Leveraging Wiring the Winning Organization for Organizational Transformation

Ron Wilson

Email address: ronbwilsonii@gmail.com

1. Abstract

Problem: Organizations often have processes in place, but they are either not followed or not regularly reviewed and updated. This often causes process issues which lead to frustration, inconsistency and deliverable dates getting missed.

Solution: Leverage the principles found in Wiring the Winning Organization to change how the organization operates to achieve success. Success for each organization will look a bit different since there isn't a one size fits all model. Organizational transformation will not occur all at once. It is a continuous journey with a combination of small and large changes. It will require champions at all levels to help drive change. Each process must be analyzed to determine how it can be most efficient and those processes must be defined and understood by those who are using them in order to make them repeatable. When all the processes are put together, the organization will achieve the objectives it was created for.

2. Biography

Ron Wilson is the Software QA Manager at Powin. In his current role Ron manages a remote workforce of software QA engineers across the globe. Ron has over 20 years of experience leading globally distributed quality assurance teams across a variety of industries. He believes in leading people by example with his hands on leadership style. He has an exceptional track record of building software quality organizations from the ground up and is an advocate of developing standard processes to achieve consistent results. Ron leverages metrics to drive organizational transformation and influence change.

Ron has participated in various testing conferences. He has a software testing blog at www.qarevolution.com and a YouTube channel called QA Revolution.

1 Introduction

Several months ago, I found *Wiring the Winning Organization* (Kim & Spear, 2023) and recommended that my company, Powin, read it to see if we could use some of the principles within the book to improve our processes. Powin is a global leader in the energy storage business. There are several products at Powin which store energy in large batteries to help utility companies take advantage of wind and solar power. As a fun fact, Powin recently powered the Super Bowl in Arizona using renewable energy. This technology is constantly evolving and requires robust testing to ensure the platform is safe and reliable.

Powin grew rapidly and as most startups go, the work had to be done quickly to meet customer priorities and there were limited resources to get the product out the door. Priorities were constantly changing, creating a series of start and stop features, which was further slowing work down. This led to a lot of frustration among the engineering team. In addition, for the most senior members of the organization, they were being torn between supporting production issues and new feature development work which carried the same equal importance in terms of priority.

In my current role, I lead a remote workforce of software QA Engineers across the globe. The software QA Engineers are distributed across several Agile teams. Originally the team had only a few members and they worked together to use best practices based upon their previous experience but it wasn't formally defined and there was a lack of consistency in terms of how things were accomplished. There was a need to standardize the testing processes so they were consistent across the Agile teams.

If we were going to have a chance at winning, something had to change. Change is the only constant in organizations moving forward, and we must do everything we can to keep adjusting to meet the company expectation of delivering software on time with high quality. In order to meet this expectation processes must be analyzed in further detail both at an individual level and collectively as a whole. This paper is a deeper dive into the recommended strategies to build a winning organization to determine which ones will help the organization you work for win. It also includes several case studies that help see those changes in action and will cover Powin's changes to make our organization compete and win daily.

2 Organizations

We create organizations for a variety of reasons, but certainly one of them is to accomplish seemingly common but actually audacious undertakings that one person cannot do alone. The goal may be as ambitious as sending a man to the moon or as common as providing a commercial product or service such as running a restaurant, bakery, or hospital (Kim & Spear, 2023, 7). Organizations can be small, with only a few people, or large across multiple continents. Every organization should have a mission or goal. It is important that each individual working within an organization understand its purpose.

It has been proposed that organizations gain competitive advantages largely by seizing opportunities that are unavailable to others. This concept, led by Dr. Michael Porter's "five forces" from his book *Competitive Strategy*, asserts that organizations enjoy unfair returns by having made the playing field unlevel: by locking in customers (so they cannot consider alternative vendors), preventing suppliers from finding other outlets for their wares or barring individuals from offering competitive products and services (Kim & Spear, 2023, 8). Organizations compete against each other and in many situations they are targeting the same customers, using similar materials, and working under the same regulations.

Organizations that create great products can dominate the industry for many years. Here are a few organizations that have beaten their competition significantly.

> Toyota has led in design and production in the auto industry for over 50 years. Despite being woefully uncompetitive in the late 1950s, it gained advantages through superior quality and productivity. It built on those leads by cutting in half the time required for major model upgrades, by cutting from weeks to minutes the time to convert plants from one model year to the next, and

> by being incredibly fast to introduce whole products and invent whole new technologies (Kim & Spear, 2023, 9).
>
> In 2007, Apple released the groundbreaking iPhone, with only dozens of software developers creating its applications and user interface libraries. The resulting product redefined what consumers expect from mobile devices (Kim & Spear, 2023, 9).

Here are a few examples where established organizations have decided to make changes in their organizational wiring to become more competitive.

> In 2002, Amazon struggled to upgrade its e-commerce software, able to make only twenty software changes per year because of the high risk of outages and the difficulty of coordinating across hundreds or thousands of software engineers. In 2014, however, Amazon was making some 136,000 deployments every day, quickly and safely (Kim & Spear, 2023, 10).
>
> In manufacturing microprocessors, the differences between the leaders and the rest are huge in terms of throughput times, quality, yield, and sustained product variety within a single plate. What's encouraging is that such performance is replicable. One plant cut its throughput times by two-thirds, increased yield, reduced scrap, and otherwise made it far easier for engineers and technicians to use the sophisticated capital equipment they had (Kim & Spear, 2023, 10).

These are just a few examples of organizations which have been able to make changes in processes which have resulted in significant increases in quality, reliability, and profit. These changes did not happen overnight but took time. Amazon for example, took 12 years to get to the point where they were able to scale their deployments with quality. I am confident throughout that timeframe that there were many failures which caused them to pause and reassess before being able to move in a positive direction.

2.1 The Three Layers of Value

Organizations work together to accomplish a series of goals. To achieve outcomes, they must collaborate and work together to solve problems. Collaboration isn't optional but is something that is required for all organizations and often one of the greatest obstacles in achieving success. While each part of the organization is working towards the ultimate goal there are often competing priorities between one department and another. For example, if the production support team needs the expertise of a team member who is working on new feature development to solve a difficult defect, the team member will need to pause the new feature development work to help the other team with a production defect.

In the example above, the software engineer needed to solve a complex defect. Let's take a look at the problem solving techniques available which will help to solve the problem.

- **Layer 1:** contains the technical objects. These include technical, scientific, and engineering objects that people are trying to analyze, create or modify (Kim & Spear, 2023, 13). The software engineer will be analyzing and modifying software code which is technical in nature.
- **Layer 2:** contains the tools and instrumentation. These are the scientific, technical, or engineered tools and instrumentation through which people work on Layer 1 objects. The software engineer will need to use the following technical tools which include Jira, Confluence, AWS, Bitbucket, Splunk, and Eclipse and to help solve the production defect (Kim & Spear, 2023, 14).
- **Layer 3:** contains social circuitry. This is the overlay of processes, procedures, norms, and routines, the means by which individual efforts are expressed and integrated through collaboration toward a common purpose. This is the "socio" part of a socio technical system.Processes and collaboration with other members of the organization occur. (Kim & Spear, 2023, 14). The software engineer will

need to use the established processes of the organization and make sure those are followed. One part of the software development process is to have a peer perform a code review to ensure there aren't formatting or potentially other defects in the code before it is merged into the build, retested, and deployed into production.

