capacity to ensure expectations are based on proven capabilities versus hopeful wishes. The leaders are also best positioned to help the organization understand that the most effective way to get more business value from the organization is not driving overcommitment and thrash. Instead, it requires a focus on removing waste and inefficiencies to change the capabilities of the complex system, like we did at HP: Quantifying the impacts of improvements by monitoring the delivery of business expectations and sustainability of work for the development teams.

Since it is usually easier to come up with ideas for software than to implement them, organizations frequently end up with a large backlog of ideas that start taking on a life of their own. This list of requirements needs estimates so the business can do a good job of prioritization. This requires pulling developers away from creating new code to doing estimates. These estimates range from quick and easy ballpark numbers to more detailed work that requires extra effort. The more accurate these estimates must be, the more work they require. The first step to making this process visible is documenting how many requirements are in each state of progress, from the name of an idea to delivery to the customer. Then, the next step is converting this metric to days of supply, as shown in the graphic below. This gives everyone in the organization a common understanding of just how much requirements work is in progress relative to what the organization can accomplish. A classic example is when we mapped this out at a client and realized they had a nine-year backlog. And while this was bad, the worst part is they were just about to pull all the team off development for a three-week sizing exercise for the next set of features. To reduce this type of waste, you should minimize the amount of effort invested and shift it to occur as late as possible by limiting the number of requirements in definition and estimation to only what is required to support business decisions or development. This frees up the developers to create more business value and minimizes the overhead of managing these big lists. The waste removed by this approach can be quantified by the reduction in requirement work for features that never get developed and the reduction in overhead managing these priorities.



Architecture

As the HP example showed, the architectural approach can dramatically impact the efficiency of the development processes and the value of the product. The current or planned architecture can either be a constraint or an enabler. Left to the architects, it will probably work but might not be optimized to deliver as much business value as possible. Gaining a common understanding requires not just an architectural map, but visibility into the implications of that architecture on the product and the development processes. Does it require lots of different teams working together closely? Are there clean interfaces where it is easy to test separate components? Is it tightly coupled to the hardware, which makes it difficult to leverage capacity in the cloud? For embedded systems, does it require the duplicate work associated with branching to support all the different product variability? Is it going to have the performance and security the business requires? Is it going to be hard to update? What business capabilities does it need to enable? We need to make the implications of the current architecture visible to gain a common understanding of the constraints of the current system.

Once you have a good understanding of the constraints of the current architecture, you need to decide if that impact on the business warrants the risks associated with the architectural changes. At HP, I didn't have a choice. The job was to rewrite 10 million lines of code from the ground up. It was the hardest thing I have ever done. It put the business at risk, because we couldn't ship another product until it was complete, and it set the development team up for a death march. Sometimes, a complete rewrite is required to meet the needs of the business–maybe it was true that time, but if possible, I would work to avoid a situation like that at all costs. If the current architecture puts unacceptable constraints on the business, is it possible to take an evolutionary approach to improving it by leveraging approaches developed by Martin Fowler? [9] If it is, I would strongly recommend that approach.

Whether you are creating a new software system, doing a rewrite, or taking an evolutionary approach to improving the architecture, it can be aided with a few common sense principles. First, the leadership team needs to be clear on the objectives for the application. If we hadn't stated that it needed to support trunk-based development for all products and scale to CE, those things wouldn't have happened. Second, we need to reduce the disconnect between the architectural vision and implementation. This can be done by involving the development teams in defining the architectural approaches. It can also be helped by resetting the expectations we have for our senior technical people. For the HP example, I intentionally used the words 'technical lead' instead of 'architect' because I believe just creating diagrams is of limited value. Instead, we need technical leaders to work with the teams to not just define direction but also create design patterns of working code the teams can use as a model as they build out their components. Also, early on, we need to be willing to prototype different approaches to figure out which will work best. We can talk about an approach all we want, but at some point, we start learning more when we start writing code. Third, we need to move past designs to working code with rapid iterations and feedback as quickly as possible. The architecture shouldn't be created based on plans, then integrated to see how the plans worked. Instead, we should put the end-to-end system of working code together early and often starting with thin slices and monitor how the complex system is evolving. Fourth, we need to monitor the system

