Leading a Software Architecture Revolution

"Part 1: Creating Awareness, Preparing and Measuring"

Marden Neubert, PagSeguro Ltd,—Brazil
Joseph W. Yoder, The Refactory, Inc,—USA

Software architecture revolution can be defined as the process of making profound, large-scale changes to the fundamental structures of a software system to improve its attributes, such as availability, scalability, and maintainability, or to enable new requirements that are incompatible with the current capabilities. Architectural revolution usually demands substantial effort from the organization and thus depends on effective leadership to be successful. However, while there is plenty of research on the technical aspects of any architectural transformation, not much is available on the leadership perspective. The role of managers and other leaders include championing the revolution initiative, prioritizing activities, negotiating the allocation of people and resources, evaluating results, taking corrective actions, and reporting achievements. This paper draws from practical experiences to describe patterns to improve the effectiveness of architectural revolution initiatives.

Categories and Subject Descriptors
• Software and its engineering ~ Agile software development • Social and professional topics

General Terms
Architecture, Leadership, Management, Sustainable Delivery, Patterns, Microservices, Monolith, Strangler

Additional Keywords and Phrases
Software Development, Evolutionary Architecture, Pattern Sequences, Pattern Scenarios, Fearless Change

ACM Reference Format:
Neubert, M., Yoder, J.W., "Leading a Software Architecture Revolution - Part 1: Creating Awareness, Preparing and Measuring". HILLSIDE Proc. of 29th Pattern Lang. of Prog. (October), 58 pages.

Author's email address: marden@mvlm.net, joe@refactory.com

1. Introduction

In the software industry, the concept of “architecture” does not have a precise definition. However, it is commonly accepted that it is related to the most fundamental structures of a software system. A more formal definition of architecture states that it is the system’s structure that comprises software elements with their visible properties and the relationships between them [Clements et al.]. Less formally, the architecture of a system is, as Ralph Johnson puts it, “those parts which are harder to change” or “the decisions you wish you could get right early in a project” [Fowler].

Ideally, software architecture should evolve gradually, constantly adapting to the product’s changing requirements [Wirfs-Brock et al.]. However, because changing it is so hard, successful products—those that survive for around a decade or more—can become fossilized in an architecture that is no longer adequate. New markets, business models, and other radical organizational shifts can occur so fast that the architecture eventually falls behind. Gaps may appear in functional capabilities and in internal aspects such as maintainability, performance, stability, and quality.

In situations in which technical debt and architectural mismatch accumulate to a point in which an evolutionary approach [Ford et al.] is no longer viable in a relatively short term, the only alternative is to engage in a committed effort to radically transform the architecture, which we call *architectural revolution*.

The Oxford dictionary defines “revolution” as “*a great transformation; a perceptible change of any kind, whether progressively, continuously, or suddenly.*” We use the term revolution to imply that architectural work will be performed and managed as a dedicated initiative, instead of just organically, with tasks living side-by-side with new features in the teams’ backlogs. We intend to contrast the name with “evolution” to highlight that the transformation process is perceptible to the organization. As the dictionary definition clarifies, a revolution can be sudden, but it can also happen progressively. It does not mean the system will be rewritten from scratch [Spolsky], although this is an option, especially for smaller applications and modules. Most importantly, a revolution initiative will be more successful by using an iterative and incremental process and applying architecture refactoring techniques to achieve its goals.

The role of leadership in an architectural revolution is critical because such an initiative requires a high level of persuasion and coordination. However, published material on software architecture focuses primarily on technical aspects, leaving leadership and management responsibilities out of the picture. This paper gathers practical experiences from the authors—a CTO who led a significant revolution and a consultant advising many companies in similar situations—to fill that gap and provide effective guidelines to managers and all types of leaders needing to revamp their architecture.

The paper gives some background information, followed by an overview of the pattern language, including some scenarios. This is followed by the patterns in detail. Although most of the patterns described in this paper can be applied in any revolution initiative, some are more relevant when migrating from a monolithic architecture (often called the “legacy application”) to microservices.

These patterns are primarily intended for potential leaders of revolution initiatives, which are typically top-level managers within the IT organization, in positions such as CTO, CIO or Director of Engineering. However, other professionals involved with software architecture will also benefit from learning the patterns, including architects, technical leaders, and developers. Even non-IT people can apply the patterns to raise awareness about the need for transforming the architecture and to spark change in the organization.

We chose the pattern format because of its clarity, modularity, and expressiveness. We use a modified Alexander style for the patterns, which includes the context of the pattern before the first bold problem statement [Alexander et al.]. This problem statement is followed by a discussion of the forces and then the core of the solution in bold, followed by details of the solution. We then end the pattern with a section after the stars which discusses consequences and related patterns.

2. Background

Business software is never finished. Sometimes simple software created by a small IT team finds a product-market fit for the organization [Ries]. Companies grow because new business opportunities emerge. Pressure to adapt to and shape the market means new features are added, new interactions are accommodated, and new teams must work on the software. Sometimes a straightforward software architecture that starts out small when communication is easy can support guided, incremental architectural changes [Ford et al.] and can gradually evolve with its environment, remaining fit for its purposes.

Other times it is not so simple: the initial software architecture can be poorly suited for supporting required changes, or the accumulation of suboptimal architectural decisions (also known as architectural technical debt [Nord et al.]) can be too severe; in either case the architecture needs an extensive revision. Because changing a software architecture is typically slower than adding new features [Besker et al.], the situation calls for an architectural revolution. Even if teams know how the architecture should be modified, implementing such changes by simply including technical tasks in their backlogs may not be viable because of business priorities.

Architectural revolution is an arduous initiative and requires a significant commitment by the whole organization, not only the IT department. When the organization embarks on an architectural revolution, teams must adapt to a new architecture, priorities will be affected, and considerable time and money will be invested. Everyone involved with product development should be aware of what is going on and understand the need for the architectural revolution.

Therefore, the role of project champion or sponsor in a revolution initiative is crucial and is typically taken by the leader of the IT organization, although the head of product development or another high-level manager can play this part. Initial responsibilities include creating awareness about the current situation, establishing the need for an architectural revolution, defining the goals, and negotiating how to allocate teams and resources to start the initiative. During the project, the champion should evaluate intermediate results, assess team effectiveness, provide support, and take corrective actions when necessary. At important milestones, the project champion must give the organization feedback on the initiative’s paybacks in order to avoid the reallocation of people and budget.

Many software projects start with a monolithic architecture [Richardson] [Yoder, Merson]. A monolithic architecture is the right starting decision for many situations—it is not an anti-pattern. However, a monolithic architecture is the style that most frequently needs to undergo an architectural revolution. Because the most limiting aspects of a monolith include coupling and having to release the "complete" system as one piece, a usual target architecture for a monolith is microservices. The microservices architectural style can provide a high level of independence among subsystems and enable pieces of the system to be deployed independently. Although any architectural style can be subject to an architectural revolution, many of our examples come from monolith to microservices migrations.

3. Overview of Patterns

Figure 1 shows the patterns discussed in this paper under the groups "Creating Awareness" and "Preparing and Measuring." We also show other groups of patterns we have identified and will describe in future works. Brief descriptions (patlets) of these patterns are provided in **Appendix B**, which includes a summary of all patterns in this language.

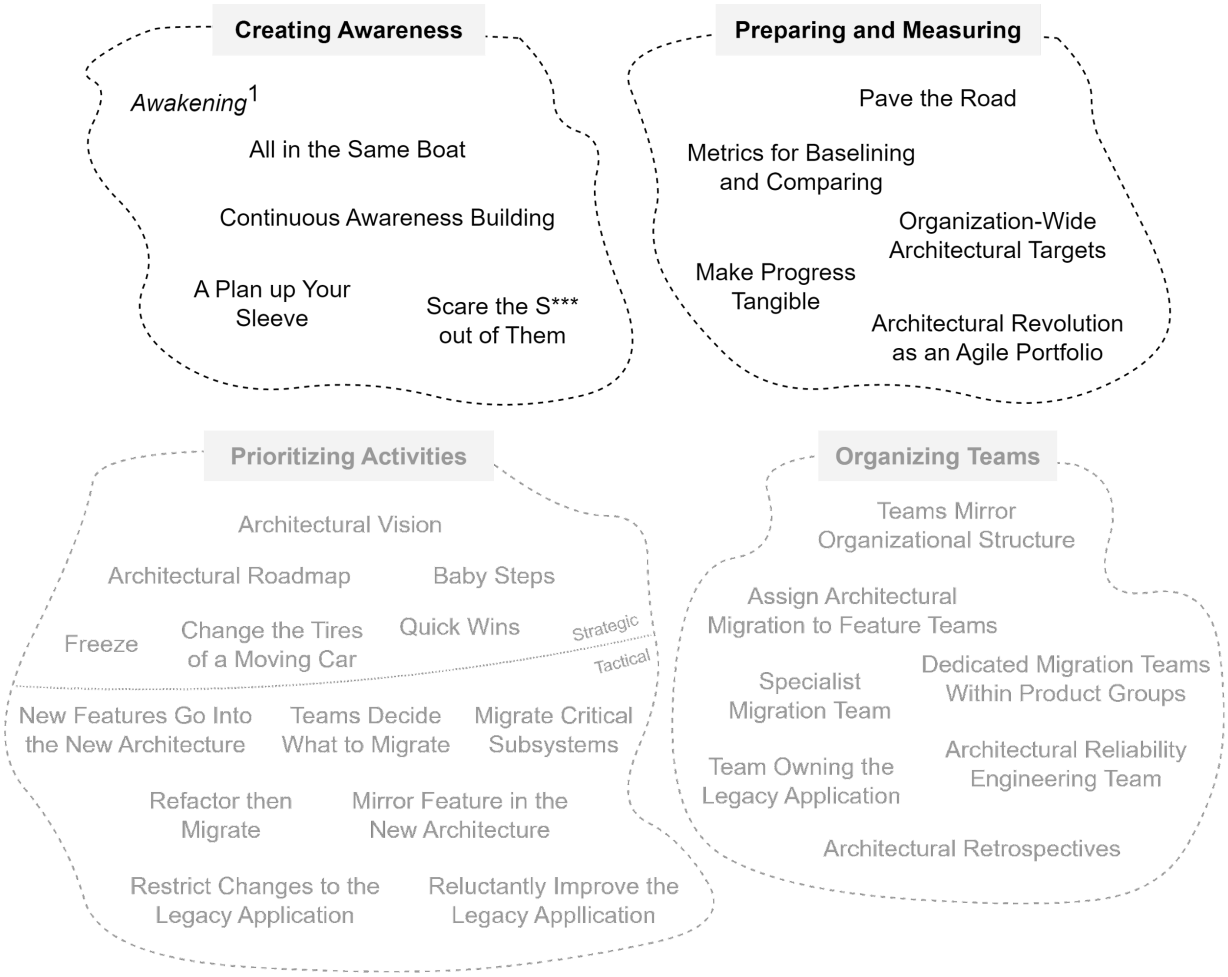

Figure 1: Patterns for Leading a Software Architecture Revolution

To avoid redundancy with related works, we adopted two criteria to select patterns for this paper. First, the patterns must be related to architectural revolution. In other words, they should be applicable in a scenario involving a committed effort to a profound change in the architecture rather than a more incremental natural evolution of the architecture. And second, the patterns must be related to a leadership or management role. That is, they reflect activities and responsibilities in the scope of the revolution initiative's leader, rather than technical patterns applicable by engineers.

The patterns described here were encountered while using agile methodologies. Although we can speculate that the essence of these patterns can be applied in waterfall-like processes, the description of most of the patterns will imply that the teams—and ideally the organization as a whole—are using an agile methodology or at least have an agile mindset.

To make radical changes to an architecture, it is important to start **Creating Awareness** of the issues so that you will have buy-in and support throughout the organization. Once you get the commitment, you should start **Preparing and Measuring** if you are moving in the right direction. To have the best outcome from the initiative, you must ensure that teams are working as effectively as they can by **Prioritizing Activities**. Revolutionary architectural changes will often require organization and cultural changes, therefore consider **Organizing Teams** to preserve the focus on the revolution initiative.

4. Patterns for Creating Awareness

Awareness is a precondition for action, so it is necessary in any change initiative. You cannot assume that everyone in the organization understands the issues with the current architecture. Even people in the IT department may not be fully aware of the impacts caused by the legacy application. This first group of patterns describes ways to level that understanding by showing practical effects in the product development cycle, the risks the organization is incurring, and what can be done to improve the situation. These patterns help create awareness and bring people together with a shared vision for moving forward with the transformation of the architecture.

Figure 2 depicts the relations of the patterns for creating awareness. The patterns in this group are shown inside the dotted line. Other patterns described in this article are outside the border, in black font. Patterns from our language that will be described in future works are shown in gray. Patterns from related works are cited in gray and italics, with the source referenced in square brackets and brief descriptions (patlets) provided in **Appendix A**. The relationships between patterns in the language are explained by labels close to the corresponding lines and should be read in the direction of the arrows.

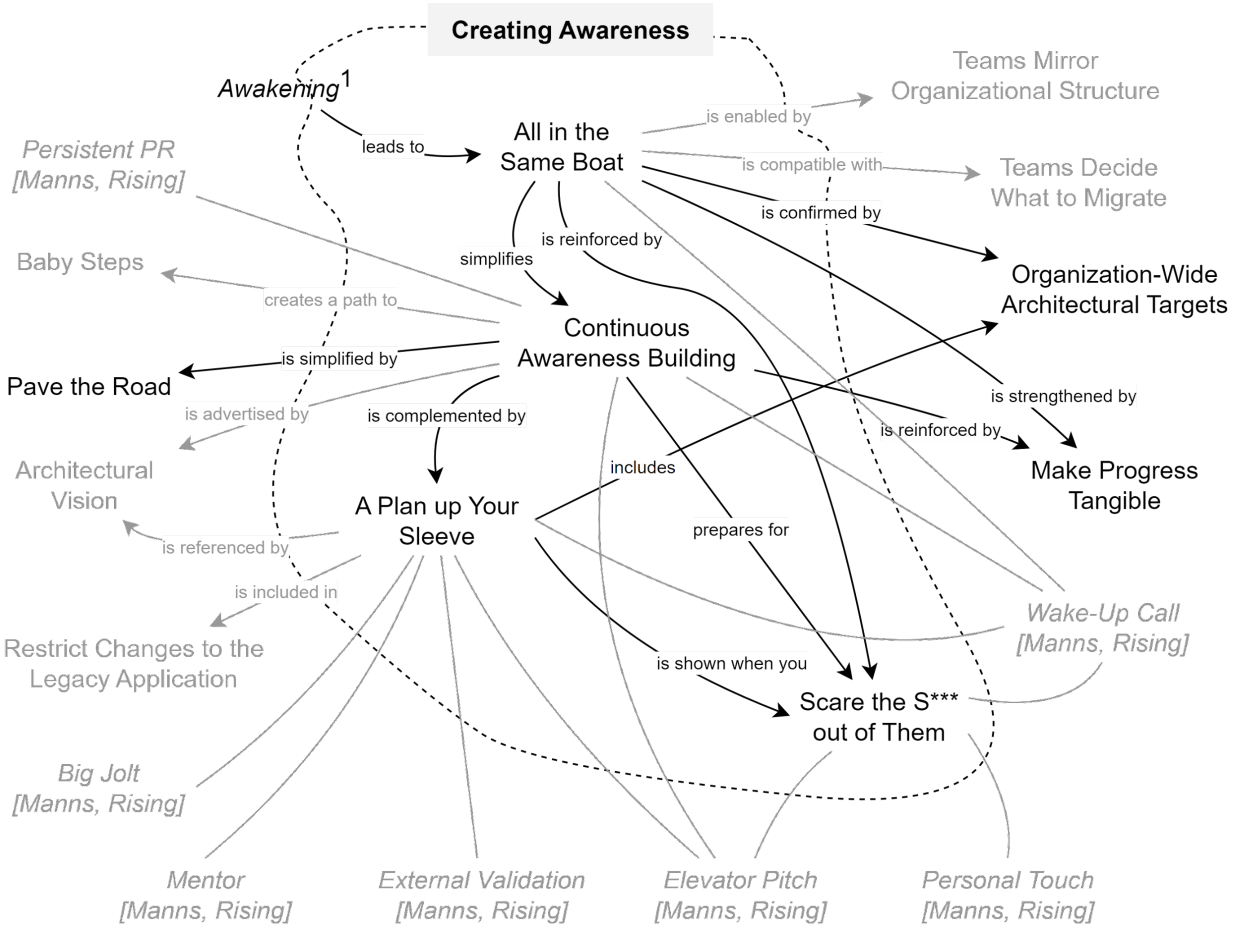

Figure 2: Patterns for creating awareness and their relationships

Before an architectural revolution can happen, the organization must become aware of the problems with its software and acknowledge it needs to be fixed. This begins with some form of ***Awakening***[1]. An ***Awakening*** occurs when an individual or a group (possibly teams) notices issues with the current system. This can include detecting that the "normal" ways of evolution are not working or that the architecture is unable to support an evolutionary approach. What does an ***Awakening*** look like?