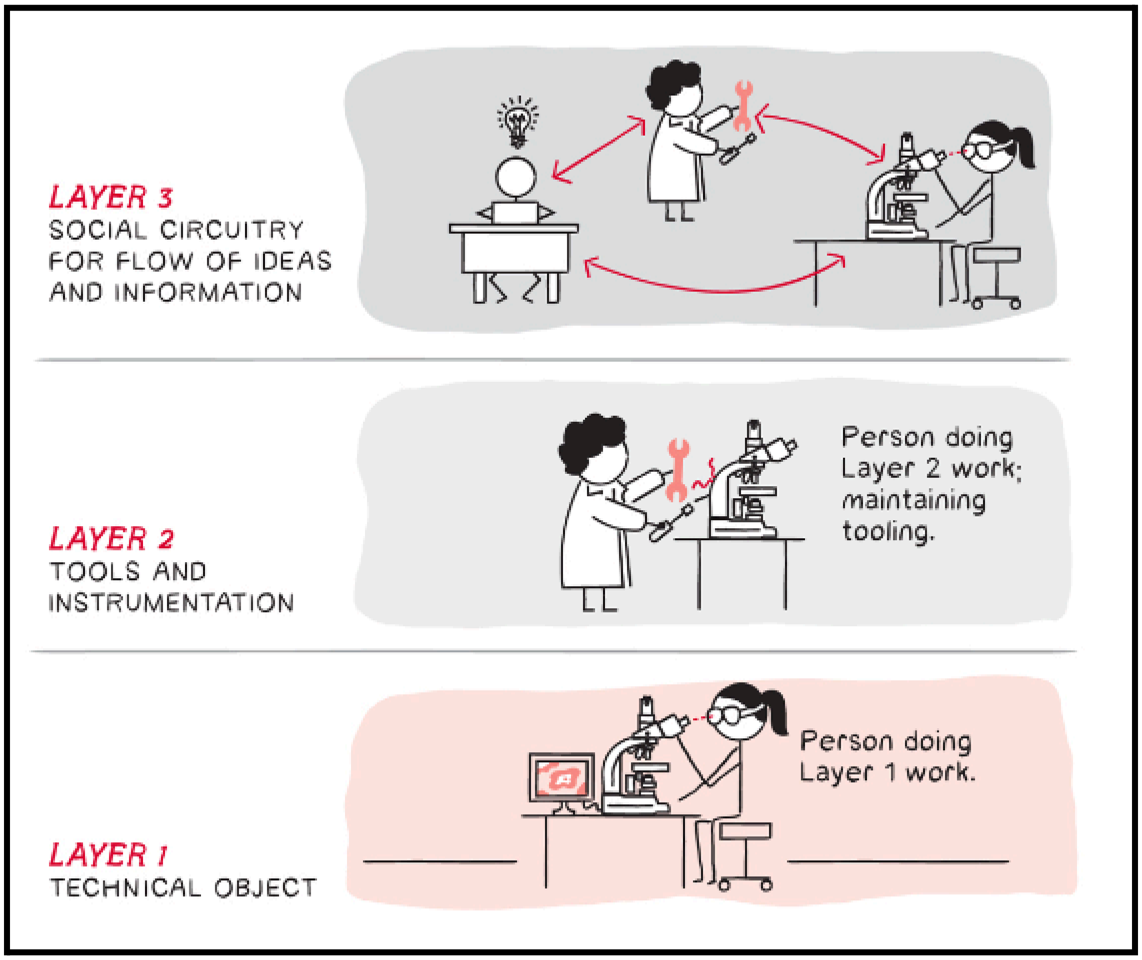

LAYER 3
SOCIAL CIRCUITRY FOR FLOW OF IDEAS AND INFORMATION
LAYER 2
TOOLS AND INSTRUMENTATION
Person doing Layer 2 work; maintaining tooling.
LAYER 1
TECHNICAL OBJECT
Person doing Layer 1 work.

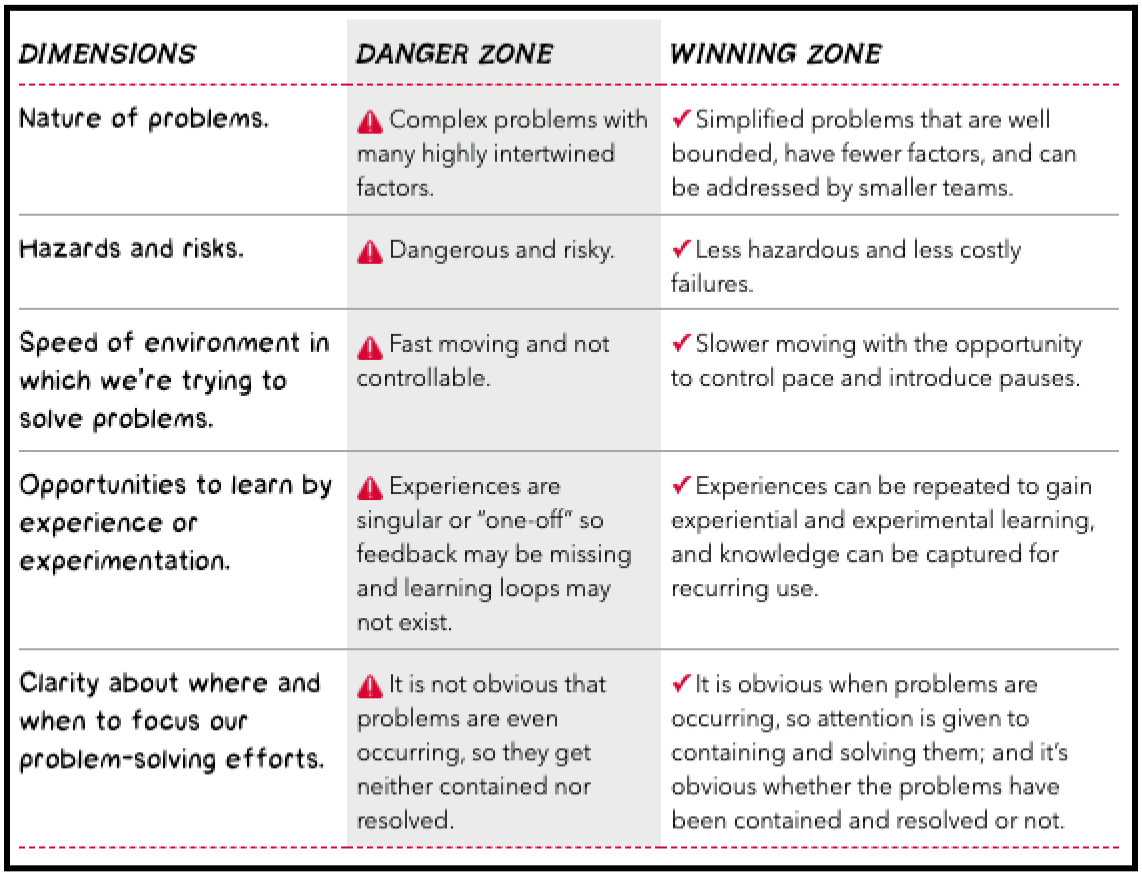
Figure 1: The Three Layers (Kim & Spear, 2023, 16)

These three layers help redesign problems to make them more accessible and faster. In Layers 1 and 2, people can focus on solving problems. The software engineer resolved a technical defect in the code by making the appropriate code changes using the software tools available. If the software engineer doesn't have the proper information or the right tools to help solve the problem effectively, then the software engineer is going to have to spend a lot of extra time working within Layer 3. This will be especially difficult if the organization doesn't have effective processes as it relates to things such as detailed requirements, code reviews, and branching strategies. The software engineer will have to over-compensate and ultimately not deliver the correct resolution to the defect with high quality in a timely manner. This will cause frustration for both the software engineer and the customer.