[9] https://martinfowler.com

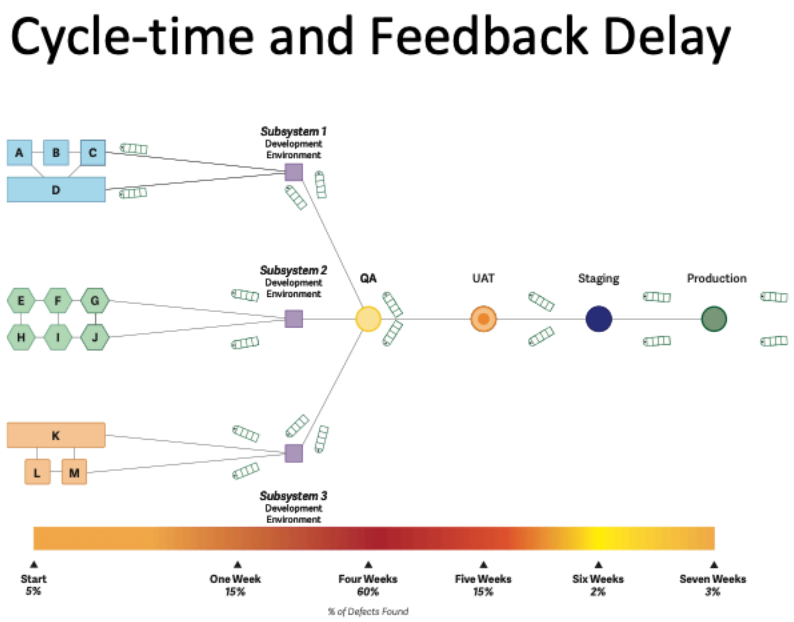
to ensure people aren't taking shortcuts and breaking capabilities the architecture was designed to enable.

Marketing the improvements of the architectural changes can be a challenge, especially when working with the non-technical people in the business. For this, we need to do more than describe the current and future architecture. Instead, we need to focus more on the business implications of the current architecture and what is enabled by the improvements.

Improving code writing

One of the biggest changes that has helped the creative process of writing software to address business problems is shifting to building in quality. Manufacturing learned a long time ago that the only way to be successful was to focus on quality. They did this by controlling the process, because the process was creating the same product repeatedly. Software is different in that it involves a person working with the business to create new and different solutions each time. Instead of controlling the process, we need to work on giving that person the best possible feedback on the intended and unintended consequences of their changes and how those work with the rest of the system. The quicker we can provide this feedback, the more help it is to the developer in terms of improving and limiting waste associated with work going down the wrong path.

The magnitude of this issue can be made visible by mapping the feedback delay. This can be done by mapping all the different environments the code has to go through on the way to production. Then, document the cycle time–or how long a code change takes to move through each of the different environments. On top of this, document what percentage of defects are found in each environment and you have the feedback delay, as shown in the example below.



To reduce the delay, we need to move testing closer to the developer with automation and/or accelerating the progression of code toward the end user or a representative of that user. The more we can work in smaller batch sizes, the easier it is going to be to localize the feedback to the right developer. The slower this feedback is–due to batching up changes for testing and infrequent releases or by isolating different batches of changes from each other with feature branches–the harder it is for developers to improve. There are lots of straightforward changes we can make to improve, and it is easy to update the feedback delay maps to show the benefits of those improvements.

Improving repetitive Tasks

Repetitive tasks like doing a build, creating an environment, deploying code and testing are different. As in manufacturing, they can be automated to reduce costs and ensure consistency. Then, like any process, they can be continually improved over time. This automation provides several benefits. First, it reduces the cost of manually doing the same thing repeatedly. Second, it provides consistency that keeps developers from being distracted by problems not associated with their code changes. Third, it enables working in small batch sizes that are much easier to debug and triage.

It is straightforward to provide visibility to the impact of manually implementing repetitive tasks. How much does it cost? How long does it take? How much time do people waste waiting? How many problems are associated with manual errors? How big are the batch sizes? How much effort does it require to debug and triage these large batch sizes?