Here are some telltales. Teams spend considerable time coordinating changes in the software, which is becoming harder to test; deployments take longer to plan and execute and sometimes fail, leading to painful rollbacks; teams have trouble meeting deadlines and cannot focus on improving quality.

At first this might seem like a prioritization problem or a lack of team coordination. The organization might believe it needs to hire more experienced engineers and adapt how teams interact. These are valid considerations and might be part of the problem, but when the organization notices the telltales, it should take a close look at the software architecture. Perhaps the current architectural style is reaching its limits and needs to change. That is, a threshold has been reached.

After an ***Awakening***, the organization will understand the situation much more easily if the departments involved in product development are **All in The Same Boat**. **Continuous Awareness Building** can help IT teams and related areas sustain development and continue to promote the need for the new architecture. You should have **A Plan up Your Sleeve** to offer some hope to the teams and pitch it to get approval for the revolution initiative. When you have the opportunity to talk to upper management about the initiative, be emphatic but **Scare the S*** Out of Them** with the risks presented by the current architecture.

Many patterns from other works related to this group come from the book Fearless Change [Manns, Rising]. A ***Wake-up Call*** can help to get **All in the Same Boat**, improve **Continuous Awareness Building** and provide stories and warnings to be used to **Scare the S*** Out of Them**. An ***Elevator Pitch*** can be a condensed message that can be included in **A Plan up Your Sleeve** and also helps in **Continuous Awareness Building** and **Scare the S*** Out of Them**. **Continuous Awareness Building** is a form of ***Persistent PR***. ***External Validation*** and ***Mentor*** can give better support for **A Plan up Your Sleeve**. A ***Personal Touch*** can be added when you **Scare the S*** Out of Them**.

[1] ***Awakening*** has not yet been written as a pattern, but could be. It is one of the first things that must happen before any architectural revolution takes place. ***Awakening*** can be considered an implicit pattern for this paper.

4.1. All in the Same Boat

Credit: Photo by "Mike" Michael L. Baird, CC BY 2.0

Your organization relies on a software system that has been providing value for some time. It enabled them to launch products and find a market segment to operate on. In the beginning, the software architecture was easy to understand and a good fit for your then small development team. They could implement new requirements quickly, and everyone knew what was happening. Gathering those requirements was simple, too—a single person was responsible for the product roadmap, defined the next priorities, was available during development, and validated the results.

Since then, the organization has grown in customers and employees, especially in IT. Stakeholders in areas such as Operations, Finance, and Compliance are requesting more software changes. Some teams have been assigned to these needs, but demand exceeds capacity. Prioritization is problematic, given that each department believes its demands are the most urgent and sometimes cannot decide which of its requests to tackle first. They do not participate in the teams' activities; instead, they present rough ideas to IT and come back just to see the end result.

IT teams are also pointing out problems. IT leaders complain that business people usually question estimates but do not want to understand the complexities involved. This led to an investigation and an ***Awakening***, highlighting that the current architectural style is reaching its limits, and there is evidence that the architecture should change.

How can we bridge the gap between the business and technical sides of the organization and get everyone involved with software development to work together to radically change the architecture?

The purpose of any IT team is to solve business problems, either directly or indirectly. Teams working on the needs of a given department are essentially a part of its organizational structure. These teams can help the area to reduce manual work, become increasingly efficient, and achieve more. However, some business leaders may not see things this way. They alienate IT teams by treating them as a mere provider of commoditized solutions, a bureaucratic gatekeeper, and a cost center. They miss the opportunity to improve the effectiveness of their departments and the organization as a whole.

When business people regard IT teams as part of their unit, any issues that affect a team should be seen as impediments to the related department, even if these are technical limitations from the software architecture. This is not to say that architectural weaknesses and other technical shortcomings are not IT problems. They are the result of technology decisions, so IT must take ownership of them and provide solutions. Yet, business people working with development teams should be aware of those issues and be interested in mitigating them.

IT teams are more motivated and engaged when they understand the purpose of their work [Melo et al.]. *Customer collaboration* is one of the core values in the Agile Manifesto [Beck et al.]. When teams work closer to their business partners and participate in discussions about challenges and goals, they feel that they are part of a group with autonomy to make decisions and adapt as needed. As a result, teams gain a deeper understanding of the problem domain, contribute to the debate, and ultimately perform better. On the other hand, teams are also part of a technical department that needs coordination to support the organization as a whole. It is important that independence and self-organization do not turn teams into separate silos that do not cooperate with each other.

Some companies have specific goals and targets for different business units, which may be tied to financial rewards, as discussed in **Organization-Wide Architectural Targets**. In those cases, trust and engagement will be higher if teams share targets with their business peers. However, this may limit the flexibility of relocating team members across business units and pose difficulties for defining targets for shared professionals (for instance, IT managers and operations).

Therefore, bring stakeholders together to participate in the daily routine of IT teams so that they understand technical challenges and share responsibility in decisions. Also, have IT leaders participate in strategic discussions, learn more about the organization, and contribute with ideas.

Bringing stakeholders together to actively participate and understand the challenges faced by the teams is the essence of **All in the Same Boat**. Having **All in the Same Boat** is a good practice for any organization and is a critical starting point when beginning a software architecture revolution. The focus of this pattern is on achieving the goals of the revolution. Once you have successfully achieved the goals of the software architecture revolution, it is a good idea to continue with **All in the Same Boat**.

A good way to start is to assure business leaders that IT teams function as a part of their organizational structure and are more effective when working closer to the problem domain. Demonstrate this by having business leaders contribute to relevant team-related decisions. Let them understand the challenges that IT faces and be open about important issues, both technical and organizational. With that, you will not only reach better solutions, but also make business leaders aware of the challenges in IT and feel that these challenges are also theirs. This can be done by either a bottom-up or a top-down approach.

Bottom-Up Approach

Carefully identify who are the business leaders that you will reach out to. They should have a relatively large scope in the organization but should not be too high in the hierarchy and far away from the operation. The closer they are to the processes of their areas, the better they will be in prioritizing work and sharing an inspiring vision. Ideally, they would be influential and charismatic so that their units would agree and follow their decisions. Focus on those with at least one dedicated IT team, as this will reinforce the sense of ownership.

Build trust as you approach business leaders with this new arrangement. Assure them that IT teams work on the priorities these business leaders define and are aligned with the goals of their areas. In most companies, developers and other team members report to a central IT hierarchy, which may make business leaders suspicious that teams follow an IT agenda. Explain that the backlog drives the teams' activities and defines their commitment. You should be able to show that having a centralized IT hierarchy benefits business leaders because it relieves them from having to hire, train, and manage the career of IT professionals.

Create agreements with business leaders and their representatives so that teams can dedicate the necessary time to IT-related activities. Show the importance of having a consistent set of standards, platforms and tools across all IT teams and the benefits of centralizing some activities such as information security and infrastructure management. This will imply that, from time to time, teams should be available to work on demands from those areas—the architectural revolution is an example. Reinforce that teams will remain primarily focused on business goals and that the organization can have the best of both worlds if teams can find a balance between domain and IT problems.

Some business leaders may think IT teams should focus only on development activities, receiving clear requirements, and delivering working software. Elaborate on how teams are more effective when working closer to the problem domain, especially when end users are involved. Show that direct communication is more effective and efficient than written documents. Connect them to other IT leaders and developers and show that these are competent professionals with deep knowledge of the operations from the software perspective.

Top-Down Approach

Another approach is top-down: start with the CEO or the executive overseeing the business units you want to have onboard. Give that person your best ***Elevator Pitch*** on the benefits of moving to the new architecture and highlight in practical terms how it will help the organization's performance.

After getting the CEO or overseeing executive onboard, you then work down the hierarchy, reaching the managers closer to the teams, showing them the benefits of cooperation, and using top executive approval as validation of your ideas and a sign of strategic value. Be careful not to use this consent as an edict, or your approach may backfire. To build a long-term relationship with your business peers, prefer the bottom-up alternative, or at least take your time to pitch the idea to mid-level managers before approaching the top of the hierarchy.

You can combine the top-down approach and piloting **All in the Same Boat** with more cooperative business leaders before approaching others. It is a good idea to take **Baby Steps** and work with just a few areas, learn, adapt and then let the more skeptical leaders know the

results. It would be even better if the business leaders participating in the new setup were the ones showing off their achievements to their peers. A combination of top-down and bottom-up will yield the best chances of success.

Business and IT Interaction

Although business leaders can be motivated to work closer to IT teams, they are not likely to have time available to act as Product Managers (PMs) or Product Owners (POs). Identify if someone from the business unit can perform that function, even if it requires formal training. Someone who already knows the area's operations and has the trust of the business leader will be better suited to fill the PM/PO role. However, other traits such as prioritization skills and good communication must be considered. If no such person can be found in the organization, suggest hiring someone with knowledge in the domain, ideally with expertise in agile. Having POs report to business leaders, although not a strict requirement, is an important sign to reinforce that teams are a part of their areas.

In some cases, one or more departments will not have enough demand to justify a dedicated IT team. Avoid having teams switch between different areas and constantly change their backlogs and missions, as this will affect the sense of ownership. Instead, try to appoint a leader to be responsible for a group of departments and define development priorities among them. Choose this leader based on the organization's hierarchy or affinity with the scope at hand.

Define how the business people will participate in the IT teams' routine. If you are using an agile method such as Scrum, one starting point is the regular meetings defined by the development process. POs will naturally participate in some of these ceremonies, but you should also invite business leaders to some important reviews and retrospectives.

It is also important to have regular strategic meetings with business and IT leaders to discuss broader topics, such as product vision, team composition, budgeting, and recurring issues. These meetings are also opportunities to discuss technical limitations that the teams may be facing and to prioritize architectural work. Sometimes the decisions to push for more features and tight deadlines are taken by PMs/POs at a tactical, team level and reflect local optimization. When business leaders can understand the long-term impacts, they may reconsider some priorities or even clarify that some deadlines did not need to be so rigid in the first place.

Stimulate business people to share their challenges and concerns. This will make these meetings more relevant and increase the feeling that IT is there to help. IT leaders have an important role in these meetings. To continually build trust, they must be transparent in matters such as the composition of IT teams, skill gaps, and technical risks. Make sure they understand the challenges the organization is facing and that they participate actively in business discussions when they arise. These are opportunities for IT leaders to preview what will be in the backlog of their teams, share their ideas, and brainstorm better cost-benefit alternatives.

Impact in Architectural Quality and the Role of the IT Leadership

Organizations that truly practice **All in the Same Boat** can curb and even reduce architectural decay. Nevertheless, they still may have to engage in an architectural revolution. If the collaboration starts after the current architecture has deteriorated significantly, simply prioritizing technical tasks in the teams' backlogs may not be enough to change the landscape. When the organization is relatively large, with around a dozen or more teams working on the same

application (typically a monolith), it will need centralized coordination to evolve the architecture and help teams perform better and grow in number. If the organization pivots its business model and quickly changes its products or customers, the mismatch between the current system and the desired state may also call for a concerted effort to change the architecture.

At that point, the role of the top IT leader comes into play. Taking the concept behind **All in the Same Boat** to the highest level of the organization, the CTO or CIO should have a seat at the table [Schwartz] with other C-level executives, including the CEO. The IT leader should summarize the current situation, show the risks, and explain the need to prioritize technical work. If this happens before the architectural gap is too large and the teams can mitigate the issues by accommodating tasks in their backlogs, a full-blown revolution may be avoided. Otherwise, the IT leader could convince other executives that a dedicated architectural initiative will be necessary.

All in the Same Boat is reinforced when business people and IT teams join to participate in decision-making discussions. A technique employed in one of our case studies was to conduct a *pre-mortem* [Kahneman et al.] early in the development of a product, in which participants imagine that the worst scenario has happened to their initiative, and develop a story exploring the causes of the failure. By having business and IT people together in that discussion, each side can better understand the challenges that the other faces and realize that both are vital to the success of the project. More specifically, when IT professionals wrote about catastrophes caused by software failures related to architectural limitations, business leaders became very conscious of the risks and were willing to promptly prioritize technical tasks.

* * *

All in the Same Boat is beneficial for any organization creating software to run its business, not only those engaging in an architectural revolution, given that this pattern enables agile practices that promote transparency and business-IT collaboration. This partnership leads to several well-documented advantages in product development, such as increased visibility and predictability, risk reduction, improved quality, greater flexibility, and customer satisfaction [Beck et al.]. It also directly benefits software architecture. Suppose business leaders genuinely regard IT teams as part of their organizational structure. When teams openly discuss architectural limitations, these leaders start to view system issues as impediments to their goals. They may then become more inclined to agree with prioritizing architectural tasks. If this new arrangement is established early enough, it may help prevent technical debt from accumulating to the point in which a revolution is necessary.

It can be difficult to implement and practice **All in the Same Boat** in organizations in which a blame culture prevails [Schwartz]. In those cases, business leaders may choose to stay away from IT and not get involved during product development. By doing so, when there is a failure, they can just charge IT for the outcome. Such leaders will not easily accept to get in the same boat and share responsibility for any system-related issues, including architectural limitations. To start changing that situation, IT leaders should take **Baby Steps** and get some more approachable business leaders in the boat, help them improve the performance of their departments, and have them show other leaders the results.

When an organization implements **All in the Same Boat**, some IT leaders, especially those used to a command-and-control management style, may feel that their authority over IT teams is diminished. Teams now have a clear goal to satisfy business needs, and business people understand that teams are part of their organization. IT leaders can no longer issue direct mandates to teams because they must align priorities with business or risk breaking the trust between the areas. This situation may give the impression that IT leaders have lost their rule over teams. Some teams may become so engaged in solving business problems that they do not welcome tasks coming from IT, reinforcing that feeling. This undesirable outcome makes technical directives more problematic to implement, which works against the architecture's soundness.

To address this consequence, IT leaders must understand that they should adapt their leadership style to the new arrangement established by **All in the Same Boat**. Shared leadership [Weichbrodt et al.] is more appropriate for self-organized agile teams. IT leaders must use more influence than authority in this setup and justify the prioritization of technical directives to the teams, the revolution initiative being an example. Aligning needs that originated within IT with business leaders also brings more transparency to teams' dynamics. Otherwise, teams would have to undertake these extra activities without the business leaders' knowledge, affecting their perceived efficiency.

Besides that, teams should know that, in addition to their core mission to solve business problems, they are part of a technology department. This unit has guidelines that play a role in the larger organization's needs. If teams do not follow those directives, undesired outcomes may arise, such as cost inefficiencies, lack of standards, security risks, operational failures, and quality issues.

Getting teams to work on a task when they are **All in the Same Boat** requires extra effort. However, this is compensated by the fact that they are small and autonomous, which enables them to make fast decisions.

Related Patterns

If technical debt accumulates despite an organizational structure that puts **All in the Same Boat**, it is possible that teams do not have a clear perception of the issues with the current architecture and what is the desired state. You may have to raise the level of understanding about those topics through **Continuous Awareness Building**.

Sometimes business leaders are not convinced about the risks of the current architecture and are reluctant to get **All in the Same Boat**. In those cases, you may have to take radical action and **Scare the S*** Out of Them**. Approach business leaders and other managers and share the effects that the legacy system can have on the operations. If you convince them that the organization should take immediate action, getting **All in the Same Boat** will be easier.

As you take the first steps in the path of the revolution, you will strengthen the feeling of being **All in the Same Boat** by informing everyone about the latest achievements and obstacles. Given that tasks are highly technical and not visible to non-technical people, you should **Make Progress Tangible** to bring business and IT closer together toward the goals.