2.2 Danger Zones versus Winning Zones

Leaders must manage complex problems at Layer 3. This helps determine if the organization can achieve terrible or great outcomes. This social circuitry makes resolving problems or generating new ideas easy or difficult. In the danger zone, conditions make solving problems or developing new ideas difficult. In the winning zone, conditions are more straightforward for solving or generating new ideas.

Here are some examples of the *Danger Zone* versus *Winning Zone*.



DIMENSIONS	DANGER ZONE	WINNING ZONE
Nature of problems.	⚠ Complex problems with many highly intertwined factors.	✓ Simplified problems that are well bounded, have fewer factors, and can be addressed by smaller teams.
Hazards and risks.	⚠ Dangerous and risky.	✓ Less hazardous and less costly failures.
Speed of environment in which we're trying to solve problems.	⚠ Fast moving and not controllable.	✓ Slower moving with the opportunity to control pace and introduce pauses.
Opportunities to learn by experience or experimentation.	⚠ Experiences are singular or "one-off" so feedback may be missing and learning loops may not exist.	✓ Experiences can be repeated to gain experiential and experimental learning, and knowledge can be captured for recurring use.
Clarity about where and when to focus our problem-solving efforts.	⚠ It is not obvious that problems are even occurring, so they get neither contained nor resolved.	✓ It is obvious when problems are occurring, so attention is given to containing and solving them; and it's obvious whether the problems have been contained and resolved or not.

Figure 2: *Danger Zone* vs. *Winning Zone* (Kim & Spear, 2023, 16)

In the *danger zone*, problems are complex, with many factors affecting the system at once, and their relationships are highly intertwined. Hazards are many and severe, risks of failure are high, and costs of failure can be catastrophic. Systems in the *danger zone* are difficult to control, and there are limited, if any, opportunities to repeat experiences, so feedback-based learning is difficult if not outright impossible. These problems are often intertwined, which makes it extremely difficult to manage. The risk of failure is high, and the cost of failure can be catastrophic (Kim & Spear, 2023, 17). At Powin, we struggled to move from a start-up culture with little documentation and standardized processes. Knowledge was limited to a select few and not distributed across the organization. Those key resources were constantly overworked and burnt out and the organization was stuck based upon the capacity of a few key players. Priority was getting the software development done and not following processes or sharing knowledge. Without a deliberate change, the same cycle would continue.

In contrast, leaders enable much more advantageous conditions in the *winning zone*. Problems have been reframed so they are simpler to address. The hazards and risks have been reduced so failures are less costly, especially during design, development, testing, and practice. Problem-solving has been shifted into slower-moving situations, where the pace of experiences can be better controlled. Opportunities to learn by experience or experimentation are increased to allow for more iteration. And finally, there is more clarity about where and when to focus problem-solving efforts, because it is obvious when problems are occurring, so attention is given to containing and solving them (Kim & Spear, 2023, 17). At Powin, we made a deliberate attempt to break the cycle. Sure, we still have issues and sometimes revert back to hold habits, but we are much more consistent in terms of our process. Instead of pushing through features to meet the schedule, we put things on hold, get the problems resolved, and then resume feature development. This helps to release a feature with higher quality versus delivering one that will leave our customers unhappy.

2.3 3 Ways to Solve Problems

Three mechanisms are available to solve problems moving from the *danger zone* to the *winning zone*.

1. **Slowification:** makes it easier to solve problems by pulling problem solving out of the fast-paced and often unforgiving realm of performance. Instead, solve problems this in the more controllable, forgiving, lower-cost, less-demanding, and repeatable realms of planning and practice. This shifting of Layer 3 problem-solving into planning and practice allows people to engage in iterative, reflective,, experientially, and experimentally-informed reasoning rather than having to constantly react with whatever habits, routines, and legacy approaches have already been ingrained (Kim & Spear, 2023, 18)
2. **Simplification:** makes the problems themselves easier to solve by reshaping them. Large problems are deliberately broken down into smaller, simpler ones through a combination of three techniques: incrementalization, modularization, and linearization. By doing so, we partition complex problems with many interacting factors into many smaller problems. These problems have fewer interacting factors, making them easier to solve. Furthermore, Layer 1 (technical object) problem-solving can be done in parallel, with less need for Layer 3 coordination, increasing independence of action (Kim & Spear, 2023, 18)
3. **Amplification:** makes it obvious there are problems, and makes it clear whether those problems have been seen and solved. Mechanisms are built into Layer 3 (social circuitry) to amplify that little things are amiss, drawing attention to them early and often. This focuses attention on containing and resolving small and local glitches before they have a chance to become large and systematically disruptive (Kim & Spear, 2023, 18)

The organization can leverage using one or all of these options for each problem—slow things down to make things easier to solve. Break down significant problems into smaller ones. Amplify when there is a glaring problem that needs to be solved. These techniques will help ensure success and move the organization to optimal performance.