Automating and improving these repetitive tasks can eliminate a lot of waste that is slowing down the organization. The first step is automation. The second step is ensuring the automation is repeatable, which is especially important for testing. Flaky test automation is going to cause more problems than it solves. Therefore, we need to be constantly monitoring and improving the stability of automation.

Optimizing flow

There are opportunities to remove waste and become more efficient with each of the types of work we have described so far. There are also opportunities to look across the entire system to identify opportunities for improvement. For optimizing the overall system, it makes sense to leverage Goldratt's theory of constraints from manufacturing. He taught us that the more work we have in process, the harder it is to see what is going on and the more waste we have in the system. And just because we have work going on doesn't mean we are getting any more work done if it doesn't make it through the bottleneck to customers. Instead, to optimize the flow to the customer, we need to understand the source of the bottleneck, make sure it is never starved for work, do everything we can to optimize its flow and limit the work in process.

For software, the bottleneck can be in the contracting, requirements, development or release process. It can happen in the contracting process but that is usually limited to government work which explains why they focus so much there. It is possible, but doesn't happen very often in the requirements process, because it is much easier to come up with ideas than it is to implement them. It is usually in either development or release. The first step to improving flow is gaining a common understanding of the bottleneck. This can be done through discussions with experienced leaders. If

not, mapping the requirements should help, but keep in mind that, as this whitepaper discusses, value stream mapping needs to be adjusted a bit to work well for software. [10]

If the bottleneck is in the release process, we should look to automate the repetitive tasks so we can work in smaller batch sizes and release more often. If it is with developers–which is where we want it since that is where the value is being created–we want to do everything we can to make them more productive. The HP example has some ideas that might help. The view of different types of work discussed above has ideas for removing waste that can impact developers. There are also new ideas, like leveraging AI. The key is working to understand how developers are working and doing everything possible to help them become more productive. We also want to ensure there is always a prioritized buffer of work ready for development the team can pull from so they are never starved for work. Then, we should limit the work in process by releasing work into the system at the rate it is being consumed by the developers.

Marketing the improvements in flow is hard to do since every time you do something it is a different size and shape, plus new work that has never been done before. The industry has been trying for decades to measure developer productivity, but for the most part has concluded it isn't possible. Additionally, if you try to force a measure and hold people accountable, it is too easy to game the metric to show the improvement you are demanding. Instead, I have found it much easier to market the improvements by quantifying the removal of waste that is slowing down the organization for each of the types of work described above and these metrics are less likely to be gamed.[11]

Flexibility to address uncertainty:

The unpredictable nature of software creates challenges for businesses. They need to deliver business results, but that becomes difficult with unpredictable components. This is even more true for embedded software that must be released in physical products that are tied to the start of manufacturing. This is where the adaptable nature of software can provide advantages. With software, if you are building in quality for a prioritized set of capabilities, then you can always release the features that are ready on the date required and add more later with updates. This ability to use the feature knob to manage uncertainty works well if you are not managing too tight a schedule.

Ideally, when you get to the release date, all the mandatory features are done and you are working on a prioritized list of ideas for further optimization. If that is the case, you can use the adaptable nature of software to address the uncertainty of a complex system. If that is not the case, then you need to realize you are putting the business at risk by trying to manage a tight schedule with a component that has a high degree of uncertainty.

Getting Started:

Getting the leadership team aligned and started on a continuous improvement journey can be challenging. Everyone has a different perspective on the issues and what needs to change. People frequently see the need for the organization to change but don't want to change themselves. How do we overcome this resistance? It starts with learning from organizational change management

[10] https://bit.ly/VSMWP
[11] Gruver, Gary. 2020. *Engineering the Digital Transformation*. Cambridge, Mass.: MIT Press

experts like Jonah Berger. As we discussed earlier, the harder you push for change, the harder people resist. Instead, focus on overcoming resistance–his book is a great source for helping people with ideas. [12] For busy executives who don't have time to read the book, I have summarized the key concepts and their relevance to software in this whitepaper. [13] A big part is letting people pick the ideas they believe are practical and will provide the biggest benefits to the business. Working one iteration at a time with opportunities to adjust the approaches based on what we are learning also helps to reduce the risk and thus resistance to change.