Defining goals related to technical capabilities for everyone involved in product development, as described in **Organization-Wide Architectural Targets**, helps business and IT people understand that they are **All in the Same Boat**. This message signals that business leaders are also responsible for the services' availability, response time, and other internal qualities.

If your organization sets targets individually per business unit, consider whether teams should be assigned the same goals as their corresponding group. Doing that will increase the perception that they are **All in the Same Boat** and energize teams and business people to work together toward their goals. On the other side, it can also drive apart teams working with different business units, making it harder to transfer team members, exchange knowledge, and foster collaboration among teams. Having some common targets related to technical capabilities such as **Organization-Wide Architectural Targets** help create a feeling of unity even when teams have different goals. The best solution will depend on how often team members relocate across units and how much business leaders value having IT teams with the same targets as their departments.

All in the Same Boat is related to the organizational patterns presented in Patterns for organizing teams. Specifically, the pattern is strengthened when applying **Teams Mirror Organizational Structure** because the pairing between business and engineering leaders brings more trust and alignment in each area of the organization. Teams are empowered and make more decisions locally, which increases motivation and efficiency. This is a way of recognizing Conway's Law [Conway] and using it to make the software architecture resemble the larger organizational structure, called Inverse Conway Maneuver [Ford et al.].

Having **Teams Decide What to Migrate** is an excellent way to show that everyone is in the same boat during an architectural revolution. This pattern empowers teams to choose their path to adopt the new architecture and promotes joint decisions from IT and business on how to approach the revolution initiative. The resulting choices are more balanced between technical improvement and business value. Another positive outcome is that everyone involved will have an increased sense of ownership over the new architecture, given that they built it together. This renewed commitment can help prevent technical debt from accumulating in the future.

A ***Wake-up Call*** can help build awareness and bring different parts of the organization into the same boat. Difficult situations can create strong bonds. When IT and business are involved in dealing with an outage or a client issue, the hardships experienced and the lessons learned reinforce the feeling that they are **All in the Same Boat**.

An ***Elevator Pitch*** can be used whenever possible to convince business leaders to treat IT teams as part of their organizational structure.

4.2. Continuous Awareness Building

Photo by Laurenz Kleinheider on Unsplash

You have a system with an architecture that has been providing value to your organization for some time, but it is now reaching its limits. Business leaders are getting **All in the Same Boat** with the teams and are experiencing these limitations. Some teams can now prioritize more technical activities. Still, the architectural gap is already too wide, and you estimate that, at this pace, you will not be able to avoid the need for a revolution.

There has been some effort by developers or teams evaluating alternatives for a new architecture and testing proofs of concept. This has led to some development experimenting with the new architecture. You believe that most teams should already be using this new architecture. However, many teams are still developing features in the old architecture, increasing the technical debt they will later have to pay.

How can we help people throughout the organization become and stay aware of the current problems with the legacy application and the benefits of moving to the new architecture?

As you try to get **All in the Same Boat**, you cannot expect everyone to board fully and immediately. Even if you successfully bring business leaders and PMs/POs to share the ownership of the software and its architecture, most teams will take time to include technical tasks in their backlogs and dedicate only part of their time to those tasks. After all, their primary goal is to help their business unit achieve its objectives.

Most people in the organization, especially those in charge of strategic and financial decisions, are not constantly exposed to the limitations of the legacy application. They may have heard of problems, likely as part of an explanation for a release delay or an outage. However, it is essential that they understand the need to invest in changing the architecture and that this is a strategic move.

At this point, you may already have a vision for what a new architecture should be. You have been experiencing the limitations of the current application for a long time, so you know what to keep and avoid in a new architecture. You have researched options for your scenario and tested some alternatives in proof-of-concept projects or a simpler subsystem.

However, in the IT department, even experienced people may get used to the “way things are done” and forget the risks that are building ahead. New hires may replicate patterns they find in the current architecture without realizing they are no longer desirable, increasing technical debt. You now have to let everyone know they should stop adding features to the existing architecture and how to use the new one.

Therefore, discuss the issues with the existing architecture frequently and at various levels of the organization, highlighting the risks lurking ahead and pointing to the proposed solution.

To create an organizational-wide awareness of the architecture’s problems, IT leaders at all levels should use any opportunity to talk about the situation and how to solve it. They should use accessible language, not technical jargon, and exemplify the issues with well-known events such as unplanned tasks, failed deploys, delayed releases, and customer-facing outages.

IT leaders can talk to team members about why the organization is starting an architectural revolution, the vision of the new architecture, and how they should achieve it. At this level, patterns such as **Architectural Vision** and **Pave the Road** can help the team envision the final destination and a path showing how they should proceed. Onboard training for new hires is an opportunity to show the legacy application’s limitations, why they should not replicate its patterns, and how the new architecture will work. This training can be offered to experienced team members as a refresher—do not assume that everyone in IT knows what is happening.

At the highest levels, IT managers interact with business representatives, such as POs and other leaders. In this context, IT should show how the legacy application affects the teams’ performance, causes release delays, and impacts service availability. Leaders at these levels will also define and prioritize broader architectural activities.

At the top level of the organization, the highest ranked IT leader (usually the CTO) can discuss with executives the broader impacts of the current architecture. The CTO may have to explain why the architecture has become so degraded that it now needs such a profound reformulation. Once this is settled, and the need for the architectural revolution is agreed upon, leadership will discuss how to proceed. The debate should include which resources will be allocated and which initiatives the revolution can and cannot impact.

Recurring meetings, such as strategic planning, budgeting, and board meetings are opportunities to create awareness about architectural issues and negotiate priorities. Organization-wide events can also be good moments for reporting the progress of the revolution initiative and engaging the teams.

* * *

Continuous Awareness Building provides opportunities for teams to review the consequences of their past decisions and discuss best practices for the future. Teams and their business peers become more aligned about the current issues with the architecture and what should be done to mitigate them. It reminds the organization that architectural choices can impact the future and prompts everyone to be mindful of tradeoffs in subsequent situations. As a result, using this pattern properly makes teams less likely to need another revolution initiative.

Conversely, **Continuous Awareness Building** requires significant effort to apply properly. As the potential leader of a revolution initiative, you must constantly look for opportunities to talk about the software architecture and its limitations. To keep everyone engaged, you must look for new and practical situations to use as examples of what teams should do and avoid. You should be able to convey your message in a clear and accessible way so that it sounds natural and not repetitive. It also can be challenging to bring IT teams and their business peers together to discuss technical topics, given they are usually overloaded and want more autonomy.

Related Patterns

Creating awareness is much simpler and more natural if everyone in the organization already experiences the impacts of the legacy application, a consequence of **All in the Same Boat**.

Watch out for how the architecture evolves during the early days after starting **All in the Same Boat**. Start defining and collecting **Metrics for Baselining and Comparing**, as they will give you concrete measurements of architectural quality. If, after careful observation, you believe that teams in the current setup need to make more progress toward the desired architecture, this is a sign that you will need to create or increase awareness about this situation.

As the leader of a revolution initiative, while you share architectural risks with your peers or the organization's board, you may be asked about your plan. It is important to have **A Plan up Your Sleeve** in these moments. You can only apply **Continuous Awareness Building** for a limited time before introducing your plan. Otherwise, you may be accused of **Crying Wolf**.

You may have to **Scare the S*** Out of Them** to build awareness in the organization. During the continuous persuasion process, you may need to scare them multiple times, using different arguments or variations of the same line of reasoning.

Building awareness will be more effective if it points to an **Architecture Vision** that addresses the problems the teams are facing. Also, the clearer this vision is, the easier it will be for the organization to understand what must be done to reach the destination. Teams will be much more aware and engaged if they know the benefits of the change and how to achieve it.

By suggesting **Baby Steps** when moving to the new architecture, you can increase awareness because the feedback loop will be shorter and teams will be able to see quick results.

With **Pave the Road**, you reduce the energy teams have to invest in starting the change and consequently enhance their engagement. It can make the path to the **Architectural Vision** clearer, which in turn helps spread understanding of the final destination.

Sometimes a ***Wake-up Call*** is helpful to reinforce the continuous building of awareness. As the Director of IT Operations from one of our case studies used to say: "We should never waste an outage." You can use an ***Elevator Pitch*** to share your views and concerns with peers informally.

Continuous Awareness Building can take advantage of ***Persistent PR*** [Manns, Rising]. By constantly reminding teams about the risks of the current architecture and promoting the new one in different opportunities, you do not let people forget about the need for change.

4.3. A Plan up Your Sleeve

Credit: Photo by Kelly Sikkema on Unsplash

The current architecture has many problems, making it difficult for teams to reliably meet the current business demands. Business leaders are beginning to get **All in the Same Boat** as the teams, experiencing the limitations of the legacy application. You have started **Continuous Awareness Building** throughout the organization about the issues and future risks with the current architecture.

Teams have attempted to address the issue but have not been successful. IT and business people are now asking how to solve the problem. You realize that you must start a dedicated initiative to transform the architecture, which will require formal approval.

How do you prepare for the moment you have the opportunity to present a plan and get budget approval for the revolution initiative?

Teams usually know the best way to create new functionality in their domains. Some have tried to tackle the limitations of the legacy application in their contexts. However, the current architecture still burdens all or most teams. Removing these limitations demands a more coordinated, organization-wide approach, and you still do not have the budget or the mandate to relocate people to work on that initiative.

Some teams are under pressure from deadlines, and their only option is to do things the old way. They have not been able to prioritize architectural tasks and keep adding technical debt to the legacy application. With that, they increase the backlog of the architectural revolution.

The teams already discuss ideas for tackling the problems they face daily. They feel that change is in the air and are getting motivated to work on revamping the legacy application. Some teams are already taking matters into their own hands and starting ad-hoc initiatives for migrating their components to a new architecture. However, they need an overarching plan or a budget to take this through.

Meanwhile, some business leaders are becoming concerned with what they see and are asking how to address the situation. They want to release products faster, more smoothly, and with fewer dependencies on other initiatives. Although they empathize with the problem, they primarily focus on upcoming product features and releases. They are concerned about how this

architectural revolution might affect value delivery in the organization and hurt the business, especially in the short term.

Therefore, from early in the planning stage of the architectural revolution, have a plan ready for pitching the initiative, including a vision of the new architecture and a rough roadmap.

As you talk to teams about the issues of the current architecture and suffer through outages and painful deploys of the legacy application, it becomes clearer and clearer what could be done to solve or at least remediate some of the problems. Some team members have been researching more modern technologies and can suggest designs for the new architecture. Carefully collect those ideas, as they will become the foundation for the **Architectural Vision** that will guide the revolution. Your plan should include an outline of that vision so that it will inspire and motivate teams.

Some teams may have started change initiatives, but uncoordinatedly, competing with other activities in their backlogs and with little or no budget. Do not try to control or standardize those initiatives first; instead, ask what they have learned and enlist them as potential pilot projects for the overall architectural revolution.

Typically teams are under pressure to meet tight deadlines to release new products. These teams have been struggling with their technical debt and have no alternative but to keep adding functionality to the legacy application. This situation increases the risks of the current architecture and creates more backlog for the revolution initiative because this code will have to be migrated to the new architecture, no matter how recent.

Thus, consider including in your plan a mandate to stop developing features in the legacy application and to establish that new functionality will only be allowed in the new architecture. Expect complaints from business leaders because, at first, their projects will take longer to complete. Some of these initiatives may be urgent, so suggest a process to decide which new functionality will be allowed to be created or evolved in the old architecture.

While your plan starts to take form, you will notice that budgeting is a critical element. Architecture migration may increase overall infrastructure expenditure during the transition process [Richardson], as the legacy application and the new architecture will have to live side-by-side during the migration [Yoder, Merson], duplicating the costs of part of the infrastructure.

The initiative will likely demand new tools (e.g., for continuous integration and deployment, monitoring, and automation in general), more infrastructure (cloud computing or at least more virtual servers), more training (in the new tools and platforms available), and also more people (to configure and deploy solutions without significant impacts to the current priorities).

An outsider with expertise in architectural transformation can help you create a more realistic plan. This person can work as a ***Mentor*** to the leader of the architectural revolution and provide ***External Validation*** for the top-level management about the decision to engage in a revolution. You can also hire external consultants to take part in technical tasks, especially if the current

teams have little expertise in large-scale refactoring or in the technologies used in the new architecture.

Your plan may include a ***Big Jolt***—a high-profile event to showcase the architectural revolution. This event often brings a well known expert into your organization to talk about the new idea and can demonstrate the importance of the initiative and boost the morale of the teams involved. You can invite the expert to speak in the event validating the significance of what is about to start. **Figure 3** illustrates a gathering that marked an important phase of an architectural revolution initiative in one of the case studies.

Figure 3: Event organized to provide visibility to an architectural revolution[2]

As a checklist, your initial plan should include an estimate of teams and financial resources, a list of first steps, and expected results. Do not invest time in a detailed plan. Instead, start with a rough roadmap; focus on creating an overview of the initiative, including the vision of the final destination and relevant milestones along the journey.

Incrementally add more detail to the plan as you better understand the issues and learn from pilot projects. As you dive into each component of the architecture, keep the plan focused on the architecture. Teams may have suggestions for improving their code and tools, however these do not necessarily fit into the revolution. Use **Metrics for Baselining and Comparing** to select only the tasks that can help improve the architectural capabilities you should focus on.

Let the plan grow into a master backlog from which the teams can pull tasks to improve their components. Highlight intermediate results, given that they may bring early wins that will benefit the organization and motivate teams to move on with the initiative. Keep the final vision as an inspiring goal, but remind the teams that it may take longer to reach it.

Finally, make your plan as modular as possible. You may not receive full approval at first, so it is essential to select the appropriate activities to start with. Concentrating on only one area or product of the organization may be a good option if the budget is limited and may also create a

[2] Photograph taken by one of the authors.

bright spot to inspire other teams. Choose some **Quick Wins**, especially those that will be visible to either external customers or internal teams.

* * *

One of the main benefits of **A Plan up Your Sleeve** is the process of creating the plan. The act of planning, especially in an agile mindset, helps prepare for situations you cannot initially foresee. Planning also trains you to pitch the initiative to management because it makes you understand the essence of the plan and what can be negotiated.

On the other hand, a good plan takes significant time and energy to elaborate. You must talk to people from both technical and business backgrounds to accommodate all views. You should also have various stakeholders validate the plan, or you risk facing further resistance. Some people may get frustrated when the plan changes—which invariably will happen—and you should be able to keep them aligned and motivated.

Related Patterns

Having **A Plan up Your Sleeve** is highly recommended during **Continuous Awareness Building**, as team members and business peers may ask you what they should do to improve the architecture.

Ask other leaders to review and agree upon the plan before you **Scare the S*** Out of Them**. Otherwise, you may seem disorganized or even irresponsible. Also, this plan can prevent others from thinking that you are just **Crying Wolf**.

The plan should be guided by **Patterns for prioritizing activities**, given that these deal with properly choosing the order of activities in an architectural revolution initiative.

An **Architectural Vision** should emerge from the hard-fought battles with the legacy system and the research of modern approaches and technologies. This vision may not be complete by the time you present your plan, but it should be clear enough to convey what you want to achieve and inspire the organization to move. Keep it as an **Evolving Vision**, so it is reviewed frequently and incorporates the learning that will happen during the revolution.

You may get partial approval for your plan but not the full budget you asked for. In that case, take **Baby Steps** in the revolution plan: start with a simpler issue that will pay back if solved and prove the new architecture, choose a team that is willing to be a pilot for the initiative and try to make it a bright spot for others.

Agreements on what can be implemented in the legacy application are an important part of **A Plan up Your Sleeve**. They establish the initiatives that exceptionally will not be required to be developed in the new architecture. Other initiatives will have to adapt and consequently may be delayed especially early in the course of the revolution. These agreements can be defined by a committee using **Restrict Changes to the Legacy Application**.

Part of your plan should outline ways to **Pave the Road**, for example, by providing templates, training, policies, and infrastructure elements that set the fundamental environment for transitioning to the new architecture. This is in a sense creating an ***Easier Path*** [Manns, Rising].

4.4. Scare the S*** Out of Them

Credit: Image by ErikaWittlieb from Pixabay

The current architecture has caused many defects, delays, and outages that have negatively affected the organization. Business leaders are getting **All in the Same Boat** with teams, and you, as the IT leader, are practicing **Continuous Awareness Building** so that decision makers are at least familiar with the situation. You have started preparing **A Plan up Your Sleeve** for the opportunity to pitch the architectural revolution initiative.