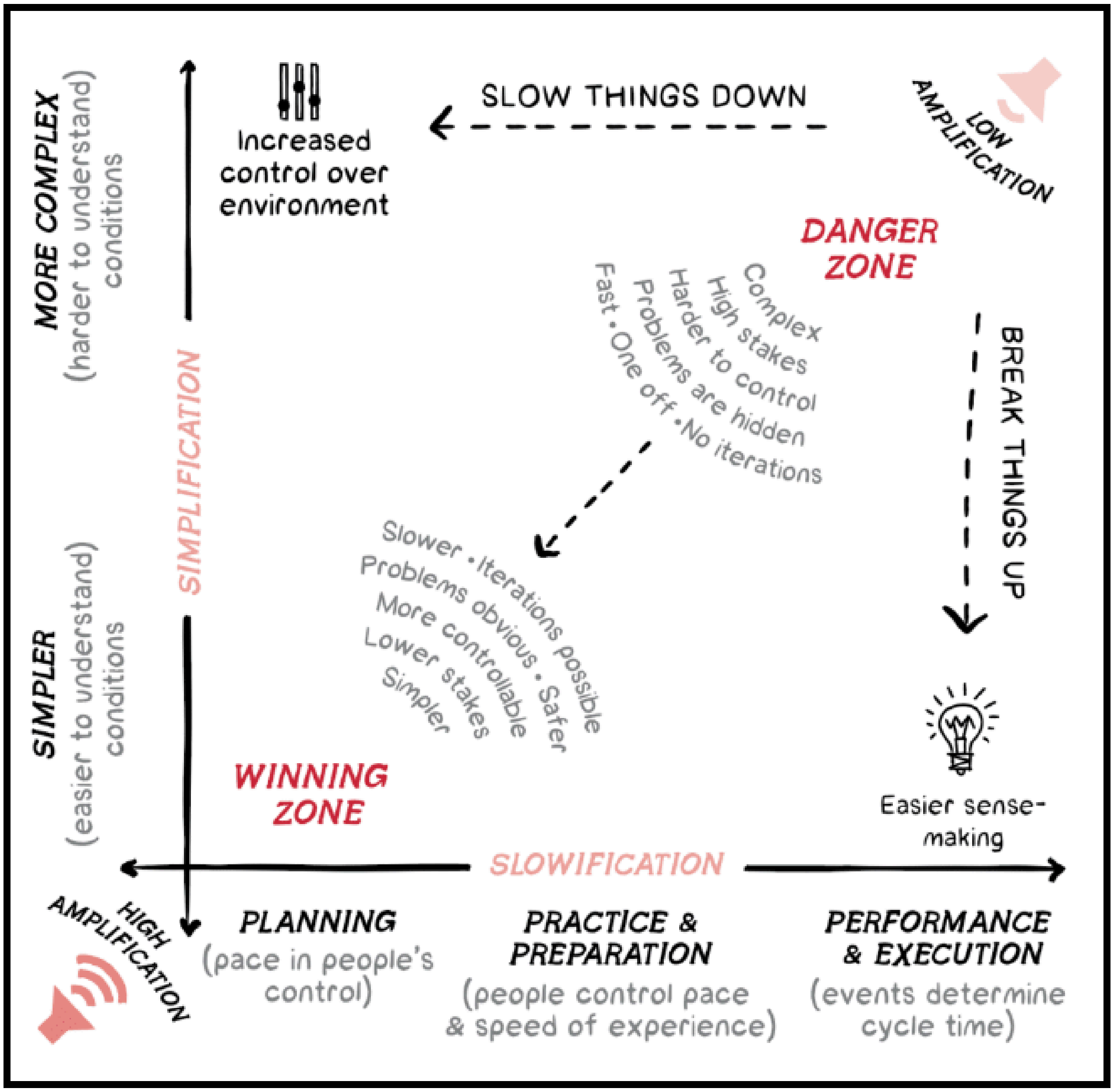


Figure 3: Moving from *Danger Zone* to *Winning Zone* (Kim & Spear, 2023,52)

Based upon the figure above, in simplification, complex problems are broken down into simpler problems in order for the larger problem to be solved. For example, if there is a large complex feature which needs to be developed those components are broken down into multiple phases. In slowification, when things are slowed down, it allows opportunities to plan which allows more control over the problems which are encountered. For example, if a feature is ready for testing, but when testing begins there are several bugs which are encountered due to design issues, that feature testing is put on hold and sent back to development so those bugs can be resolved before testing resumes. In amplification, when there are problems attention needs to be given to those problems in order for them to be solved in a timely manner. For example, if the requirements of a given feature don't provide enough detail for the development team, that problem is escalated to project management in order for those requirements to be further defined, so the development team has the right amount of information needed to proceed with feature development.

3 Slowification

Slowification is the first of the three mechanisms that move organizations out of the *danger zone* and into the *winning zone*. There are many maxims that convey a similar idea: go slow to go fast, pause to sharpen the ax, and so forth. Yet there's no single English word that conveys the duality of the idea that going faster often depends on going slower. In the absence of such a word, we opted to make one up: slowification (Kim & Spear, 2023, 70).
As an example, there was a problem where a software feature needed to be developed quickly for one of our customers. The team decided to rush through the requirements, design, and development phase. When it was handed over for testing, multiple bugs were discovered within a few hours. Instead of continuing to press through, the team decided to put testing on hold and go back and create acceptance criteria, build unit tests and execute those before it moved back into testing. While this effort slowed things down initially, it reduced the amount of defects identified during testing and accelerated the testing phase. In addition, our internal product team and the customer were happy with this feature. This allowed them to gain more confidence in our product and know that future feature requests will have higher quality which will meet their needs.

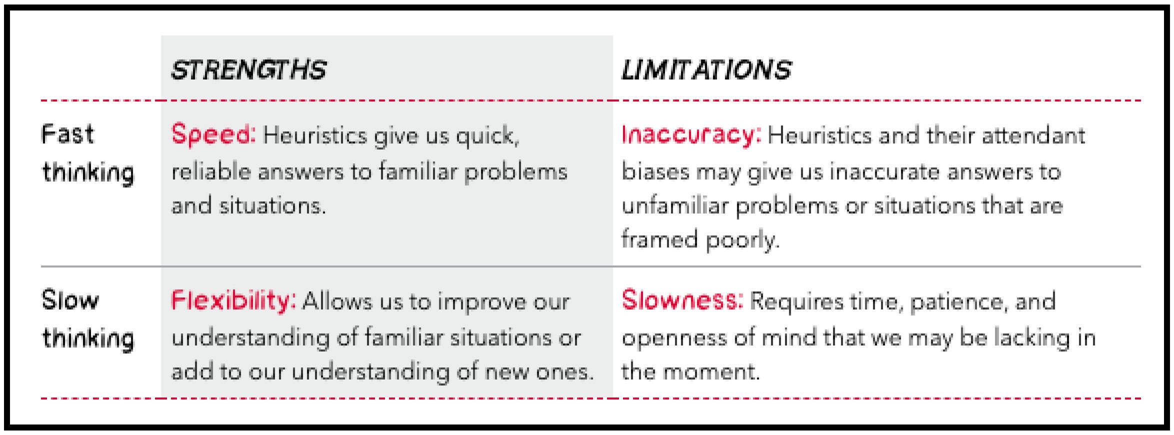
3.1 Fast Thinking vs. Slow Thinking

There are two modes of thinking: fast thinking and slow thinking. Dr. Daniel Kahneman, 2002 Nobel Laureate in Economic Sciences, and his colleague, the late Dr. Amos Tversky, distinguished between two thinking modes: System 1 (or fast thinking) and System 2 (or slow thinking) (Kim & Spear, 2023, 71).

I have given this much thought and practiced it several times and it really makes a lot of sense. I find myself often rushing to solve a problem and on a typical day I am usually faced with at least 20 problems that I need to solve. There are some problems where I can apply fast thinking and resolve those problems based upon my years of experience and familiarity within my work environment. There are times when those quick decisions backfire because I didn't take enough time to analyze and come up with the proper solution. Fast thinking can definitely cause some major problems. For example, if I would have performed slow thinking to analyze the bug that was found during testing, I would have determined that it should have been a release blocker and should have been fixed before the code was deployed to production. If this change would have deployed into production, our customers would not have been able to move power using our platform.

When we are thinking quickly, we are using only what we already know. Fast thinking doesn't encourage or allow time to improve our thinking. This muscle memory works in situations for which we've practiced, but it fails us in situations that are unfamiliar (Kim & Spear, 2023, 71).

Slow thinking is used most effectively when the proper amount of time is spent to solve a problem. If it is a new problem we are facing, we don't have the luxury of relying on our experience because it is new. For example, if you are going to a new city to meet a customer, it is a place you are unfamiliar with. While you have made the car, hotel, and plane reservation and even had video conferences with the customer, it will take you extra time to get familiar with your surroundings and be comfortable in meeting the customer in person for the first time. The next time you make the trip you can apply fast thinking because you have already had the experience and it will be much easier on the second visit.



	STRENGTHS	LIMITATIONS
Fast thinking	Speed: Heuristics give us quick, reliable answers to familiar problems and situations.	Inaccuracy: Heuristics and their attendant biases may give us inaccurate answers to unfamiliar problems or situations that are framed poorly.
Slow thinking	Flexibility: Allows us to improve our understanding of familiar situations or add to our understanding of new ones.	Slowness: Requires time, patience, and openness of mind that we may be lacking in the moment.

Figure 4: Advantages and Disadvantages of Fast and Slow Thinking (Kim & Spear, 2023, 71)

3.2 Cognitive Bias

We are all prone to cognitive biases. Biases show up everywhere, including how we perceive individuals (Kim & Spear, 2023, 73). As an example, I once hired a contractor who previously worked at Google. I had interviewed the candidate and he interviewed extremely well and I was biased to believe that he would be a great fit. When he joined, he was arrogant, didn't ask for help and struggled to learn the product and wasn't a team player so I ended up having to let him go. Reading this book has helped remind me that I have cognitive bias and that I need to be more careful when making decisions. If I see a potential candidate from a leading technology company, I will perform more due diligence with that candidate to gain a better understanding of their ability to collaborate with other members of the team and their desire to learn new products. I will also involve more team members in the decision making process to reduce my cognitive bias with this particular situation.