The first step is agreeing on where to start. The previous section provided basic principles for discussing different options with links for deep dives for those wanting more detail. Start with the vision, then make things visible so everyone has a common understanding of the current state and goals. You can't do everything at once, so start where the team has the most passion. I would recommend starting with making sure everyone has a common understanding of the implications of the current architecture because it can be a constraint or an enabler. Not that I recommend starting with architecture improvements; I don't. It is just that those constraints provide a good framework for discussing the changes that will provide the biggest bang for the buck.

Does the leadership team believe that evolving to an adaptive approach is key to achieving the vision? Discuss where the team feels they need better visibility into the different types of work to gain a common perspective. What are the improvements they believe provide the best opportunities? What are the types of things best addressed by the leadership team because of their unique perspective and influence? These discussions will probably lead to a long list of things that should be fixed, but we can't do everything at once. As we start making changes, we learn a lot that needs to influence our priorities as we take an adaptive approach. Keep the list of all the ideas, but prioritize where you want to start. Then, break that down into things everyone believes you can and should be able to accomplish during the first iteration. Discuss what you are going to monitor to track how things are evolving. Spend time in the organization learning what is working and what is getting in the way. Schedule a checkpoint at the end of the iteration to review what was accomplished, what was learned and what the priorities should be for the next iteration. Be sure to allow time for feedback on those plans because it is important to bring as many people along on the journey as possible. Provide transparency on progress and plans with things like team meetings. And don't forget to take the time to quantify and market the improvements.

Leveraging proven frameworks

There are a lot of frameworks in software that have proven successful. They were designed to address a lot of the problems highlighted in the common sense principles. I went out of my way not to discuss them or use any terminology that would point to them, not because I think they are bad ideas but because when people hear them, they have preconceived notions. Plus, when you break them down to basic principles, it is harder to refute and easier to gain alignment across the organization.

However, adopting a framework should never be the goal. Start with the vision and objectives for the business. Use the basic principles to align on what you want to improve. Then, see if there are

[12] Berger, Jonah. 2020. *The Catalyst: How to Change Anyone's Mind*. Simon & Schuster
[13] https://bit.ly/ChangMgt

proven approaches that would help. As you start implementing those approaches, monitor how they are helping to address those issues–not how well you are adopting the framework.

Summary:

The conventional wisdom in the software industry is that the best role for leadership is removing roadblocks and getting out of the way. This works well for small applications or large applications that are architected so small teams can work independently. It can also work for delivering larger, more complex systems–I just don't think it is the best approach.

Breakthroughs for these systems require much more engaged and adaptive leadership. The problem is that leaders frequently don't see their role as helping to adapt complex software systems and creating processes where employees can be successful. Instead, they see themselves in a role outside the team in which they measure and motivate the team to commit to as much as possible. It is no surprise that developers prefer the leaders to empower them and get out of the way! Given that approach, I can see why they feel that way. And while that would be nice, I don't think it would have delivered the types of breakthroughs we achieved at HP–and it won't deliver what your business deserves.

My goal is to help leaders understand that there is an important and much more helpful role for them to play–not by telling them what to do but by giving them a framework for discussions to gain alignment. The HP example showed the impact and types of improvements an engaged leadership team can provide. Leadership is best positioned to ensure strategic alignment with the business, look at cross-team inefficiencies and influence big changes. How does your leadership team see their opportunities from that perspective? Where can they provide the most value?

The basic principles were provided to help the leadership team align on the types of improvements that will address your unique challenges. I worked to provide common sense principles using basic logic that is hard to refute. I also tried to rise to Goldratt's challenge to explain it simply–and if you can't, you probably don't understand it well enough yet. Does your leadership team understand the current state of your complex system and how it needs to evolve to optimize business outcomes well enough to explain it simply? Are they ready to engage in an adaptive approach to working with the teams to discover what is possible? Or would they rather hire consultants to roll out a complex framework and hope it helps?