How can you get decision makers within the organization to understand the seriousness of the issues that everyone is facing with the current architecture?

The only way to change from the mindset of just continuing with the old architecture is to take the matter to the top-level management, including the decision makers, likely the CEO or the board, to explain the gravity of the situation and get priority for the revolution. You feel that this will be an uncomfortable discussion.

To get the attention of management, you will have to show them negative indicators about the software architecture, such as low availability, poor development performance, and hard limits it is about to reach. This is all your responsibility. You may fear being called out for letting the situation become so bad and that this whole situation will damage your career.

It is easier not to rock the boat and lie by saying that the situation is as bad as it is. The temptation is to reduce the emphasis of your argument, because you feel there is a risk you may get fired. However, this is your chance to get the revolution officially started.

Other leaders may not want a radical change because it would shuffle priorities and cause delays in their initiatives. They may have overheard that the new architecture is complex and believe your plan will slow down the whole organization. They will be ready to push back and say you are just **Crying Wolf**.

You keep receiving signals of serious issues with the legacy application: outages, failed deployments, delays in releases, and engineers leaving the organization unsatisfied with their work. You know that the only way to tackle these problems is to change the architecture radically; for that, you need support from the decision makers.

Therefore, when an opportunity arises to have the undivided attention of management to talk about the issues of the current architecture, be as assertive as possible to show them the risks it represents for the organization, both in the short and the long term.

Do not waste this chance. Recall negative events (such as outages) caused by the legacy application, highlight the worst-case scenario in future projections, and pitch the revolution initiative. To be successful in your request for priority and resources for an architectural revolution, you have to make an impact. If you understate the risks or fail to connect with real problems, you may lose your chance to get approval. So be very assertive in your argument. It is important to have **A Plan up Your Sleeve** for the architectural initiative and that you present your plan with confidence; use images and graphs instead of technical jargon, be prepared to be questioned and emphasize the cost of not acting.

The opportunity to scare decision makers may present itself in different ways. You may have it in a pre-scheduled meeting, such as a budget discussion or a prioritization committee since you will ask for financial resources and change some of the current priorities. You can also call a specific meeting for this topic. Whichever is your moment, ensure you have the right audience so the group can decide about approving the revolution initiative. Also, before the meeting, clearly state that this issue will be on the topic list. You do not want to casually present a life-threatening situation for the organization in a meeting without at least some introduction.

During the meeting, you may fear sounding irresponsible for letting the situation get to this point or that the executives may end up **Shooting the Messenger**. You may think it would be better to water down your argument and be less dramatic. *Do not do that*. It is positive to show vulnerability and admit that you should have acted on this issue before. It is also acceptable to point out the difficulties that your team faced which contributed to the current situation: tight deadlines in succession, obstacles to prioritizing architectural tasks, and little time to address technical debt.

However, do not imply that the problem is not yours to resolve; take full responsibility and admit that the quality of the architecture is yours. This is more easily accomplished in a blameless culture [Kim et al.], which might not be the case for your organization. Still, your goal here is to get approval for the initiative, not to find who is guilty of architectural decay.

In your argument, use the language that better connects to your audience. Mention concrete examples to illustrate the shortcomings of the legacy application, such as defects, delays, and outages. Point out real risks, not theoretical ones — use worst-case scenarios if necessary to tell what could stop working or be severely affected in the near future. Be as visual as possible; employ graphs and images instead of text. Avoid technical jargon as much as possible — you do not want to sound like this is a purely technical initiative or a whim from the IT department.

Show your plan shortly after that, restating the need for urgent action. Be prepared to respond to questions and to receive negative comments. Focus on convincing the ultimate decision maker (usually the CEO or the chairman of the board) that this is a life-threatening situation for the organization and that something must be done. If the discussion becomes a negotiation, demonstrate some flexibility and explain that your plan can start gradually and later include more areas of the organization. Leave the meeting confident that you have passed your message and get the mandate to start the revolution, even if in a smaller context.

* * *

As the leader of a nascent architectural revolution, when you **Scare the S*** Out of Them**, you increase the chances of getting approval for the initiative. You can also use this pattern more than once during the course of the revolution to secure more resources or stress its importance. Another benefit of this pattern is to increase management's awareness of the architectural limitations. It creates a memorable event for those with less frequent contact with technical topics—an effect similar to a **Wake-up Call**.

To reach your goals, you can **Scare the S*** Out of Them** only a few times. If you resort to this pattern too frequently, you risk being accused of **Crying Wolf**. You need to choose the appropriate events and arguments to make an impact. Your call for action must also be clear, so management can immediately decide about starting or strengthening the revolution initiative.

It takes courage to use this pattern. Exposing the fragility of the architecture and the risks it represents to the organization is a painful task, especially if you are the CTO/CIO or another IT leader. Depending on the organization's culture, being honest and emphatic about problems may endanger your status as a leader, as discussed in **Shooting the Messenger**. Particularly if you have adversaries in the organization, you also may put yourself in a delicate position by voicing these issues. You may hurt your status in the organization or even be fired.

Related Patterns

Applying **Scare the S*** Out of Them** is more effective and less risky for your reputation if you have enough credit with your peers and top-level management. You can earn those credits by having a record of positive outcomes in your tenure as a leader and building trust through **All in the Same Boat**. Also, you may be in a better position to use this pattern if you are in a "honeymoon period," that is, if you have recently joined the organization or stepped up to a leadership role in IT.

You will have more chances to succeed if you can refer to problems they already know through **Continuous Awareness Building**. This pattern also helps you feel less uncomfortable bringing up this difficult subject—no one can complain that they have not been warned before.

You must have **A Plan up Your Sleeve** before sharing some negative information with the decision-makers in your organization. It would be irresponsible to call attention to such a serious issue without some action plan.

The opportunity to scare may appear soon after a ***Wake-up Call*** because this will help you make a bigger impact. Remember that if you do not emphasize the issues enough, you may not get the support you need. After all, if it were easy to advocate for the architectural evolution initiative, it would have been done before.

It can be challenging to get time to discuss the details of issues with top-level management in an organization. Therefore create an ***Elevator Pitch*** that briefly highlights the problems with an outline of the plan for when opportunities for shorter discussions become available.

A ***Personal Touch*** [Manns, Rising] can help strengthen your argument by sharing issues stakeholders will remember, such as outages, defects, and delays with severe consequences for the organization.

Crying Wolf (Anti-pattern)

Scare the S* Out of Them** must be used with care. You may resort to it a few times to reach your objective of getting approval for an architectural revolution. However, if you apply it too frequently, or use it in situations that do not warrant its impact, your peers may believe that you are **Crying Wolf**. You should avoid this scenario because it will harm your ability to promote a revolution of any further change initiative.

You should make compelling arguments when scaring. If your call for action is too weak or unclear, others may also feel that your point is not so strong and that you are **Crying Wolf**. Likewise, make sure that you are ready to take action shortly after you **Scare the S*** Out of Them**. Otherwise you lose momentum and may have to scare them again, risking being seen as **Crying Wolf**. This is why you should have **A Plan up Your Sleeve** ready to be kicked off right away.

Shooting the Messenger (Anti-pattern)

One possible negative consequence of **Scare the S*** Out of Them** is that the bearer of the bad news may take guilt for the architectural issues and suffer consequences, such as losing status in the organization or being fired.

This outcome is more likely in a blame culture, which punishes those who point out problems and risks. The person scaring top-level management might be even more concerned if they have been in the organization for a relatively long time and participated in or were accountable for architecture-related activities, such as an experienced architect or a CTO.

As discussed in **All in the Same Boat**, organizations should recognize that although architectural gaps are a technical problem *per se*, the context that leads to them is more complex, involving time-critical business decisions, pressure from market competition, and prioritization of work. An organization that understands this situation is more inclined to accept that teams made the best decisions with the knowledge they had and to focus on a solution rather than look for culprits.

However, some organizations do not function that way, and the person trying to approve a revolution initiative may fear for their job in such a scenario. If that is the case, the leader should take a more subtle method to get approval for the revolution initiative. One option is to approach executives individually and show them the risks more bluntly and with less drama. An alternative is to engage other non-technical leaders and have them report it to their managers, letting the issues reach them indirectly. Be aware that if you pick any of these options instead of trying to **Scare the S*** Out of Them**, you will reduce your chances of getting approval for the revolution and the potential allowed budget.

5. Patterns for Preparing and Measuring

As you create awareness of the issues caused by the current architecture and finally get approval for starting the revolution initiative, the natural question to come up next is “How should we do it?” At this moment, start preparing the infrastructure for teams to work on. Engage them by showing them your vision and passing confidence, sharing how you all could achieve that goal. It is also the time to define which measurements will track your improvements and create processes to gather them; these metrics will be important to sustain the engagement you created and keep the initiative a high priority.

Figure 4 depicts the patterns for preparing and measuring the architectural revolution. As in the previous section, the patterns in this group are shown inside the dotted line, and the other patterns from this article are in black font, outside the border. Patterns from our language that will be described in future works are shown in gray. Patterns from related works are cited in gray and italics and are briefly described in **Appendix A**. Relationships between patterns in our language are labeled and should be read following the arrows.

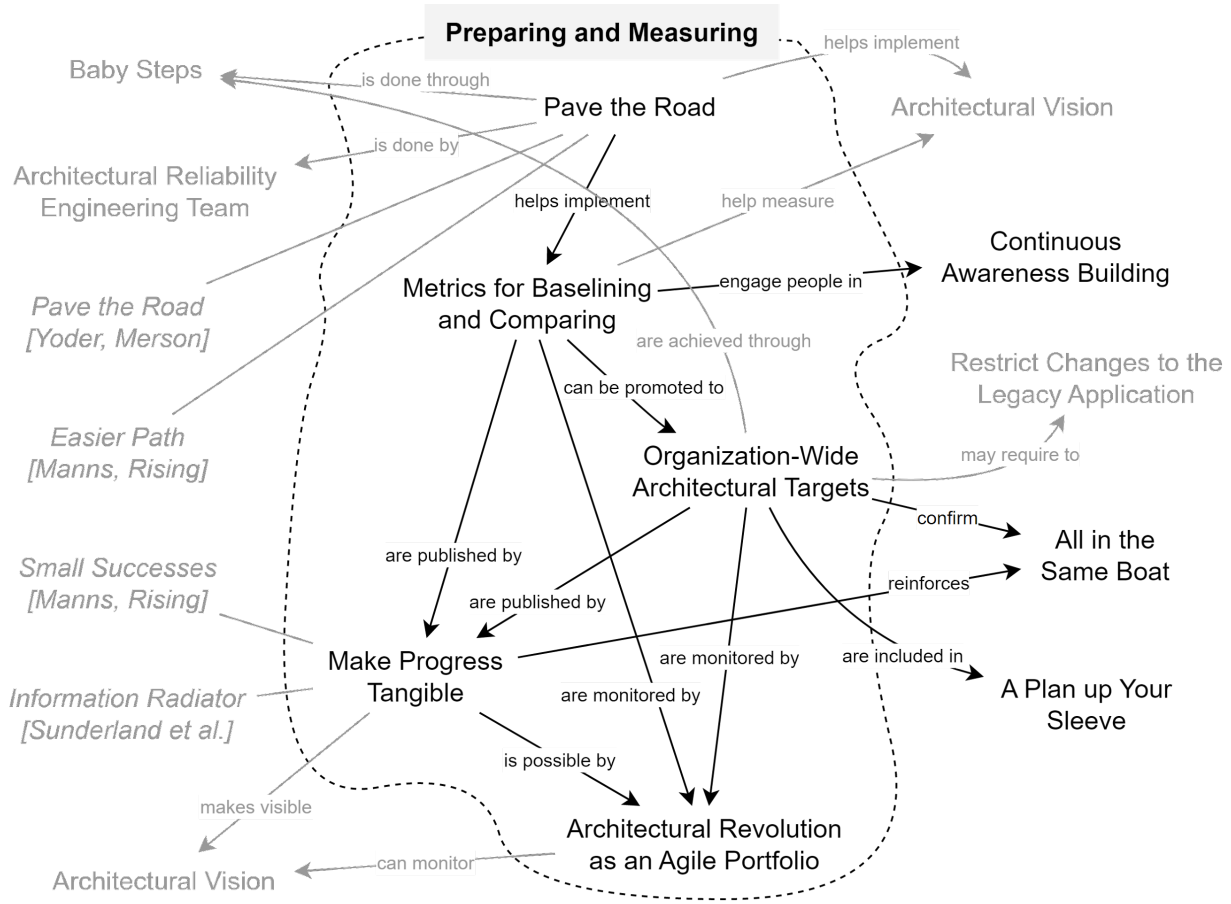

Figure 4: Patterns for preparing and measuring and their relationships

First, it is important to **Pave the Road** by creating the infrastructure and educating the organization about the new architecture. As you start the revolution initiative, it is helpful to collect **Metrics for Baselining and Comparing** the architecture before and after each step. These metrics can serve as a basis for defining **Organization-Wide Architectural Targets**, which is a strong message to the organization that the revolution is a priority. As the initiative evolves, keep in mind that it is not easy to have a clear feeling of the progress in a journey of changing software architecture. **Make Progress Tangible** so that teams and business leaders will be motivated by the advances. As a best practice to have insight and control over the initiative, we recommend that you manage **Architectural Evolution as an Agile Portfolio**.

There are patterns from other works that are related to this group. ***Pave the Road*** [Yoder, Merson] is a particular case of the homonymous pattern in this article. It provides details on preparing the teams to work on a microservices architecture. ***Easier Path*** recommends that to encourage a new idea, you should remove obstacles that stand in the way, which is the goal of **Pave the Road**. When you **Make Progress Tangible**, the teams celebrate ***Small Successes*** and are more motivated. ***Information Radiator*** keeps information visible to all stakeholders and helps **Make Progress Tangible**.

5.1. Pave the Road

Image by Joe from Pixabay

You have shown the problems with the old architecture by getting **All in the Same Boat** and by engaging in **Continuous Awareness Building**. You talked to top management, and you did **Scare the S*** Out of Them** and have secured approval for starting the revolution. You have a vision of the new architecture and have drafted **A Plan up Your Sleeve**.

Teams are about to begin working on the new architecture. However, they are used to the old practices of the legacy application, and some are insecure about jumping into the new way of doing things.

How do you encourage teams and make it easier for the organization to start using the new architecture?

Most teams in your organization work side-by-side with business units to develop solutions. They want to focus on the problem domain and create and deploy applications faster. The new architecture promises to address some of those aspirations, because developing features in the legacy application is becoming more complicated, and deployment is slower and riskier than ever.

However, moving to a new architecture also brings anxiety and risks. Some developers have been working with the legacy application for a long time, and the new architecture brings up some concepts that may be foreign to them. They can be hesitant in using the latest technologies, and if they are not properly trained, they might replicate bad practices from the legacy application.

Other engineers may be excited to start working on a new, more modern architecture. However, each might have a different concept of what it should be. Some expect to be able to try out the new shiny framework they just heard of. If developers do not have some guardrails, teams may make very different choices, turning the new architecture into an incoherent mix.

As with other change processes, team performance may decline at the beginning of the architectural revolution. The legacy application has many issues, but it already contains all the features the organization needs, enabling teams to quickly make small changes to existing functionality. In contrast, the new architecture is far from complete, and creating even the simplest components may require significant effort.

Much of that is boilerplate work, and you believe this could be automated or refactored to a library. This could also reinforce the best practices you want to promote with the new architecture. To accomplish that, you will need one or more teams dedicated to configuring tools and creating reusable components for the revolution initiative.

Therefore, make it easier to develop features in the new architecture by training teams, hiring dedicated people, and providing the fundamental environment for building and deploying applications.

Although you have been sharing the **Architectural Vision**, you need to provide teams with more detailed material so they can feel more confident in using it. Create engaging content and use various types of media at your disposal, such as online documents with samples, reference sheets, podcasts, videos, etc. Organize regular training sessions about the new architecture and use the feedback from the attendees to improve the material and the core components. If the new architecture employs concepts that are common practice in the industry, such as microservices and cloud computing, consider offering external training sessions and stimulating your team to get certifications.

Teams will need to test and explore concepts to become comfortable with the new architecture. You will need to provide separate environments for experimentation and validation before the teams are confident with showcasing their applications to their business peers. Because the teams have to still support the legacy system, you will have to provide infrastructure for both the old and the new architecture.