3.3 The 3 P's

Because of the limitations of fast thinking, in many situations we want to employ slow thinking. We want to create opportunities where we can be more deliberate, contemplative, self-reflective, and inquisitive. In this unrushed, lower-risk environment, we can deliberately challenge our pre-existing mental models and update them based upon additional experiences and experimentation. This is essential if we are going to get to new and useful outcomes and discover great things (Kim & Spear, 2023, 71).

As a leader of a team, it is important that I spend time thinking about standard processes which the team needs to follow. I have developed processes in fast thinking mode and rushed to get them out to the team, and often found those processes have gaps and it requires additional changes to make them work.

The *planning environment* (e.g., design or development) is the slowest-moving, lowest-cost, safest environment in which to develop and test ideas. In the context of this paper ideas include processes or software feature development. Here, ideas are literally or figuratively words and drawings penciled on paper, so the cost of expressing and capturing them is low. Ideas can be tested many times (e.g., through mock-ups, simulations, thought experiments). The low cost also allows running iterative experiments toward ever-improving solutions. This is the fastest, quickest, easiest environment in which to find flaws in thinking before they become flaws in doing (Kim & Spear, 2023, 75).

I am a firm believer that Agile planning in software development is essential. According to Mike Cohn, The key to successful agile planning is to recognize that it is an ongoing process, not a one-time event. Planning is important because it helps the team focus on the highest-priority tasks and ensures that everyone is aligned on the goals and objectives of the project (Cohn, 2005). Without proper Agile planning, the execution and delivery are going to have issues. As an example, when a feature was delivered, there was a rush to jump straight to building test cases in preparation for the delivery of code so the test cases could be executed. While the functional tester was greatly constrained by time and pressured to test quickly, it ultimately resulted in features having poor quality. I decided the team must focus more energy with test planning and created a process to require test plans for every feature. In addition, I recognized that test planning should be a collaborative activity across Architects, Developers, and Product Management. By including other teams in the process, it helped to gain additional input and make the test plans more robust. This process change resulted in improved quality reducing the number of defects in production.

The *practice environment* (e.g., pre production, testing, offline problem-solving) is a more demanding environment than planning because ideas are being put into action. However, actions are taken in a safer and more controllable environment. In practice, we can control the pace and the (incremental) complexity of what is occurring so we are not overwhelmed. We can also build multiple learning cycles into the experience, which allows us to keep adjusting what we are doing and how we are doing it (Kim & Spear, 2023, 76).

It is critical to have multiple test environments in software development since each environment will serve a different purpose. As an example, in my current organization, we have 5 environments. The development environment is where all new feature development and testing of those features take place. This environment has the least amount of restrictions and development and testing is done on the latest code branch. Code builds are deployed multiple times per day. The pre-production environment is a bit more restrictive in terms of user permissions and the code delivery is controlled. Code builds can only be deployed by Quality Engineering. This environment is used for final testing of the features and regression. The staging environment is used to deploy the approved code and run some basic sanity tests to ensure there are no issues. If there are issues, the code is rolled back and fixes are retested before it is deployed back into staging. The staging environment is further restricted in terms of users. The production environment is the most restrictive environment and code is deployed only with approved builds that have moved through all the other environments. We have one final environment which is our break fix environment. This environment has a copy of the production code and fixes are deployed into that environment when we need to deliver a patch outside of our normal release schedule. These environments were built over a period of time and helped to reduce the amount of conflicts we had in using the environments and helped to improve the quality of the code that was delivered to production. While the customer will not be aware these changes are being made, in the end they will be using a higher quality product that operates more efficiently and is safer to use.

The *performance environment* (e.g., operations, execution) is the most unforgiving environment. It controls the pace of the experience, forcing us to depend almost exclusively on already-developed routines, skills, and habits. We likely have limited ability to redo or otherwise correct our actions in this environment. Performance is the last place we can learn, but only if we recognize that our situation is different from expected or needed. This is also what enables us to swarm problems, limit their duration, contain their spread, and solve them. It also enables us to share and systematize what we’ve learned so the problem doesn’t reoccur. (Kim & Spear, 2023, 77).

Production environments should be restrictive and have the proper processes in place to ensure the software code is the highest quality and can perform repeatable processes over and over again. Our production environment is restricted by who has access to the environment and what they are allowed to do within the environment. Code deployments can only be done by a few members of the software engineering organization. When there are issues in production, there is an escalation process in place to quickly resolve those issues. In addition, we have a root cause analysis process which helps to ensure critical problems are quickly resolved and limit the possibility they will occur again.

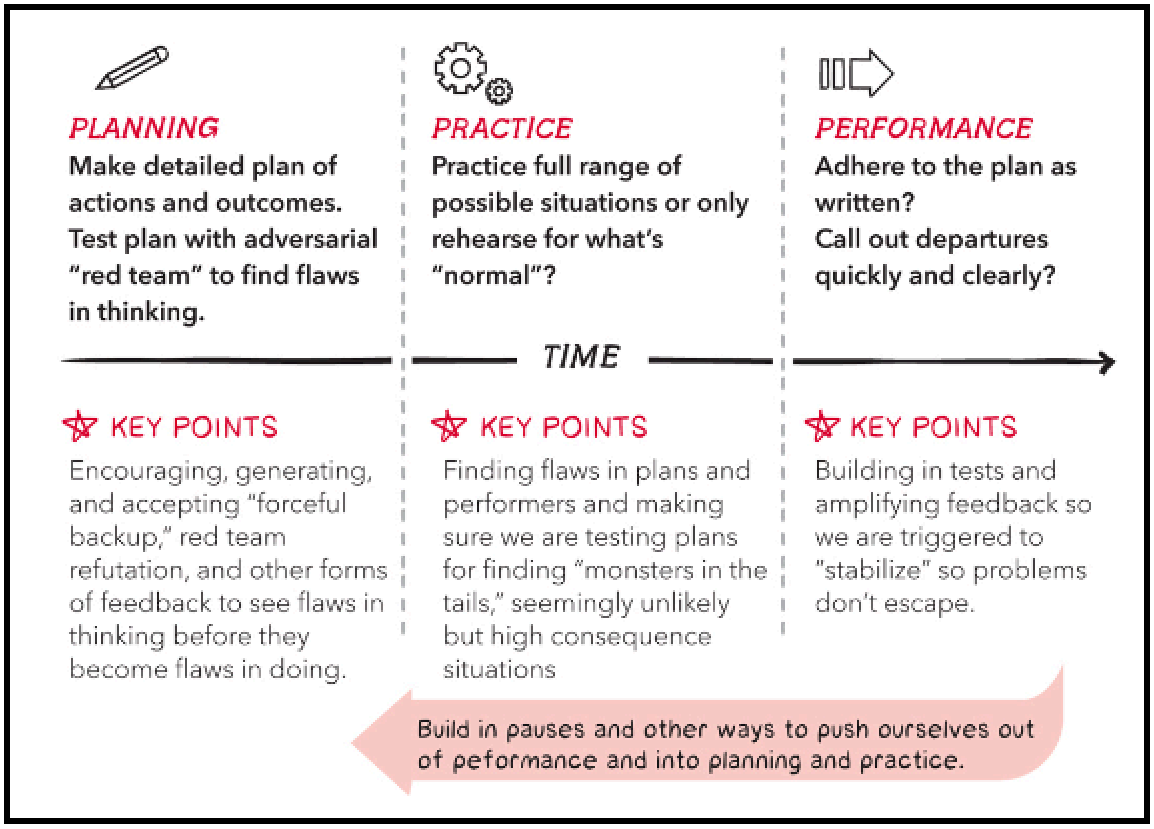

PLANNING
Make detailed plan of actions and outcomes. Test plan with adversarial "red team" to find flaws in thinking.
PRACTICE
Practice full range of possible situations or only rehearse for what's "normal"?
PERFORMANCE
Adhere to the plan as written? Call out departures quickly and clearly?
TIME
KEY POINTS
Encouraging, generating, and accepting "forceful backup," red team refutation, and other forms of feedback to see flaws in thinking before they become flaws in doing.
KEY POINTS
Finding flaws in plans and performers and making sure we are testing plans for finding "monsters in the tails," seemingly unlikely but high consequence situations
KEY POINTS
Building in tests and amplifying feedback so we are triggered to "stabilize" so problems don't escape.
Build in pauses and other ways to push ourselves out of peformance and into planning and practice.

Figure 5: The Three Ps: Planning, Practice, and Performance (Kim & Spear, 2023,75)

The figure above illustrates some important points to keep in mind with these principles. As it relates to planning, when a functional tester is going to test a new feature they must create a test plan as a part of the process. Instead of creating the test plan for their own use, they should collaborate with Architects, Developers and Product Management to build a robust test plan. This will result in a higher quality product delivered to the end customer. As it relates to practice, it is important to recognize that there might be edge cases or other features which have dependencies that should be included in the test plan to confirm both the edge cases and dependent features work as intended. As it relates to performance, it is important that the functional tester follow the test plan and identify issues as they occur. Recently a functional tester collaboratively worked with Architects, Developers,and Product Management to build a test plan for a new feature. When the code was delivered, there were other safety features which prevented testing of several of the test cases identified in the test plan. The functional tester amplified the issue so the stakeholders would be aware that the test plan would change based upon the unknown dependency. The functional tester proceeded to complete the testing and the feature was successfully delivered to the customer.

3.4 Capture Knowledge

Regardless of which phase we find ourselves in—planning, practice, or performance—how often and well do we capture lessons learned in a way that is easy to access? And how well do we create opportunities for those lessons to be learned so they can be incorporated into future planning and used in future practice? As we think through opportunities in our own work, consider that knowledge capture has been a key mechanism leading to betterment through human history, taking full advantage of writing, imaging, modeling, and so forth (Kim & Spear, 2023, 122).

In my current organization, there are a few ways we capture and gain knowledge.

1. Online Training: There are online training courses every employee is required to take. This helps to ensure that all employees learn about safety and human resources policies. Without having this knowledge captured in training courses, it would be impossible for the employees to know what policies are in place and violations would constantly occur.
2. Centralized Online Content: We use web pages to document product requirements and software development processes. These pages are updated when the information changes.
3. Lunch and Learn: Lunch and learn sessions are an effective way to encourage continuous learning and knowledge sharing within agile teams. These informal gatherings allow team members to share insights, best practices, and experiences in a relaxed setting, fostering a culture of collaboration and ongoing education (Elssamadisy, 2008). In our software development organization we have bi-weekly meetings. For the new employees, this is one of the items they must complete as a part of the onboarding process. Knowledge sharing is extremely important since we have grown rapidly over the past several years and we recognize the importance of distributing the knowledge from a few key resources to the whole team.
4. Sprint Planning: Regular planning sessions, such as sprint planning, are critical in agile as they allow teams to adjust their priorities, set realistic goals, and ensure that everyone is aligned with the project objectives. These sessions provide a structured opportunity to plan the upcoming work while taking into account the feedback and learnings from the previous iteration (Schwaber, 2004). For the software development organization, we hold a monthly meeting to review innovation projects, team retrospectives, and review features planned for the upcoming release.
5. Daily Standup: The daily stand-up is a meeting where each team member shares what they did the day before, what they will do today, and any impediments they are facing. This practice not only keeps everyone informed but also fosters a culture of knowledge sharing, where the team can collectively address challenges and synchronize their efforts (Sutherland & Sutherland, 2014)
6. Stakeholder Demos: The purpose of the stakeholder demo is not just to show what has been accomplished, but to share knowledge, gather feedback, and ensure alignment between the team and stakeholders. It’s an opportunity to validate the work done and to foster collaboration and transparency (Rubin, 2013).

4 Simplification

Simplification makes it easier to solve problems by changing the conditions to solve the problem thus making them easier to solve. Simplification makes the problems themselves easier to solve. For example, by limiting the number of active projects and streamlining tasks, the team can reduce context switching, decrease errors, and accelerate delivery times. This approach to simplification ensures that the team can focus on completing one task at a time, leading to more predictable outcomes and higher quality in the software development process (Kim & Spafford, 2013).

Simplification is achieved through three techniques—incrementalization, modularization, and linearization. In short, simplification breaks up situations that are big, complex, convoluted, integrated, or highly intertwined and makes them more manageable because they are smaller, contain fewer departures from what is already known, and are easier to understand in their construction (Kim & Spear, 2023, 130).

Moreover, simplification offers another advantage: breaking large, complex problems into smaller, easier pieces means more problem-solving can occur simultaneously (in parallel) with less coordination required. The net result is making problems easier to solve and solving more of them at once. This ability to solve more problems independently and in parallel is a consequence of all three simplification techniques partitioning large systems into smaller, coherent pieces (Kim & Spear, 2023, 130).

In our product, we have complex features that are developed. The features typically span across multiple teams and will involve hardware, firmware, and many software code repositories. The feature must be broken down into smaller components in order to build the feature and ultimately deliver it to the end customer. Our initial strategy was to have a single feature for each team. Some of these features would be large and could include up to 40 sub-features. It would typically take 2-3 months to deliver the feature and we would wait until that feature was fully complete and deploy the completed feature into production. After reading this book, we decided to change our process. Instead of having a single large feature, we would break down the feature into several smaller features. In addition instead of waiting for the full feature to complete, we would deliver the work incrementally into production. This process change did several things.

1. Breaking the feature into smaller pieces, allowed those pieces to be delivered incrementally.
2. It improved the overall morale of the team because they could see progress with the smaller epics getting completed and delivered.
3. It allowed delivery of partial features to our customers.
4. If the partial feature delivered wasn't meeting customer expectations, it allowed the opportunity to resolve those issues so when the full feature was delivered, it would result in greater customer satisfaction.

Simplification uses three techniques:

4.1 Incrementalization

Incrementalization partitions what is novel (which needs to be tested) from what is known (which is already validated) into their own self-contained, coherent units and adds to the novelty in many smaller increments rather than in a few large attempts. The benefit is that we iterate and test changes on fewer factors and on a smaller portion of our system more quickly and safely (Kim & Spear, 2023, 132).

From the example above, instead of delivering the full feature after several months of development, It was delivered in monthly increments. This encouraged more frequent feedback which helped to increase customer satisfaction when the final part of the feature was delivered.

4.2 Modularization

Modularization partitions a large, integrated system that is unwieldy in size, complexity, or intertwined relationships into smaller, simpler, more numerous coherent pieces. These coherent modules are less coupled to each other because they are connected through only a few well-defined and stable interfaces. The benefit is that small teams gain independence of action, enabling them to work and experiment on more manageable parts of the problem in parallel and more quickly and safely, with lower costs of coordination (Kim & Spear, 2023, 132).

After reading the book, we decided to change our process to allow various components of code to be delivered independently into production. In order to make the change we needed to decouple the components and validate that those components could operate independently. This allowed certain features, in our case the UI and Data API components to be delivered more frequently into production. This resulted in higher customer satisfaction.

4.3 Linearization

Linearization partitions operations that are complex and share resources to accomplish multiple objectives into independent (decoupled) and coherent workflows. Each is focused on one or a few objectives that can happen in parallel. (Kim & Spear, 2023, 132).