All this new complexity requires that you evaluate and license tools for helping teams manage new applications and environments. Automation enables you to scale up the number of components and people working in the new architecture without increasing support teams. Some key aspects to be considered are infrastructure as code, data management, build automation, continuous integration/deployment (CI/CD), and application performance monitoring (APM).

You can also **Pave the Road** so that teams can create their business logic faster and with more quality. Once teams start developing using the new architecture, they will be more frequently creating new components and applications (for instance, microservices in such architectural style). Providing templates that enable teams to bootstrap new elements frees teams from repeating the same steps each time and enforces best practices.

If some aspects cannot be codified in a template, you can provide guidelines to orient teams during development and help in code reviews. You can also measure the quality of the code base afterward with **Metrics for Baselining and Comparing**. This enables you to monitor relevant characteristics of the new architecture, compare them with the legacy application, and take action if necessary.

Finally, you will need to provide infrastructure components in each supported environment. This typically includes application servers, databases, queues, data streaming platforms, deployment pipelines, and container orchestrators, to name a few. For example, in a microservices architecture, each application will potentially have one instance of those services. This means

that you will also have to consider automation in infrastructure provisioning; otherwise, operations teams will become a bottleneck. Automated provisioning also allows application teams to be more independent, productive, and compliant with standards. In one case study, we heard from a Director of IT Operations that one of his goals was “to make it easier for the teams to follow the standard than to reinvent the wheel.”

* * *

One of the main benefits of **Pave the Road** is that it creates an ***Easier Path*** for developing applications using the new architectural style. New teams or people can roll out their applications more quickly by learning from the examples, documents, and templates created by the pioneer teams that helped **Pave the Road**.

On the other hand, it requires a lot of time and effort to build software, processes, templates, documentation, etc. Some of these can be difficult, and there can be maintenance issues associated with these items. The initial projects that you use to **Pave the Road** will take longer and require a high upfront investment that will only pay off later.

Related Patterns

Metrics for Baselining and Comparing help define guidelines to **Pave the Road** because they hint at which qualities you are expecting in the new architecture. As you **Pave the Road**, you should be looking for ways to make it easier to achieve those qualities. Also, **Pave the Road** can standardize processes to instrument new components and gather measurements from them.

Pave the Road makes it easier to reach **Organization-Wide Architectural Targets** by allowing teams to focus on their specific goals and leaving the complexities of the infrastructure to a dedicated group. With that potential redundant work out of the way, teams can achieve results sooner and more effectively.

An **Architectural Reliability Engineering Team,** responsible for creating and supporting the core platform, helps to **Pave the Road** because it gives more control over the software stack and standardizes monitoring and other basic services.

As you plan how to **Pave the Road**, look at the **Architectural Vision** and imagine what teams will have to create to materialize that vision. You should consider including in **Pave the Road** all work that seems repetitive, foundational, or unrelated to business problems. Also, if you want to reduce the variation in a given process, consider standardizing it by having **Pave the Road** implement it.

This pattern goes hand in hand with **Baby Steps**. An initial small project might be the pilot project that will shed light on the various new technologies and tools that get to be adopted for cloud development.

If you are considering a move to microservices, consider the homonymous pattern ***Pave the Road*** [Yoder, Merson] for specific details on proven practices on how to pave the road for that architectural style.

5.2. Metrics for Baselining and Comparing

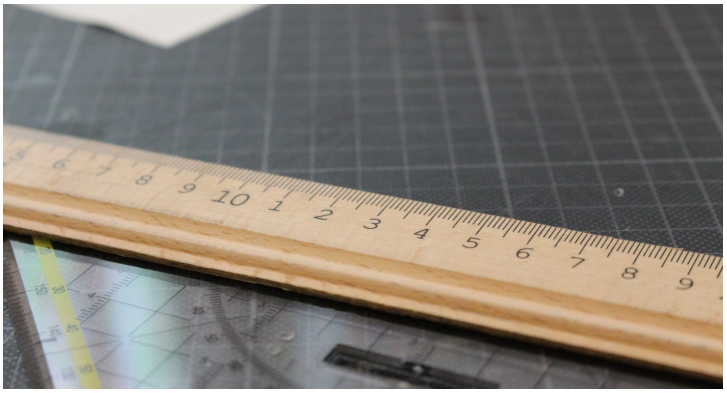

Photo by Christian Kaindl on Unsplash

You have advocated for a new architecture and argued that the old one represents risks for the organization. You have shown many problems while applying **Continuous Awareness Building** and have evidenced the cost of not acting to **Scare the S*** Out of Them**.

The revolution initiative has been approved and is starting up. You want to **Pave the Road** for the teams as they begin working on the new architecture. Given that you have **A Plan up Your Sleeve**, you may have goals related to architectural indicators.

How do you track the progress of the revolution initiative and ensure that the new architecture is improving service quality and team performance?

It was no easy task to negotiate the approval of the architectural revolution. Some of the arguments you had to fight against were that the initiative could delay critical releases and disrupt the organization's services even more. Some criticism focused on the new architecture: it seems more complicated, and some features took longer to be developed in the novel standard compared to similar projects in the old system. Your colleagues in the Finance department complained about the extra budget the initiative will require.

Some teams may be anxious about the change, especially those engineers working on the legacy application for years. Experienced engineers are concerned that less sophisticated teams will replicate in the new architecture some of the bad practices they saw in the legacy application. In general, the teams do not know how to ensure that they are improving the system's overall quality.

Adopting a new architectural style can slow down development for a while. Issues such as more complex deployment, monitoring, and troubleshooting are common, especially when migrating from a monolith to a microservices architecture, introducing more dependencies due to its distributed nature [Newman]. In the medium and long term, however, you know that the benefits can outweigh these obstacles, but you need a way to validate this.

It is not simple to demonstrate that the revolution initiative is progressing without disruptions. Major system migrations can cause a momentary decrease in availability, and adverse events can leave deeper marks in stakeholders' memories. You want the real outcomes of the architectural revolution to be visible to you, your team, and the whole organization. You would

like for decisions to be grounded on factual data instead of anecdotal evidence or mere impressions from stakeholders. These capabilities will give your team the visibility they need to steer through the revolution initiative by receiving quick feedback and addressing issues as they occur.

You may need this information at any point during the architectural revolution to justify the continuity of the initiative or, later on, to look back and show that it paid back. This information can also help adjust the direction of the revolution as it takes place. However, it can be difficult and take a lot of time and effort to set up relevant metrics and get meaningful results.

Therefore, choose a set of metrics for baselining the legacy application and comparing the new architecture against it. Use those metrics to evaluate the progress of the revolution initiative and report them to the organization.

Collect system metrics from the organization level down to the component level and make teams responsible for their respective outcomes. Report the more general metrics regularly to the whole organization, explaining what they mean and what improvements the revolution initiative brings for the organization.

At first glance, there may be many options to pick from, so it is important to prioritize. First, prefer metrics that reflect the perspective of end users. Usually, these will be your customers, so select activities most critical to them and try to collect information close to their point of view. For instance, measure response times in the user interface rather than in application servers. Depending on the attribute you are measuring, end users can also be internal users (for example, from Operations or Customer Support) or even developers, testers, system administrators, and other members of the teams.

Second, prioritize metrics that address concerns voiced by your stakeholders. This will help dismantle their arguments against the revolution initiative and increase empathy toward it. Identify those concerns in the criticism you received, create processes to extract the data, and ensure that you have the same view as your peers—you do not want a debate about which number is correct, yours or theirs.

Third, pick indicators that can be obtained with reasonable effort. If it is too costly or takes too long to get the data, you will probably stop doing it along the way. Automate data collection as much as possible to avoid error-prone manual processes and reminders. Maybe you will have to modify the legacy application to generate information that is not currently exposed; this is better than scraping multiple logs or having an unreliable result. Beware of user-generated information such as customer satisfaction indices. Those are costly to obtain, cannot be updated frequently, and might be influenced by other factors not related to the systems you want to evaluate—a user dissatisfied with the cost of your service may give negative feedback even if the systems performed correctly.

Lastly, select only metrics that can trigger concrete actions. If you cannot do anything about a certain indicator, there is no point in having it. Also, prefer the ones whose actions are within the scope of the revolution initiative. This enables you and the teams involved to be autonomous in the resolution of the root cause of the issue. If you depend on another department or vendor to

solve a problem that is impacting the architectural revolution, you may have to review your plan and include them in the initiative.

The following table includes some sources for metrics that proved to be very useful in our experience:

Class of metrics	Discussion	Examples of metrics
Reliability	A measure of operational performance, indicating how well services meet user expectations regarding availability, latency, performance, and scalability. Should be measured from the end users' perspective and preferably incorporated in a single metric	● *Application Performance Index* (Apdex) [Sevcik]
DORA	Identified by a program called *DevOps Research and Assessment* (DORA) [Forsgren et al.] that analyzed several DevOps practices and their benefits	● Deployment frequency ● Lead time for changes ● Mean time to recovery ● Change failure rate
Lean	When applied to portfolio management, preferably with an agile perspective, provide good insights into development performance	● Lead time (for delivering a new feature) ● Cycle time (within your development pipeline) ● Throughput (features delivered per time)
Source code	Extracted from the source code via static or dynamic analysis, may address the concerns that bad habits from the legacy application may trickle into the new architecture and provide good quality indicators	● Cyclomatic Complexity ● Number of Methods ● Number of Attributes ● Fan-out ● Fan-in ● Afferent Couplings ● Efferent Couplings
Codebase activity	Monitoring and comparing the activity in the codebases (repositories) of the new components versus the legacy application offers good insights into whether the new architecture is being used	● Number of commits ● Number of new lines of code ● Number of distinct committers
Scalability	The ability to handle increased workload without adding resources to a system or doing so by repeatedly applying a cost-effective strategy for extending a system's capacity [Weinstock, Goodenough].	Under increasing load: ● Response time ● Latency ● Availability ● Cost of operation

Cost	The total of expenditures needed to develop and operate the system, often referred to as Total Cost of Ownership (TOC) [Giray, Tüzün], including software development and maintenance, licenses, hardware, cloud infrastructure, Software-as-a-Service (SaaS), personnel, etc.	Examples of cost classes: ● Fixed costs ● Variable costs ● One-time costs ● Recurring costs ● Capital costs (Capex) ● Operational costs (Opex)

When comparing the old and the new architectures, keep in mind that you should compare both realities from the perspective of end users. For instance, using the number of incidents by itself as a metric may not be a good idea. In a microservices architecture, you may end up having more incidents in general (given that you register one incident per component affected), but the impact for the end user may be lower than in a monolithic architecture. This is true if you are using fault tolerance techniques for graceful degradation [Herlihy, Wing], such as asynchronous communication, batching, back-pressure, and other techniques related to reactive development [Allen].

Avoid comparisons that do not isolate external factors. For instance, a component that makes requests to an external service will likely have a higher response time than another that only makes internal requests. If you want to look at the Apdex of both components in the same dashboard, you may have to tweak the parameters, using higher values for satisfied and tolerated response times of the externally dependent component.

Consider that there is a period of adaptation to the new architecture and that teams may become less productive soon after the change; that is, expect a J-shaped curve in productivity. The same happens to availability, given that the rollout of the new components may require some downtime in the legacy application. Thus, remember that early measurements may not reflect a stable reality. If you report them too early and without the proper consideration, you may do a disservice to the revolution initiative. It is better to withhold reporting until the measurements are stable than do **Cherry Picking** and report only those metrics that already show some improvement.

Resist the temptation of comparing different teams in project-related metrics such as lead time to deliver new features or throughput of releases. Doing that fails to isolate external factors (complexity of the domain, size of the features, relationship with the business team, etc.). Instead, it is usually best to start by comparing a team against itself before and after they start using the new architecture.

It is a good idea to include metrics representing competing forces to ensure that the architectural revolution is balanced. For instance, consider including an availability indicator, such as the Apdex, and a productivity thermometer, such as lead time for new features. If productivity is improving, but availability is suffering, this will tell you that teams are lacking in one important aspect of the revolution initiative.

Metrics should change as the architectural revolution evolves through different stages. At the beginning of the initiative, it is worth monitoring codebase activity to ensure that teams use or at least explore the new architecture instead of implementing features in the legacy system. As

components are deployed in the new architecture, replacing old ones, the most relevant metrics are related to reliability and operability (for instance, change failure rate and mean time to recovery). After typically six months to a year of working with the new architecture, teams become proficient. It becomes relevant to start analyzing productivity metrics such as lead time, throughput, and deployment frequency. While the revolution is in its course and after the new architecture is established, metrics such as reliability, scalability, and cost should remain closely observed and even be required to improve.

With all these metrics, it is possible to have regular retrospectives on the architectural revolution initiative and show how much progress has been made. These retrospectives may include all teams and can be about the whole initiative or specific domains. This is especially important in long architectural journeys to keep everyone motivated. Showcasing these results regularly for the whole organization also helps to justify the continuation of initiative, because there may be pressures to relocate teams prematurely to other product or revenue-oriented undertakings.

* * *

The main benefit of **Metrics for Baselining and Comparing** is visibility: you cannot improve what you cannot see. By defining relevant and objective indicators, you provide a direction so that teams can focus on what needs to be improved. A good set of metrics brings confidence to the teams because they can make changes and be sure they are improving. As a result, they are also more motivated to move on with the revolution. Stakeholders certainly appreciate the insight into the achievements of the initiative.

However there are many challenges when defining **Metrics for Baselining and Comparing**. For example, defining which metrics are more valuable requires insight and alignment across the organization. Setting up the infrastructure to generate, collect, aggregate, and store the measurements represents significant cost and effort. It is also necessary to constantly analyze and report the metrics, as well as consider when it is time to replace some with new ones. During this process, as the revolution leader, you should avoid **Cherry Picking** the metrics that are more favorable to the initiative.

Related Patterns

The organization can select the most relevant **Metrics for Baselining and Comparing** and promote them to **Organization-Wide Architectural Targets**. This strategy will increase the visibility of the selected metrics and make them more likely to improve. However, given that there is a limited number of organization-wide goals, only a few metrics can be upgraded each period.

Defining and gathering relevant **Metrics for Baselining and Comparing** is critical for the revolution's success. Still, this alone will only bring the desired effects if you can **Make Progress Tangible** to the organization. Only then can you reap the benefits of increased engagement, effective prioritization of architectural activities, and raised attention to technical quality.

Although each team should have its own **Metrics for Baselining and Comparing**, a standard core set of metrics is vital for the leader of the revolution to maintain focus and be able to compare across different components. Treating the **Architectural Revolution as an Agile**

Portfolio establishes a consistent dialog with teams, ensures the validity of the metrics, highlights early issues, and helps keep the information manageable for the leader.

The most relevant metrics may appear while you prepare **A Plan up Your Sleeve**, as you collect evidence to support your argument. This feedback may come from positive expectations—"It will be great to have this capability in the new architecture"—or from concerns and warnings—"Yes, but what if the migration affects our availability even more?" Consider both cases when selecting **Metrics for Baselining and Comparing** so that you can address supporters and detractors of the revolution.

Much of the infrastructure needed to collect and report metrics can be laid out while you **Pave the Road**. Having standardized criteria and tools for handling metrics will free teams to focus on relevant tasks for the revolution and assure that manual work on this aspect is minimized or eliminated.

The metrics should be aligned with the **Architectural Vision** that drives the revolution. They will typically measure relevant attributes that new architecture aims to improve, and they can also monitor behaviors related to the legacy system that you want to avoid.

Metrics for Baselining and Comparing can be considered *architectural fitness functions* [Ford et al.]. According to the definition, "an architectural fitness function provides an objective integrity assessment of some architectural characteristic(s)." These metrics should last longer than the revolution initiative and can continue to be used to ensure that new components adhere to the defined architectural standards.

Cherry Picking (Anti-pattern)

Given that you want to protect the revolution initiative, you may feel tempted to select and report only those metrics showing that it is progressing and improving the architecture. Teams and stakeholders may notice this attitude and accuse you of **Cherry Picking**. This situation may work against you, and the initiative reports will lose credibility.

Collect metrics that show the reality of the architecture while the legacy application is still active, and continue gathering information as the new architecture is deployed. If measurements show a degradation in some indicator, be open and transparent about it. Analyze the root cause and define a plan for solving the core issue. If it is too early to consider this a real problem, withhold reporting this metric until you have more confidence, but keep the indicator on your list.

5.3. Organization-Wide Architectural Targets

Photo by Afif Kusuma on Unsplash

The architectural revolution is taking form, with the **Plan up Your Sleeve** refined and serving as a shared team vision. The **Metrics for Baselining and Comparing** are starting to show the initiative's benefits. Some teams are already developing features on the new platform, while others are struggling to get business leaders to change their priorities to focus on architecture-oriented tasks. As an IT leader, you are concerned that not enough people in the organization understand that the revolution is a vital initiative.

How do you make clear that the architectural revolution is a strategic priority and engage the whole organization in the initiative?

Although you have been practicing **Continuous Awareness Building** for a while and have formal approval for the architectural revolution, you may find that teams do not engage with the initiative consistently. Some are eager to start working in the new architecture and port their legacy code. These teams have support from their business peers and adjust priorities to make room for these activities. Other teams keep doing things the old way, adding more complexity to the legacy application. Maybe they do not feel comfortable changing their practices or are pressed to meet tight deadlines that do not allow for extra technical tasks.

It is becoming clear that having a mandate to revolutionize software architecture will not be enough to make it happen. You realize that routine decisions at the team level can overrule a strategic priority. The driving force behind these decisions is not inherently wrong but based on local optimization. Teams are pushed to deliver new features so their business units can make more revenue and beat their targets. There may even be financial incentives behind these targets, so you cannot blame the teams for not cooperating. It is no surprise that removing technical debt has not been prioritized and that architecture did not evolve gradually.

You need a stronger message to mobilize teams and business leaders. You need to show in concrete terms that the architectural revolution is backed up by management and should be prioritized versus short-term initiatives. Given that architectural work is often left aside to focus on tasks that support short-term targets, additional targets tied to the objectives of the revolution initiative might help balance the incentives.

Therefore, create organization-wide targets for the architectural revolution. Use these targets to signal that the revolution initiative is an important strategic goal and to help teams prioritize architectural work.

Setting targets is a common practice among companies of all sizes and sectors for improving organizational performance [McTaggart, Gillis]. Having just a few strategic and inspiring targets is fundamental to making this work, so including architecture-related objectives in such a short list requires a significant dose of persuasion. It is also a test of how much management supports the initiative.

Once targets are defined for the whole organization, the architectural revolution will be in the spotlight and will be in a better position to compete with business priorities for the valuable time of teams. The initiative will be featured in recurring meetings about the targets and other organization-wide events. Use those opportunities to reinforce **Continuous Awareness Building** and to call for action. Mention situations in which technical tasks are not getting priority, and point out best practices with the new architecture.

Having targets related to the architectural revolution does not mean it will have more priority than all other business goals. It just puts the initiative on the same level as other objectives and allows for a more transparent discussion. In some situations, launching a new feature a few weeks later to include some architectural work will not hurt the business. In other cases, however, hard deadlines must be met, such as a regulatory milestone.

The team is the first level of decision; ideally, each team can decide what is best for it. However, even with all the awareness built and the visibility given by targets, it may still be hard to prioritize architectural work at the team level. To mitigate this, try designating “architecture champions” for each team. They can be managers or engineers, but they must be high-level enough, have negotiation abilities, and are directly involved in the team. They should argue for prioritization of architectural tasks, ask for help from their managers when necessary, and warn you about teams regularly lagging in their technical goals.

Some issues cannot be solved at the team level, so propose the creation of a committee, including leaders from both IT and business, to discuss the prioritization of architectural tasks in a more strategic stance. Some patterns in the **Prioritizing Activities** group (including **New Features Go Into the New Architecture** and **Restrict Changes to the Legacy Application**) suggest that, at some point early in the architectural revolution, by default, teams should develop new features only in the new architecture. This committee is also the appropriate forum to decide which initiatives will be allowed to be implemented in the old architecture.

Because an organization can only have a few targets at a given time, choosing which architectural objectives will become organization-wide targets is critical; maybe only one or two can be selected. They have to be most representative of the progress of the revolution initiative, as well as good drivers for prioritizing architectural work versus new features. For that, they should be consistent with the vision depicted in the **Plan up Your Sleeve** and the indicators in the **Metrics for Baselining and Comparing**.

We have seen two main categories of architecture-related targets which are outlined below:

- *Deliverables*. These are milestones in the revolution initiative, such as migrating a critical process to the new architecture or decommissioning a legacy database. Typically, they are listed in the **Plan up Your Sleeve**. They contribute to measuring the progress of the initiative and prioritizing migration tasks.
- *Constraints*. These correspond to architectural attributes that the revolution initiative is supposed to sustain or improve, such as performance, availability, or development productivity. Usually, they are drawn from **Metrics for Baselining and Comparing**. They promote quality in the new architecture and drive reliability and infrastructure tasks.

Ideally, architectural targets should include both deliverables and constraints to have a balance between the two categories. If only deliverables are included, teams might rush to meet the targets and cause disruption as they roll out the new services or compromise the quality of the new architecture. If there are only constraints, progress can be slower than desirable, as teams might be too cautious pushing new services to production.

Just as with **Metrics for Baselining and Comparing**, architectural targets should always be evaluated from the perspective of the end user. Constraints will naturally follow this recommendation, assuming they are selected from those metrics. For deliverables, the same rule applies, but it is important to be aware of indirect dependencies. A process or component should only be considered "done" (that is, migrated to the new architecture) if it does not depend on the legacy application being up to work properly from the end user's point of view. Otherwise, an outage or slowdown in your legacy may impact the new component and people will not trust the revolution initiative if the new architecture can be affected by the old one. A good practice we have seen in our case studies is to only consider a process or component as part of the new architecture only if it still works with the legacy application down for a couple of hours—this was tested in practice during the long deployments of the legacy application.

Companies usually review and redefine their targets in annual or quarterly cycles, and architectural targets should evolve accordingly as the revolution initiative advances. Deliverables will move their focus from one process or component to another, following **A Plan up Your Sleeve** and prioritization strategies defined in **Patterns for prioritizing activities**. Constraints may increase service level requirements in each cycle, raising the bar regarding availability or productivity.

Shorter cycles (per quarter or month) invite you to take **Baby Steps** with deliverables. They create timeboxes that can help prioritize tasks and avoid procrastination, which has probably affected the teams. Another benefit of shorter cycles is that they allow teams to pursue smaller increments in constraints, although they also make targets more susceptible to individual events, such as a significant outage.

Longer cycles (a year or six months) call for more ambitious deliverables, even though intermediate checkpoints are fundamental, following an agile mindset. Relative to constraints, larger timescales better absorb one-off occurrences but may average out much of the improvements made later in the cycle. In one of our case studies, an availability objective had a yearly target and was also split into two separate steps per six-month period.

In some organizations, targets are also tied to financial incentives, such as annual bonuses. In those cases, architectural targets should follow the same standard; otherwise, they will be relegated to a lower level of importance and will not have the priority intended. A small part of the incentive real estate is enough to signal that the architectural revolution is a priority for the organization. We have seen cases in which 10% or 5% of the annual bonus was allocated to architectural targets, sufficient to spark interest in the revolution initiative and give it the visibility and priority it needed. For architectural revolution, financial incentives work not as much as a carrot but as a billboard.

One important caveat with architectural targets is that they should be organization-wide or at least include everyone involved with product development, especially those upstream in the teams' workflow. If only IT people are included, the targets can negatively impact the revolution initiative because they will send the message that it is just a technology issue. Areas that create demand for IT and are not affected by the targets will keep pushing for new features, creating even more conflict in the prioritization of architectural work.

Finally, clearly define their acceptance criteria, especially when financial rewards are involved. In the case of constraints, establish not only the final metrics but also the parameters for the calculation. For instance, when setting a target based on Apdex, ensure the tolerable response time is well-defined. For deliverables, have a precise scope for each process or component, down to the level of screen or action. Also, since the migration of a feature to the new architecture may not be visible to users, it is a good idea to demonstrate it by running the feature with the legacy application down.

* * *

Having **Organization-Wide Architectural Targets** indicates to the whole organization that software architecture is essential for its success. Adequate targets help get **All in the Same Boat** because they create alignment and a spirit of collaboration in the whole organization. They also help teams and their business peers focus on the current, most relevant technical objectives. When an organization regularly sets architectural targets, it maintains constant alertness about technical excellence and is less likely to need another architectural revolution.

However, it takes careful thinking to come up with objectives that help unite different departments. Some preparation is required before technical goals can be promoted to organization-wide targets. If this pattern is applied before a minimal sense of **All in the Same Boat** has taken place, it may backfire. Business people might see this situation as unfair and feel that there is nothing they can do to help reach the targets. Defining architectural targets before the teams have enough experience with the revolution can also be an issue. An early failure to reach a target can affect the trust in the initiative. After technical goals are upgraded to organization targets, they become less flexible, and there is less room for change. Moreover, it can be very challenging to convince top-level management to agree with the targets and include them in the narrow list of organization-wide targets.

Related Patterns

When your top-level management defines **Organization-Wide Architectural Targets**, it sends an important message to everyone involved with product development. The idea is that they are

all responsible for internal attributes of the software, such as reliability and quality, not only for creating features. This message is a powerful signal for getting **All in the Same Boat**.

Organization-Wide Architectural Targets are closely related to **A Plan up Your Sleeve**, given that targets consisting of deliverables will usually correspond to the most relevant milestones in the plan. However, keep in mind that, early in the course of the initiative, teams miss their estimates more frequently because they are not used to the new architecture. So, it can be frustrating to set targets with deadlines before teams get up to speed. Wait until the revolution is mature enough to commit to such targets.

Before committing to targets, it is essential to **Pave the Road** by training teams in new technologies, hiring people to create infrastructure, selecting tools for monitoring and automating tasks, etc. Take **Baby Steps** and try some simpler tasks first and get a sense of how hard it will be for teams to work in the new architecture.

Metrics for Baselining and Comparing offers insight on selecting **Organization-Wide Architectural Targets** because you can define constraints that teams should comply with based on the most critical metrics of the new architecture, such as availability and response time. Do not aim to drastically boost service levels in the beginning of the revolution. Instead, set targets that keep or slightly increase key metrics.

Make Progress Tangible is critical for explaining the concept of and reporting on **Organization-Wide Architectural Targets**. Understandably, everyone involved with architectural targets, including those outside IT, will be interested in following the progress toward the objectives. With **Make Progress Tangible**, you can clearly describe targets to non-technical people and create more interesting reports.

Approaching **Architectural Revolution as an Agile Portfolio** can help you get more accurate information about **Organization-Wide Architectural Targets** and increase your chances of success. A portfolio provides an integrated view of all smaller initiatives within the revolution, with information coming directly from the trenches. Effective portfolio management can spot early issues, and either help teams self-correct or make the situation visible to the leadership.

Architectural targets may help you to **Restrict Changes to the Legacy Application** and signal to the organization that teams should avoid adding more functionality to the old architecture. For example, a target might specify a maximum number of features that could be added to or changed in the legacy application over a period of time.

Objectives and Key Results (OKRs) [Niven, Lamorte] are a goal-setting framework that can create adequate conditions for applying architectural targets. OKRs can help companies to reach their targets by aligning objectives related to transforming the architecture with specific key results that need to be achieved. This alignment helps to ensure that everyone in the organization is working toward the same goals and that progress can be tracked. By setting clear and measurable targets, OKRs can help companies stay focused on their priorities, increase accountability, and improve performance. Additionally, by regularly reviewing and updating OKRs related to software architecture, companies can adapt to changing circumstances and course-correct as needed to reach their targets for transforming the software architecture.

5.4. Make Progress Tangible

Credit: Gladiator (2000)

You have defined **Metrics for Baselining and Comparing** the new architecture against the old and have convinced top management to create **Organization-Wide Architectural Targets** for the revolution. The organization is more engaged than ever, and you feel the teams are **All in the Same Boat**. Now people are asking how the initiative is going and how far the organization is from its targets.

How do you keep the whole organization informed about the progress of the architectural revolution and let non-technical people have a more concrete idea of what is going on?

A lot happens in any architectural revolution initiative. You can see multiple fronts advancing if you are managing **Architectural Revolution as an Agile Portfolio**. However, because most of the changes affect only the code and not the behavior of the system, no one outside IT teams can notice any difference. This can be frustrating to many in the organization, given that now there is so much engagement.

People can become anxious, especially if there are financial incentives associated with **Organization-Wide Architectural Targets**. Often it is not apparent to many in the organization that we are making progress and that our investment in the new architecture is paying off.

Creating meaningful ways to communicate progress to people within the organization can be challenging. This is especially true when communicating with various teams with different roles. These often take a lot of time and effort to develop and maintain.

Therefore, adopt an accessible style for communicating the progress of the revolution initiative. Translate technical concepts to visual or physical representations that anyone in the organization can understand.

Carefully define channels for communicating about the revolution initiative and select a frequency for reporting its progress. For companies whose employees are colocated, boards in walls and shared spaces are simple and effective options that can also help spark conversations and create opportunities to reinforce **Continuous Awareness Building**. A more sophisticated alternative is using monitors for displaying real-time data on metrics tied to targets. These are ***Information Radiators*** [Sutherland et al.] that make information available on demand and signal to the whole organization what is happening.

Besides those “pull” channels, there are “push” options that are helpful for regularly spreading status information—such as the weekly availability index—but are especially adequate for sending one-off notifications, for instance, the completion of an important deliverable. Email is an obvious choice, but if the organization uses an online chat tool, it may be a better choice, given that this is becoming the primary communication channel for many IT teams. All-hands meetings, either in-person, fully online, or hybrid, are also great channels for discussing the status of the architectural revolution. Being in the spotlight in those meetings reinforces the importance of the initiative and further helps its prioritization within teams.

The frequency of communication is an important consideration. Communicate constantly to keep everyone up-to-date with key metrics and relevant deliverables. We recommend having at least one monthly update for the whole organization on the overall status of the initiative. More detailed information, such as a specific metric or the progress of a migration, should be available more frequently—for instance, weekly—to whoever is interested. Team members should be aware of their progress daily, because this should be discussed in stand-up meetings, assuming an agile process.

Constraints can usually be visualized in graphs, preferably superimposing data from the new architecture over the old one. It is also helpful to show metrics trends when only data on the new architecture is available. In an organization that used Apdex as a constraint during its revolution initiative, data was not collected until the beginning of the year when it became a target. The result for January was used as a baseline to define the index to be reached by December, which was then split into two average indices, each one per six-month period, the second being higher than the first. In every meeting about targets, a presentation would show the initial objectives for each month, the results already achieved so far, and the indices that should be reached in the upcoming months so that the averages would meet the objectives.

Deliverables can be shown as a step-by-step process, with the final goal of migrating all the components agreed upon in the current cycle. When reporting results digitally, use a progress line or other sequential display with discrete steps. If the organization’s employees are mostly co-located, it is possible to use a physical representation of the target, which can be more tangible to everyone and create more engagement.

In one of our case studies, the organization employed a large structure with many pieces, each similar to a large Lego block (see **Figure 5)**, assembled in a place with high visibility in the headquarters. Each piece represented one user-related flow that should be migrated to the new architecture before the end of the year, one of the targets of that cycle. There would be a small celebration when a team confirmed that a given flow was working independently of the legacy application. The team responsible for the migration would gather and remove the corresponding piece from the structure. During the year, anyone in the organization could pass by the structure and have an idea of how that target was going. It also created a lot of engagement, especially for the teams involved in the revolution initiative, which were looking forward to their turn to remove a piece from the structure.

Figure 5: Team celebrating removal of a feature from the legacy application[3]

Be creative when selecting the type of media to use in your communication. In one organization we analyzed, the marketing department created an appealing visual identity and selected a light tone of voice for the architectural revolution. This was used in emails and presentations and allowed for fun and engaging materials. That organization also used video resources, recording a series of short interviews with people involved with the revolution initiative, in which they explained the issues with the legacy application and showed how the new architecture was going to improve customer experience. These videos soon became part of the onboarding training for every new employee.

Another organization changing a monolithic architecture to microservices decided to validate the migration of a feature by recording a screencast of the functionality working during maintenance of the monolithic system—this maintenance happened at least once a week. Each video demonstrated how the team successfully migrated the corresponding scope to the new architecture. It motivated the teams and gave them confidence that the new feature implemented with microservices would work even when the legacy system was down.