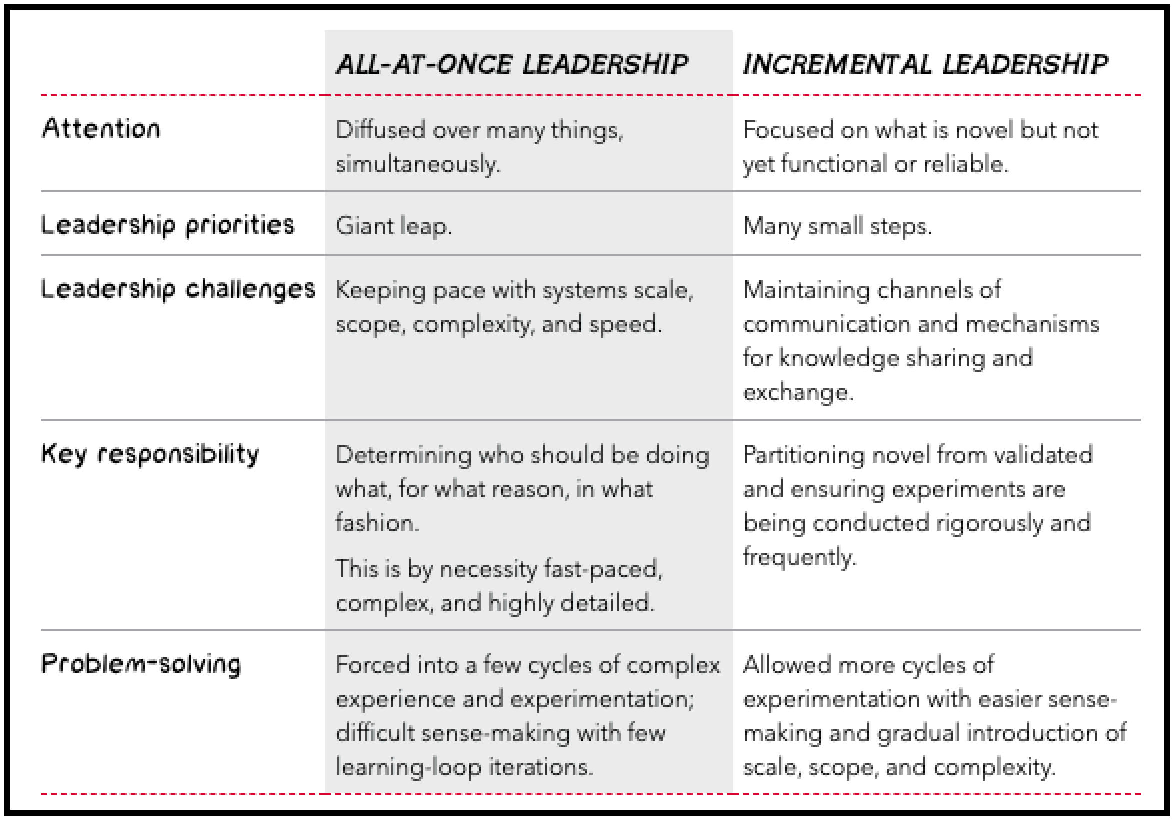
In software testing there are several activities that need to take place when testing a new feature. For the most part the activities are completed in a sequential workflow. The work tasks include: test planning, test execution, defect retesting, and regression. In our organization we have a functional team and an automation team. For the test cases that need to be executed frequently, especially after each release, we have a process in place to identify those test cases and assign them to the automation team to build and have them executed without human intervention. This allows those tests to run faster than if someone on the functional team executed them manually. Both teams are able to focus on their respective tasks independently and ultimately achieve the objective using the linearization technique.

5 Simplification Implications for Leadership

How you lead an organization that has been simplified—through some combination of incrementalization, modularization, and linearization—differs massively from how you lead one that has not been simplified (Kim & Spear, 2023, 213).

Understanding this concept has helped me change how I lead in focusing on smaller components such as a problem with a current process or an identification and implementation of a new process. This mindset change has enabled improvement of the standard set of procedures which the organization operates. Once a series of problems are solved or processes are simplified, I also reflect on those changes, to see what additional improvements can be made.

5.1 Incrementalization in Leadership

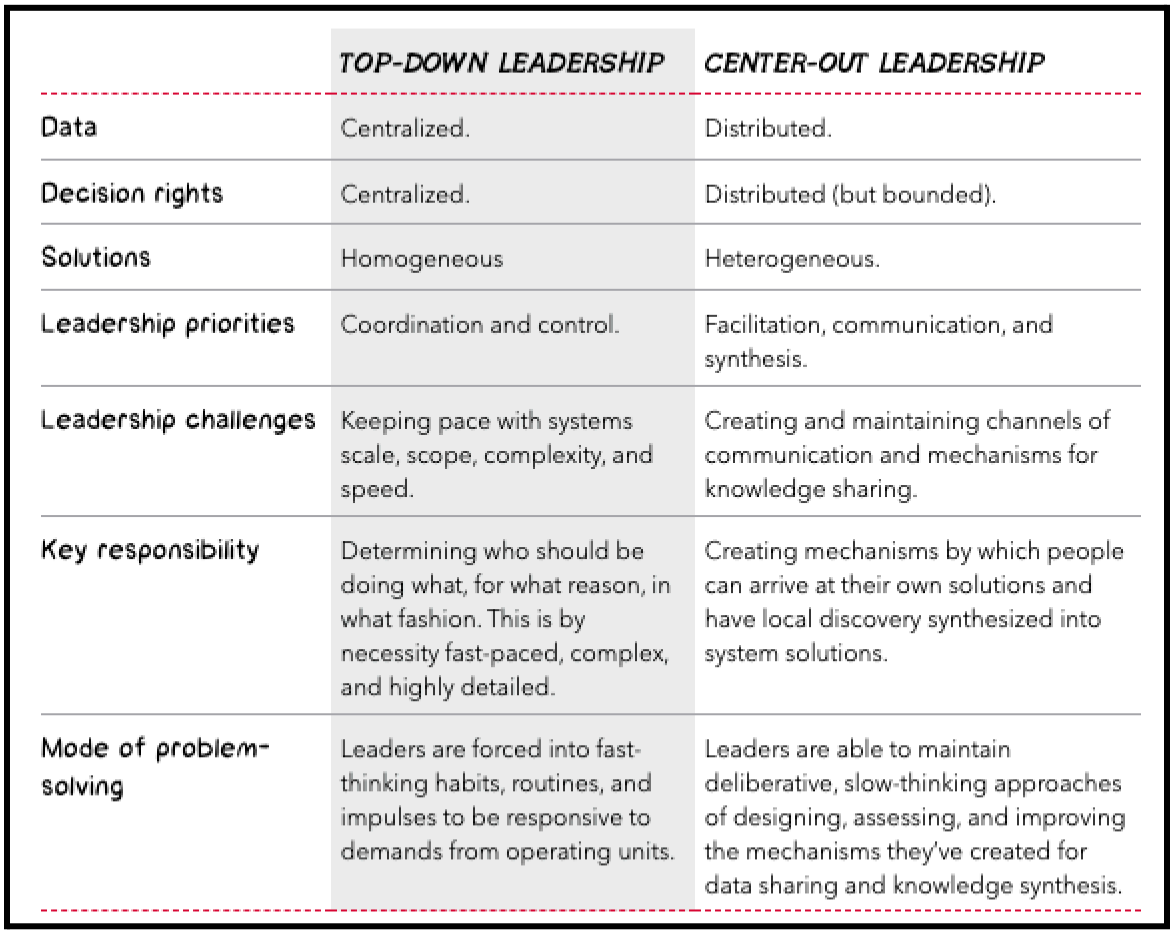


	ALL-AT-ONCE LEADERSHIP	INCREMENTAL LEADERSHIP
Attention	Diffused over many things, simultaneously.	Focused on what is novel but not yet functional or reliable.
Leadership priorities	Giant leap.	Many small steps.
Leadership challenges	Keeping pace with systems scale, scope, complexity, and speed.	Maintaining channels of communication and mechanisms for knowledge sharing and exchange.
Key responsibility	Determining who should be doing what, for what reason, in what fashion. This is by necessity fast-paced, complex, and highly detailed.	Partitioning novel from validated and ensuring experiments are being conducted rigorously and frequently.
Problem-solving	Forced into a few cycles of complex experience and experimentation; difficult sense-making with few learning-loop iterations.	Allowed more cycles of experimentation with easier sense-making and gradual introduction of scale, scope, and complexity.

Figure 6: Leadership Challenges with All-At-Once vs. Incremental Approaches (Kim & Spear, 2023, 213)

In the figure above, it provides a clear illustration of the differences in leadership style between all at once and incremental leadership. The all at once leader is likely going to constantly be stressed and is going to have difficulty managing everything at once, and in the end will accomplish less than if the incremental leadership techniques were leveraged. The incremental leader leverages others to gain knowledge and encourages sharing of that knowledge. Focused attention on conducting a series of experiments helps to understand more about the problem and ultimately come up with a better solution.

5.2 Modularization in Leadership



	TOP-DOWN LEADERSHIP	CENTER-OUT LEADERSHIP
Data	Centralized.	Distributed.
Decision rights	Centralized.	Distributed (but bounded).
Solutions	Homogeneous	Heterogeneous.
Leadership priorities	Coordination and control.	Facilitation, communication, and synthesis.
Leadership challenges	Keeping pace with systems scale, scope, complexity, and speed.	Creating and maintaining channels of communication and mechanisms for knowledge sharing.
Key responsibility	Determining who should be doing what, for what reason, in what fashion. This is by necessity fast-paced, complex, and highly detailed.	Creating mechanisms by which people can arrive at their own solutions and have local discovery synthesized into system solutions.
Mode of problem-solving	Leaders are forced into fast-thinking habits, routines, and impulses to be responsive to demands from operating units.	Leaders are able to maintain deliberative, slow-thinking approaches of designing, assessing, and improving the mechanisms they've created for data sharing and knowledge synthesis.

Figure 7: Comparing Top-Down vs. Center-Out Leadership (Kim & Spear, 2023, 215)

The figure above illustrates the differences between top-down leadership and center-out leadership. The two styles are very different. I prefer having a leader who practices center-out leadership and therefore I try to model that behavior. In my opinion, it is very difficult to manage using the top-down leadership style and it usually results in making poor decisions. The reason poor decisions are made is this style only considers a single perspective and does not encourage getting a different perspective that would result in a better decision. My preference is to share knowledge, maintain communication, and allow team members to come up with solutions based upon their own discovery. There are times however, where I might have to use top-down leadership. For example, when there is a release deadline and if I feel there is potential for the testing to not finish on time, I will provide direction on what needs to be accomplished.

5.3 Linearization in Leadership

Just as modularization and incrementalization change the nature of leadership, so does linearization. Shifting to linearized workflows partitions the entire enterprise into distinct pieces. For example, instead of having a single software development team focus on both new feature development and production support issues, split the team into two different groups so each team is able to have a single priority. That has benefits, like shifting from top-down to center-out management. Leaders have to spend less time doing frenetic, impulsive, fast-thinking work. Instead, they can be more deliberative, first figuring out the partitioning that allows for linear flows. That's deliberative slow thinking, a more productive use of people's minds. Then, they can continue in the slow-thinking, deliberative, productive mode by solving the engineering (technical) problems that affect safety, cycle time, quality, and yield (Kim & Spear, 2023, 216).

As a leader, I need to understand the problems that my team has on a daily basis. This allows me to help remove the roadblocks so they can efficiently complete the tasks they are working on and be able to move to the next task. One particular challenge we have had is the developer having a different setups on their computers versus how the computer is set up for the tester. We decided to invest time in building a separate team that would be focused on building Docker images that can be shared between the developer and tester in order to ensure consistency and be able to reproduce the issues regardless of the computer that is used. This process change will help to resolve the issues faster and help ensure consistency in how the software is configured. While this is a work in progress, I am confident this change will reduce the frustrations and allow the developer and tester to focus on higher value work that is completed.

6 Amplification

Amplification is the act of calling out problems loudly and consistently enough so help is triggered to swarm them. Once the problems are swarmed, they are contained so they neither endure locally nor spread systemically. Then, they are investigated to determine their causes and create corrective actions that prevent recurrence. This requires that the signal of a problem is successfully generated, transmitted, received, and then reacted to (Kim & Spear, 2023, 233).
As a leader of a software quality engineering team, this is a technique I use often. Even before I read the book, I would frequently call out blocking issues and escalate those issues to project management or development. Recently, we had a major release for our largest customer and we were faced with a very tight deadline. While I had called out issues which were blocking testing on a daily basis in the daily standup, the issues were not getting solved in a timely manner. I decided to create a slack channel specifically for that project and include the leadership team. The leader of the organization recognized the importance of removing those blocking issues and encouraged his leadership to quickly work to resolve those issues. By using the *amplification* technique, we were able to complete delivery of that critical project.

7 Conclusion

Organizational Transformation is a journey. It starts with a single person's desire to transform the organization into something different. One successful change led to another change, creating a series of changes. Wiring the Winning Organization provides various principles and techniques to influence change. Some changes will fail miserably, and that is perfectly acceptable. I highly encourage failure for the organization to have the experience to learn from that failure. This takes an extreme amount of courage. Let's face it, nobody wants to fail. It will require you as the change agent to lead by example and be willing to demonstrate that failure is acceptable. Once the failure turns into a successful change, others will be willing to fail to help the organization change.
Organizational Transformation requires change agents who are willing to roll up their sleeves and champion the change. It involves change agents at every layer of the organization. I desire that this paper will fuel a passion inside you to become a change agent, take just a few of these principles, and challenge the status quo at your organization to influence real change. That is the only way organizations can improve and win against the competition. Do not be afraid to speak out. Your voice is valuable. Go out and compete and win!

8 Next Steps

Here are the next steps I recommend in starting your journey towards Organizational Transformation.

1. Read *Wiring the Winning Organization*.
2. Recommend a few of the key leaders within your organization read the book and ask them if they would like regular meetings to review the book chapter by chapter.
3. Identify ways to make a few small changes within the organization as a proof of concept.

Once you have done these three steps and determined the concept will work within your organization, have a kick-off meeting to review the changes and encourage everyone to join the Organizational Transformation Journey.

References

Cohn, Mike. 2005. *Agile estimating and planning.* Prentice Hall.

Elssamadisy, A. 2008. *Agile adoption patterns: A roadmap to organizational success.* Addison-Wesley Professional.

Kim, Gene, and Behr, Kevin, & Spafford, George. 2013. *The Phoenix Project: A novel about IT, DevOps, and helping your business win.* IT Revolution Press.

Kim, Gene, and Steven J. Spear. 2023. *Wiring the Winning Organization*. IT Revolution Press.

Rubin, K. S. 2013. *Essential Scrum: A practical guide to the most popular agile process.* Addison-Wesley.

Schwaber, K. 2004. *Agile project management with Scrum.* Microsoft Press.

Sutherland, J., & Sutherland, J. J. 2014. *Scrum: The art of doing twice the work in half the time.* Crown Business.