As the revolution initiative reaches a larger audience, including people more removed from product development, you may need to use analogies to explain some technical concepts. Analogies are far from perfect, so make clear to what extent they represent the idea you are trying to convey and where they fail to correspond to reality.

* * *

When you **Make Progress Tangible**, the architectural revolution's motivations and progress become more accessible to everyone in the organization, increasing the initiative's chances of success. This pattern strengthens the feeling of being **All in the Same Boat** and thus benefits collaboration and teamwork. Constantly communicating with the whole organization about the architectural revolution helps in **Continuous Awareness Building** and avoids roadblocks to the initiative.

[3] Photograph taken by one of the authors.

However, to **Make Progress Tangible** requires significant commitment, in addition to the other activities in the revolution initiative. It demands creativity and skill to communicate technical concepts in an accessible form. Finding the right channels, the frequency of communication, and the appropriate tone of voice are also challenging aspects of applying this pattern.

Related Patterns

When you **Make Progress Tangible** by effectively communicating the status of the revolution to the organization, you reinforce that you are **All in the Same Boat**. You remind business and IT that they have goals in common and their work is paying back. When there are obstacles, everyone can see them clearly and discuss how to solve them.

It is much easier to **Make Progress Tangible** for the organization when you have clear and relevant **Metrics for Baselining and Comparing**. Once everyone understands the metrics involved in the revolution initiative and agrees that they are suitable for measuring progress, communication about the progress flows more effectively and it is easier to come up with the analogies that make it more tangible.

An organization using **Organization-Wide Architectural Targets** can benefit from **Make Progress Tangible**. People can become anxious about the targets, especially because advancements are hard to visualize and sometimes can only be noticed at the end of a long process. Non-technical people may even question the relevance of architectural targets if the concepts are not properly explained. These risks can be mitigated with tangible and transparent reporting.

When you treat **Architectural Revolution as an Agile Portfolio**, information on the initiative's progress is more structured and reliable because it is more aligned with teams. More quality information leads to better status reports, which will help **Make Progress Tangible**. Besides, the portfolio manager can play the central role of reporting the progress to the whole organization.

The progress toward the **Architecture Vision** becomes more visible by this pattern. Given that software architecture is an abstract concept and its foundational components are even more difficult to grasp, when teams **Make Progress Tangible** they are able to motivate themselves and others about moving to the new architecture.

Make Progress Tangible can benefit patterns in the **Prioritizing Activities** group by promoting more frequent and engaging reports on the revolution tasks. For example, you can **Make Progress Tangible** by getting **Quick Wins** with **Baby Steps** toward the **Architectural Vision** Teams get more motivated and perform better when they realize that the organization understands and sees value in the progress they are making.

The best way to **Make Progress Tangible** is by using ***Information Radiators*** because they naturally permeate the workplace, making information easily available and concrete to everyone in the environment.

Celebrations such as the one depicted in **Figure 5** are a form of ***Small Successes***, and help improve the morale of the teams even when most of the time they are working hard behind the scenes.

5.5. Architectural Revolution as an Agile Portfolio

Photo by Jo Szczepańska on Unsplash

You have engaged the whole organization in the architectural revolution, with **Organization-Wide Architectural Targets** showing the importance of the initiative and frequent communication helping **Make Progress Tangible**.

Now a significant number of teams have prioritized architecture-related activities. Some are working exclusively in the revolution, while others have started some tasks for migrating their systems to the new architecture, but priorities come from their respective business units.

How do you track the progress of the architectural revolution across multiple teams and respond to eventual roadblocks?

As the organization grows, many agile teams will be working independently and together with multiple business units. Tracking an initiative that spans many areas of the organization is a complex task, and the architectural revolution is one such initiative. Most teams involved are focused on business goals and will have only part of their backlogs dedicated to the revolution.

Product Owners define the teams' priorities, and although you can influence them through **Organization-Wide Architectural Targets** and patterns for **Prioritizing Activities**, you do not have direct control over them. Because business urges may change the priorities in the backlogs, revolution-related tasks can get postponed, and it becomes hard to get a commitment for the completion of architectural activities.

On the other side, you need to report the progress of the revolution initiative to the organization. If there are financial rewards involved, your colleagues might constantly ask you about updates and expect consistent feedback. Watching the pace of infrastructure initiatives is key to keeping everyone motivated. If business-oriented teams face regular delays in core activities, they may lose their enthusiasm and put the revolution at risk.

Teams working on core components for the new architecture may become bottlenecks. These teams may be dedicated to the revolution initiative, so their backlogs are under control, but multiple business teams may depend on those components to get their work done. Therefore, defining the priorities for the core teams may require coordination across various departments and a broader view of the initiative.

You realize that, in essence, the architectural revolution is a portfolio of initiatives spread across the whole organization. You need a consistent way to know the status of each portfolio item and to be able to take action when necessary.

Therefore, treat the architectural revolution as an agile portfolio. Assign a portfolio manager to the initiative and give them access to all teams involved.

Agile portfolio management [Krebs] is a well-established process that applies agile values such as autonomy, adaptability, and transparency to groups of initiatives within an organization. It is well suited to independent teams working iteratively toward a shared goal. Agile portfolio management can help improve communication across different departments, increase the visibility of potential issues, and highlight hidden dependencies between tasks.

In the context of an architectural revolution, agile portfolio management can help coordinate the work of the potentially large number of teams involved in the initiative and focus on optimizing their interactions. As a consequence, it can enhance the visibility of the outcomes of teams working exclusively on the initiative and those focused on business goals. For the dedicated teams—such as those with the mission to **Pave the Road**—the portfolio helps visualize milestones and uncover dependencies. For the teams working part-time on architectural tasks, it facilitates the detection of delays and other impediments.

Typical artifacts of portfolio management in an architectural revolution are visualizations of the teams' progress toward the initiative goals. In agile practice, a cumulative flow diagram (CFD) conveys information about tasks, such as arrival and departure rates, work-in-progress (WIP), throughput, lead time, and cycle time. A skilled manager can use those metrics to understand the performance of an individual team or a group of teams, detect issues and get insights on how to solve them. A more traditional but very effective artifact is a Gantt chart. It displays a list of activities that teams perform over time and shows when they will reach relevant milestones.

Besides reporting progress toward deliverables, portfolio management can include collecting and communicating other facts related to the architectural revolution, such as the ones described in **Metrics for Baselining and Comparing**. Effective portfolio management can provide more standardization in reporting, including aggregated and comparative views across teams and subsystems. The extra attention helps spot issues at early stages, fine-tune measurements, and take corrective actions.

The Agile Portfolio Manager (APM) role is crucial for the effectiveness of this process. This person is responsible for reaching out to the teams, conducting regular meetings, collecting information to build a consolidated view, and communicating with the revolution's leader. While interacting with the teams, the APM can also help mitigate issues by clarifying ideas, uncovering dependencies, and reordering backlog items. The APM is not a manager in the traditional sense; instead, the APM is a facilitator and a communicator, improving information flow among teams and between teams and management.

The APM can take up some of the responsibilities usually assigned to the revolution leader. For instance, the APM can produce all the communication material about the initiative, including slides for status meetings, live dashboards with metrics and milestones achieved, and reports

on risks, staffing limitations, and other topics. The APM can also facilitate discussions among teams, resolving many issues before they reach the IT leader.

Architectural Revolution as an Agile Portfolio corresponds to the Level 2 (Coordination) of the Flight Levels framework [Leopold] concerned with aligning and integrating the different teams and their objectives toward a common goal, which in this case is transforming the software architecture. This level focuses on establishing clear roles, responsibilities, and communication channels among teams to ensure that they work toward the goal and that any conflicts or duplicated efforts are identified and resolved quickly. At this level, regular meetings and progress updates are essential for keeping teams informed and aligned on the overall strategy and ensuring the architectural revolution is progressing as planned.

* * *

Treating **Architectural Revolution as an Agile Portfolio** helps to detect and remove obstacles when they appear, minimizing their impact. Proper portfolio management for the revolution supports **All in the Same Boat** because the communication necessary to keep the portfolio up-to-date increases alignment and boosts motivation with the initiative. This process also promotes **Continuous Awareness Building**, as the frequent discussions about the initiative's progress make the impacts of the old architecture clear. The work of the portfolio manager provides feedback for adapting the plan to fit the reality of the organization better.

Depending on the organization's size and the architectural revolution's scope, it may require significant effort to manage the initiative as an agile portfolio. Gathering information from the teams involved may demand excellent interpersonal skills, especially in the presence of obstacles. Communicating the status of the revolution is also challenging, given the different audiences with diverse backgrounds.

Related Patterns

As you approach **Architectural Revolution as an Agile Portfolio**, you receive much more insight on what is really going on in the teams. The information bandwidth is much larger than what you would get by just looking at automated reports. This helps you monitor **Metrics for Baselining and Comparing** more closely, understand why some teams may not be reaching their goals, and do something about it.

Architectural Revolution as an Agile Portfolio helps monitor the progress toward **Organization-Wide Architectural Targets**, providing the organization visibility and a more profound understanding. Tracking those key targets should become the main focus of portfolio management during an architectural revolution.

Architectural Revolution as an Agile Portfolio is critical to the mission to **Make Progress Tangible**. Effective portfolio management produces insightful and up-to-date knowledge about the revolution initiative. Information gathered on architectural milestones and new metrics can be summarized and communicated. The APM plays a central role transmitting progress status.

The path to the **Architectural Vision** can be long and involve many teams. Approaching **Architectural Revolution as an Agile Portfolio** improves coordination, creates alignment, and increases the chances of successfully implementing the vision.

6. Putting it All Together

We have introduced patterns that are part of a language for helping with architectural transformation. This section wraps up the concepts introduced in this article and provides additional context on putting them to work. The patterns were presented roughly in the order they would naturally appear in a journey of architectural revolution, but it does not have to be that way. Some patterns may not apply to your organizational context, some may already be in place, and others may be employed in a different sequence.

Although you can use the patterns in many different scenarios, it all begins with some form of an ***Awakening***. You start to notice that the architecture has fundamental issues *and* that you cannot solve this by organically adding technical tasks to the teams' backlogs. Problems may have piled up because suboptimal decisions have accumulated over time or the context has changed rapidly. Examples of sudden change are the number of developers growing significantly, a spike in customers, or the business changing to new directions.

Getting **All in the Same Boat** is a great start if you decide to involve the whole organization in this transformation. Business leaders will experience the challenges teams face with the architecture and will be more cooperative in prioritizing technical tasks. Maybe your organization already has IT and business people working together in multidisciplinary teams. In that case, you have a great head start on the road of the architectural revolution.

Even with IT and business working together, you may need to make the architectural limitations more evident to the organization. You should not assume that everyone understands the issues. Business people may associate them with capacity limitations in IT. New developers may replicate bad examples from the old architecture in their code. To avoid repeating the same mistakes, you should engage in **Continuous Awareness Building**. Use every opportunity to let everyone involved understand what is wrong with the current architecture and show how it can be fixed. Corrective actions may start as simple tasks for paying back technical debt and evolve into more sophisticated ideas as the vision of the new architecture becomes clearer. A positive effect is that the increased awareness can help teams prioritize technical work they have been struggling with.

As you talk about the need to transform the architecture, you may have the opportunity to get more priority and funding for the revolution. For this reason, having **A Plan Up Your Sleeve** from the beginning is essential. It does not have to be complete or definitive. The plan should provide an overview of the expected outcomes, a list of teams that will likely be involved, an estimate of costs, and guidance on some first steps. Make the plan modular so you can get results even if you do not obtain approval for everything, and be prepared to adapt it to the circumstances. While you can move on without a structured plan in less formal scenarios, you should at least have a list of goals to check later if you have achieved them.

Occasionally, you may have to ask management for more budget or priority for your plan. Maybe teams are not progressing as expected because business leaders keep asking for new features, or you may need to hire more people or acquire licenses you did not anticipate. One option is to reach out to management and **Scare the S*** Out of Them**, showing the seriousness of the current situation. If you do this, do not hold back and be emphatic, recalling negative events caused by the legacy application. Practicing this pattern can be uncomfortable

and may put your career in danger, depending on your organization's culture. However, sometimes scaring management is the solution to put the architectural revolution on track.

When you succeed in moving on with the architectural revolution, many teams may start working on it. Most of these teams will do very similar activities, moving their components to the new architecture. They will need some basic infrastructure to deploy and run their services. They also may need training to understand the concepts behind the new architecture. To avoid redundant work and variability, you can **Pave the Road** so that it will be easier for teams to design, develop, and deploy new components in the context of the revolution. Your goal should be to make it easier for teams to follow the standard than to create their way of doing things. You may skip this pattern if your scenario is straightforward, for instance, if you have only one or two teams involved with the revolution.

While teams start working on the new architecture, you should define some indicators that will tell how close you are to the goals laid out in your plan. First, you need to know how the legacy application affects the organization. You must also understand how the new architecture is behaving and whether it is improving the situation. For that, you should choose some **Metrics for Baselining and Comparing** the before and after of the architectural revolution. Select metrics reflecting the end user perspective to avoid bias. Include some indicators that address the concerns of your stakeholders about the revolution disrupting the organization's services. This information can help you show the initiative's return on investment and maintain its priority.

Metrics are a great start, but sometimes people need more incentives to be motivated and dedicated to a potentially long endeavor. Your organization probably has goals for other strategic objectives, such as revenue volumes and the number of customers. Including architectural revolution milestones as part of those objectives will prove that the organization puts the revolution at the same level of importance as those objectives. Convincing top-level management to define **Organization-Wide Architectural Targets** will significantly increase the revolution's visibility and priority among teams. You may not accomplish this at the beginning of the journey, and depending on your culture, you never will.

The knowledge you gather about the status of the revolution will provide more value for the organization if you are able to communicate it effectively. Most of the activities and indicators related to software architecture are technical and cannot be easily grasped by people outside IT. You can share the revolution's status in an engaging and accessible form and **Make Progress Tangible** for the organization. If your organization is small, highly technical, or your revolution will be quick, you may not need to translate the status.

The architectural revolution may involve multiple teams and contain several milestones, some dependent on others. One alternative to handling complexity is approaching the **Architectural Revolution as an Agile Portfolio**. You will have more visibility into the teams' reality, get early feedback on potential issues and increase release coordination. You may not need this step if only a few teams participate in the revolution, and it is straightforward to access everyone involved. Another approach based on more traditional project management methodologies may be preferable depending on your organization's culture.

7. Future Work

We are working on a broader set of patterns related to architectural revolution, including other categories that address different aspects of the journey. In this section, we describe these categories and the main patterns we have already discovered in each.

7.1. Patterns for Prioritizing Activities

This group of patterns deals with prioritizing activities in the revolution initiative. These patterns are important because you want to deliver value as soon as possible. While value here does not mean new features, customers will benefit from a more stable service, fewer defects, and more frequent releases. Stakeholders will also gain from internal characteristics, such as more flexibility, better testing, simpler deployment, and better tooling.

Early on your revolution journey, you may have an **Architectural Vision** that inspires teams to join the initiative and guides them toward the desired goal. This vision will be prioritized on an **Architectural Roadmap**. You should make progress toward the new architecture by taking **Baby Steps** and creating just enough infrastructure to support simple scenarios and validate the vision. As you start migrating to the new architecture, look for **Quick Wins** so that teams can learn fast and be motivated by delivering some results. A frequent discussion in prioritization is whether you should **Freeze** all other initiatives and focus solely on the revolution to mitigate the risks sooner and reap the benefits of the new architecture. The alternative is to **Change the Tires of a Moving Car** and keep other initiatives going. These two options can coexist: you may choose to freeze some parts of the system to migrate them while other product development initiatives are allowed to move on.

Ensure that teams do not create more backlog for themselves by making **New Features Go Into the New Architecture**, even if it is necessary to migrate a larger subsystem to add a small feature. When a certain subsystem is too complex and risky to be migrated without disruption, or using the new version would require some action from external parties (for instance, customers or partners), **Mirror Feature in the New Architecture** and keep both versions working in parallel. It is valuable to have **Teams Decide What to Migrate** because they know best what affects their work. Sometimes the legacy application code is so poor that it pays off to **Refactor then Migrate** a subsystem. Prioritize the improvement of critical and risky parts of the architecture even when they are not being actively changed by having teams **Migrate Critical Subsystems**. Finally, create gatekeepers to **Restrict Changes to the Legacy Application** and allow only top priority initiatives to be implemented in the old architecture and **Reluctantly Improve the Legacy Application** to avoid increasing risks.

This group is closely related to architecture migration patterns found in the literature; however, our focus is not on technical aspects. Rather it focuses on large-scale decisions about how to prioritize the revolution initiative in the context of the organization's project portfolio.

7.2. Patterns for Organizing Teams

This group is related to how you should organize teams to make the revolution initiative most effective. These patterns aim at properly assigning responsibility for activities and minimizing business priorities interfering with those activities. Conway's Law [Conway] plays an important role in all patterns in this group, and you should leverage this in your favor.

A great option to keep engagement and quality high in revolution initiatives is to **Assign Architectural Migration to Feature Teams** instead of having someone outside those feature teams modify components. If it becomes too hard to reconcile feature development with architectural revolution, you may consider creating **Dedicated Migration Teams Within Product Groups**. A **Specialist Migration Team** can help feature teams engage in the architectural revolution by sharing expertise gained during the initiative and pairing with them in difficult scenarios. To keep in check the risks and impacts of the old architecture, have a **Team Owning the Legacy Application**. A general good practice that will help keep the whole organization in the same boat is to have **Teams Mirror Organizational Structure**. A **Architectural Reliability Engineering Team** may provide a standardized infrastructure for all teams creating microservices and a uniform view of how new services perform. Finally, conduct **Architectural Retrospectives** to discuss the new architecture and get insight on the technical challenges teams are facing.

8. Conclusion

Software products may start with the right design but, for many reasons, reach a state in which the current architecture is no longer suitable. If the need for change becomes too urgent at that point, the organization is forced to choose a more radical approach to remodeling the architecture. We call this type of initiative a “software architecture revolution,” and this paper presents patterns we found useful in those contexts.

We describe the patterns from the perspective of leaders because they play a vital role in an architectural revolution initiative. Their influence and seniority are needed to carry out some crucial activities, such as creating organization-wide awareness of the problem, mobilizing teams, budgeting resources, measuring progress, and reporting results.

The patterns in this article discuss the goals of creating awareness about the current situation of the software architecture and preparing and measuring for the change. These patterns help the leader of the architectural revolution make the organization understand the seriousness of the issues, create a roadmap for addressing the issues, get approval for the initiative, engage the organization, and measure results. We will describe other patterns in future work, organized into two more groups: i) prioritizing the activities and (ii) organizing teams for more effectiveness.

9. Acknowledgements

We would like to thank our shepherd Richard P. Gabriel for his valuable insights, comments, and feedback during the PLoP 2022 shepherding process. Richard spent many hours helping us with our paper both before and after the PLoP conference. He also encouraged us to experiment with various ideas to better outline and describe the patterns for our paper. Additionally our PC reviewer, Mary Lynn Manns read and gave us valuable feedback before our writers’ workshop at PLoP. This work was partially done by one of the authors (Joe) while collaborating with USP-IME.

We would also like to thank our 2022 PLoP Writers Workshop Group: Filipe Correia, Philipp Bachmann, Christian Scheller, Pavel Hruby, Paulo Marques, Joelma Choma, and Mary Lynn Manns for their valuable comments and suggestions. In addition, Claudia Melo made insightful suggestions on content and format.

10. References

[Alexander et al.]	Alexander, C., Ishikawa, S., & Silverstein, M. (1977). A Pattern Language. Oxford University Press.
[Allen]	Allen, J. (2017). Reactive Design Patterns. Simon and Schuster.
[Beck et al.]	Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D., (2001). Manifesto for Agile Software Development, https://agilemanifesto.org/.
[Besker et al.]	Besker, T., Martini, A., Lokuge, R. E., Blincoe, K., & Bosch, J. (2018, September). Embracing Technical Debt, from a Startup Company Perspective. In 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 415-425). IEEE.
[Clements et al.]	Clements, P., Garlan, D., Little, R., Nord, R., & Stafford, J. (2003, May). Documenting Software Architectures: Views and Beyond. In 25th International Conference on Software Engineering, 2003. Proceedings. (pp. 740-741). IEEE.
[Conway]	Conway, M. E. (1968). How do Committees Invent? Datamation, 14(4), 28-31.
[Ford et al.]	Ford, N., Parsons, R., & Kua, P. (2017). Building Evolutionary Architectures: Support Constant Change. O'Reilly Media, Inc.
[Forsgren et al.]	Forsgren, N., Tremblay, M. C., VanderMeer, D., & Humble, J. (2017, May). DORA platform: DevOps Assessment and Benchmarking. In International Conference on Design Science Research in Information System and Technology (pp. 436-440). Springer, Cham.
[Fowler]	Fowler, M. (2019, August 1). Software Architecture Guide. Martin Fowler. Retrieved July 12, 2022, from https://martinfowler.com/architecture/
[Giray, Tüzün]	Giray, G. & Tüzün, E. (2018). A Systematic Mapping Study on the Current Status of Total Cost of Ownership for Information Systems. Gazi University International Journal of Informatics Technologies. 11. 131 - 145.
[Herlihy, Wing]	Herlihy, M. P., & Wing, J. M. (1991). Specifying Graceful Degradation. IEEE Transactions on Parallel and Distributed Systems, 2(1), 93-104.
[Kahneman et al.]	Kahneman, D., Lovallo, D., & Sibony, O. (2011). Before You Make That Big Decision. Harvard business review, 89(6), 50-60.
[Kim et al.]	Kim, G., Debois, P., Willis, J., Humble, J., & Forsgren, N. (2021). The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations. IT Revolution Press.
[Krebs]	Krebs, J. (2008). Agile Portfolio Management. Microsoft Press.
[Leopold]	Leopold, K. (2020). Rethinking Agile: Why Agile Teams Have Nothing To Do With Business Agility. Findaway World.
[Manns, Rising]	Manns, M. L., & Rising, L. (2015). More Fearless Change: Strategies for Making Your Ideas Happen. Pearson Education, Inc.
[McTaggart, Gillis]	McTaggart, J., & Gillis, S. (1998). Setting Targets to Maximize Shareholders Value. Strategy & Leadership.

[Melo et al.]	Melo, C. D. O., Santana, C., & Kon, F. (2012, September). Developers Motivation in Agile Teams. In 2012 38th Euromicro Conference on Software Engineering and Advanced Applications (pp. 376-383). IEEE.
[Newman]	Newman, S. (2019). Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O'Reilly Media.
[Niven, Lamorte]	Niven, P. R., & Lamorte, B. (2016). Objectives and Key Results: Driving Focus, Alignment, and Engagement with OKRs. John Wiley & Sons.
[Nord et al.]	Nord, R. L., Ozkaya, I., Kruchten, P., & Gonzalez-Rojas, M. (2012, August). In Search of a Metric for Managing Architectural Technical Debt. In 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (pp. 91-100). IEEE.
[Richardson]	Richardson, C. (2018). Microservices Patterns with Examples in Java. Manning.
[Ries]	Ries, E. (2011). The Lean Startup: How Today's Entrepreneurs use Continuous Innovation to Create Radically Successful Businesses. Currency.
[Schwartz]	Schwartz, M. (2017). A Seat at the Table: IT Leadership in the Age of Agility. IT Revolution.
[Sevcik]	Sevcik, P. (2005). Defining the Application Performance Index. Business Communications Review, 20, 8.
[Spolsky]	Spolsky, J. (2000, April 6). Things You Should Never Do, Part I. Joel on Software. Retrieved August 20, 2022, from https://www.joelonsoftware.com/2000/04/06/↩ things-you-should-never-do-part-i/.
[Sutherland et al.]	Sutherland J., Coplien J. O., Hollander M., Ramos C., Vervloed E., Harrison N., Harada K., Yoder J. W., Kim J., O'Callaghan A., Beedle M., Bjørnvig G., Friis D., Reijonen V., Benefield G., Østergaard J., Eloranta V., Leonard E., Aguiar A. (2019). A Scrum Book: The Spirit of the Game. Pragmatic Bookshelf.
[Weichbrodt et al.]	Weichbrodt, J., Kropp, M., Biddle, R., Gregory, P., Anslow, C., Bühler, U. M., Mateescu, M. & Meier, A. (2022). Understanding Leadership in Agile Software Development Teams: Who and How?. In: Stray, V., Stol, KJ., Paasivaara, M., Kruchten, P. (eds) Agile Processes in Software Engineering and Extreme Programming. XP 2022. Lecture Notes in Business Information Processing, vol 445. Springer, Cham.
[Weinstock, Goodenough]	Weinstock, Charles., & Goodenough, John. (2006). On System Scalability (CMU/SEI-2006-TN-012). Software Engineering Institute, Carnegie Mellon University/.
[Wirfs-Brock et al.]	Wirfs-Brock, R., Yoder, J. W., & Guerra, E. (2015, October). Patterns to Develop and Evolve Architecture During an Agile Software Project. In HILLSIDE Proc. of 22nd Pattern Languages. of Programs Conference (PLoP 2015).
[Yoder, Merson]	Yoder, J. W., & Merson, P. (2020, October). Strangler Patterns. In HILLSIDE Proc. of 27th Pattern Languages. of Programs Conference (PLoP 2020).

Appendix A – Patlet Descriptions of Related Patterns

The following is a patlet summary of the related patterns from other works mentioned in this article. A patlet briefly outlines the gist of a pattern, usually in one or two sentences.

Patlet Name	Reference	Description
Baby Steps	[Manns, Rising]	Take one small step at a time toward your goal.
Big Jolt	[Manns, Rising]	To provide visibility for the change effort, hold a high profile event to showcase the new idea.
Easier Path	[Manns, Rising]	To encourage adoption of a new idea, experiment with removing obstacles that might be standing in the way.
Elevator Pitch	[Manns, Rising]	Have a couple of sentences on hand to introduce others to your new idea.
External Validation	[Manns, Rising]	To increase the credibility of the new idea, bring in information from sources outside the organization.
Information Radiator	[Sutherland et al.]	Collaboratively maintain physical artifacts that keep information visible to all stakeholders.
Mentor	[Manns, Rising]	When a project team wants to get started with the new idea, have someone around who understands it and can help the team.
Pave the Road	[Yoder, Merson]	Make it easier to develop microservices by providing the fundamental environment for creating them.
Persistent PR	[Manns, Rising]	To keep the new idea in front of everyone, consistently promote it in a variety of ways.
Personal Touch	[Manns, Rising]	To convince people of the value in a new idea, show how it can be personally useful and valuable to them.
Small Successes	[Manns, Rising]	To avoid becoming discouraged by obstacles and slow progress, celebrate even a small success.
Wake-up Call	[Manns, Rising]	To encourage people to pay attention to your idea, point out the issue that you believe has created a pressing need for change.

Appendix B – Software Architecture Revolution Patlets

The following is a brief discussion of the entire collection of patterns using patlets in the tables below. A patlet briefly outlines the gist of a pattern, usually in one or two sentences.

Creating Awareness

You cannot assume that everyone in the organization understands the issues with the current architecture, so you must level that understanding by showing practical effects in the product development cycle, the risks the organization is incurring, and what can be done to improve the situation.

Patlet Name	Description
Awakening	Detect and become aware of the problems with the architecture and acknowledge it needs to be transformed with a dedicated effort.
All in the Same Boat	Bring stakeholders together to participate in the daily routine of IT teams so that they understand technical challenges and share responsibility in decisions. Also, have IT leaders participate in strategic discussions, learn more about the organization, and contribute with ideas.
Continuous Awareness Building	Discuss the issues with the existing architecture frequently and at various levels of the organization, highlighting the risks lurking ahead and pointing to the proposed solution.
A Plan up Your Sleeve	From early in the planning stage of the architectural revolution, have a plan ready for pitching the initiative, including a vision of the new architecture and a rough roadmap.
Scare the S* out of Them**	When an opportunity arises to have the undivided attention of management to talk about the issues of the current architecture, be as assertive as possible to show them the risks it represents for the organization, both in the short and the long term.

Preparing and Measuring

As you seek approval to start a revolution, start preparing the infrastructure for teams to work on. It is also the time to define which measurements will track your improvements and create processes to gather them. Ideally, some of these will become targets for the organization. Manage the initiative by treating it as an agile portfolio and reporting its results in a transparent way.

Patlet Name	Description
Pave the Road	Make it easier to develop features in the new architecture by training teams, hiring dedicated people, and providing the fundamental environment for building and deploying applications.
Metrics for Baselining and Comparing	Choose a set of metrics for baselining the legacy application and comparing the new architecture against it. Use those metrics to evaluate the progress of the revolution initiative and report them to the organization.
Organization-Wide Architectural Targets	Create organization-wide targets for the architectural revolution. Use these targets to signal that the revolution initiative is an important strategic goal and to help teams prioritize architectural work.
Make Progress Tangible	Adopt an accessible style for communicating the progress of the revolution initiative. Translate technical concepts to visual or physical representations that anyone in the organization can understand.
Architectural Revolution as an Agile Portfolio	Treat the architectural revolution as an agile portfolio. Assign a portfolio manager to the initiative and give them access to all teams involved.

Prioritizing Activities

You want to deliver value as soon as teams start working on the revolution initiative. Prioritizing which activities to engage first according to the reality of your organization is crucial for ensuring that internal stakeholders will quickly see results from the new architecture. Prioritizing activities can be broken down into “Strategic Prioritization” and “Tactical Prioritization” as outlined below.

Strategic Prioritization Patterns

Patlet Name	Description
Architectural Vision	Define the new architecture in a clear and inspiring way so that teams will know what to expect and be motivated to migrate.
Architectural Roadmap	Create a roadmap for the new architecture which includes when various architectural features should be addressed. This roadmap prioritizes the deployment of the architecture and when to migrate critical pieces.
Baby Steps	Take small steps toward the new architecture. Provide just enough infrastructure to support the simplest cases and validate the vision.
Quick Wins	Migrate simpler, less risky, and less coupled subsystems before you engage in more complex activities so teams can learn about the new architecture and the migration process.
Freeze	Stop developing features while teams migrate to the new architecture to achieve faster results in the revolution.
Change the Tires of a Moving Car	Reconcile architectural migration with feature development to keep the business evolving during the course of the revolution.

Tactical Prioritization Patterns

Patlet Name	Description
New Features Go Into the New Architecture	Avoid increasing the backlog of the architectural revolution by establishing that teams should only develop new features or change existing ones in the new architecture.
Mirror Feature in the New Architecture	Create an improved version of a feature in the new architecture while keeping the original available so that users can compare both versions and switch to the new feature when convenient.
Teams Decide What to Migrate	Let the teams engaged in a particular subsystem decide which components they should migrate and in which order.
Refactor then Migrate	Refactor the legacy application to more easily migrate a subsystem to the new architecture.
Migrate Critical Subsystems	Migrate subsystems that are causing disruption or increasing risks for the legacy application, even if they are not being actively changed.
Restrict Changes to the Legacy Application	Allow some changes to the legacy application while defining objective criteria that must be met to justify each request and assigning a review board to discuss the candidates.
Reluctantly Improve the Legacy Application	Mitigate the major pain points of the legacy application to reduce risk, improve productivity and relieve pressure on the revolution initiative.

Organizing Teams

Team organization is a key aspect in the success of a revolution. An adequate structure helps to keep engagement high and leverages Conway's law to your favor.

Patlet Name	Description
Teams Mirror Organizational Structure	Design the teams communication paths and the IT hierarchy to reflect the organizational structure, leveraging Conway to your advantage.
Assign Architectural Migration to Feature Teams	Have feature teams include architecture migration tasks in their own backlogs, so they will know and own the new components.
Dedicated Migration Teams Within Product Groups	Assign architectural tasks to dedicated teams inside the same product groups as feature teams when migration is more complex.
Specialist Migration Team	Create a team composed by architecture specialists to guide other teams in their migration tasks.
Architectural Reliability Engineering Team	Assign a core team for defining standards for infrastructure, monitoring, security, and other operational aspects that should apply to all components in the new architecture.
Team Owning the Legacy Application	Have one or more dedicated teams own the legacy application and be responsible for improving it, implementing approved changes, and collaborating with migration teams.
Architectural Retrospectives	Conduct regular retrospectives with the technical and business people within the organization to discuss the new architecture and get insight on the challenges teams are facing.