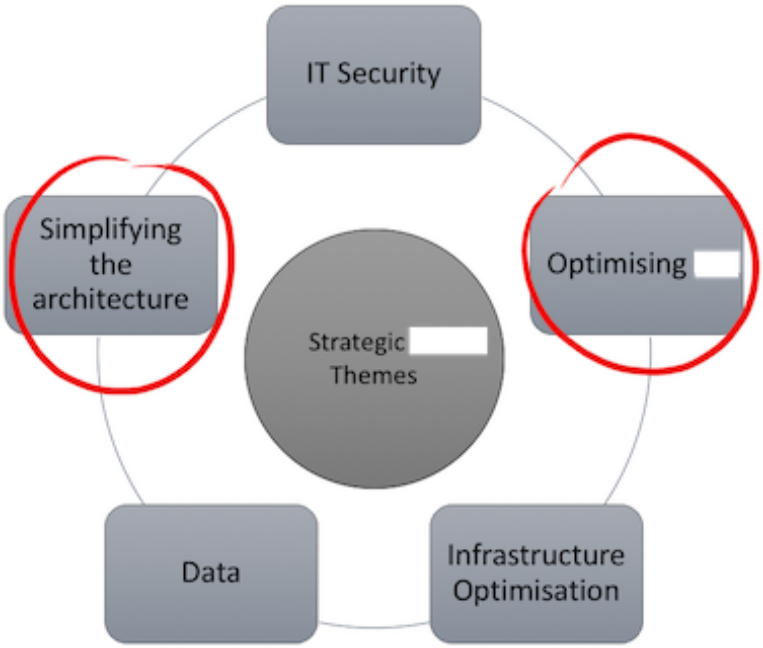
Alignment to corporate strategy using Spine Model

last updated 19 May 2016 by Kevin Trethewey[1]

> *Download a pdf version of this case study* ***here***[2].



Context

Driven Alliance[3] facilitates an initiative for a client which they call the Continuous Improvement Initiative (CII). The client is a bank, and the initiative is sponsored from the centralised IT function. The initiative applies Agile and Lean principles to the way work works across the org, in order to improve (continuously).

This centralised IT function grouped its strategy into five themes. The CII was asked to align their work to these themes. This was done using the Spine Model.

It was felt that the CII would have the biggest impact on two of the five strategic themes, namely *Simplifying the Architecture*, and *Optimising the central IT function*.

This page shows how the Spine Model was used to map out a framework of Values,

Principles and Practices that support each of the two themes.

NOTE: All names have been redacted and some sentences removed entirely for confidentiality purposes.

Theme - Optimising [Division]

Due to the organisational history of [Org], there is a legacy of disparate systems that duplicate functionality. This causes large amounts of accidental complexity, leading to bloated cost of maintenance, repair and enhancement. Some of the systems that when implemented provided a competitive advantage for the bank are now considered commodity and can be bought cheaper than they can be built and maintained. These factors mean there are optimisation opportunities available.

This optimising takes two forms, continuous and discontinuous. We believe the CII plays a central to both forms in [Division].

Discontinuous Improvement

Discontinuous improvement is about strategically changing the organisational structure to a more effective shape.

As consultants we understand the principles of "how work works". We've both studied and applied theories from Agile, Lean, Complexity and Queuing theory in environments like [Org] and have a wealth of on the ground experience.

We can assist in planning and execution of strategic changes to structures to make sure they are well thought through from all perspectives.

Continuous Improvement

Continuous improvement is about the people in the system constantly looking for ways to optimise the way they work, their skills and their delivery.

As coaches we play a consistent role in assisting staff and teams in this "inspect and adapt" mindset. We teach the principles that help people understand their environments better and help them to use their practices and tools more effectively.

We have a strong reputation and track record for doing this within [Org] already.

Values

> *Values are the qualities we believe we should optimise for. They can be used as measuring sticks when deciding how to apply principles, or which path forward to take (eg. will X change give us more predictability or less?).*

Some of the values that could be used to meet the need above are...

1. Predictability

Without predictability it is very hard to improve, or to gain the trust of those people and departments around you. If work and its results are erratic it is hard to know what changes to the system were beneficial and which were harmful.

2. Real world feedback

Without real world feedback about what is actually happening, decisions will made based on assumptions and power structures. They system will be unable correct itself and will consistently make the same suboptimal choices.

3. Adapting to change/feedback

It is impossible to optimise if you are unable to respond the feedback you get, or the changes around you. Without everyone buying into a culture of continual improvement, small incremental change opportunities will not be created and any beneficial changes will be lost due to entropy.

Principles

> *Principles are general truths or propositions that could have numerous ways of being applied. They describe dynamics of how work works in an particular environment. They are commonly expressed as heuristics that provide boundaries within which concrete practices are applied.*

Some of the principles that could be used to optimise for the values above are...

1. Minimise waste

Optimising is really about removing waste from the value stream to make sure as much effort as possible is going towards doing the work that is valuable to the business/customers.

There are seven common forms of waste in software development intensive systems:

1. Partially done work
2. Extra features
3. Relearning
4. Handoffs
5. Delays
6. Task switching
7. Defects

2. Common way of viewing work

In order to optimise for Predictability and Real world feedback, we need to have a way of looking at the work that is understood by business and technical people. Decomposing work into smaller parts should be done by business value, so that the smallest part that is still valuable is what is worked on and delivered quickly.

All tracking and reporting should happen by business value, in a way that clearly shows how close the value is to being delivered. This is done so that all parties can continue to view, understand and give feedback on solutions as they are worked on and delivered.

3. Real business involvement

In order to optimise for Real world feedback and Adapting to change/feedback, business/customers should be directly involved in the teams on a continuous basis, providing feedback and setting appropriate direction. This works best if the person(s) are measured as part of the team, making sure a culture of ‘we are in this together’ is formed and reinforced.

4. Stable capacity

In order for Predictability, and to Minimise waste (see Values above), we need to have stable delivery capacity. Work should be completed iteratively by teams that remain together in order for them to start to predict their throughput and provide more accurate predictions about their ability to deliver value.

5. Design against demand

In order to optimise for Real world feedback and Adapting to change/feedback, it is important to design delivery teams against demand for service from business/customers. Teams that are aligned to a particular business need, and can deliver value directly are more able to adapt themselves appropriately than functional silo teams that are responsible for a specific technology and do not deal with business directly.

6. Single flow of work

In order to optimise for Predictability, work should flow as a single stream through a value chain in a "Pull" flow, where the capacity to deliver determines the amount of new work that is started. This prevents overloading which leads to unpredictability, ineffective prioritisation and waste.

A single flow of work also allows for the value stream to be optimised as a whole, instead of local optimisations happening which can negatively affect the greater stream.

Practices

> *Practices are the structures we use and things we do in order to get work done. To be applied effectively they should be supported by one or more principles that are well understood by all the people implementing the practice.*

Some of the practices that could be used to apply the principles above are...

1. Cross-functional feature teams

Using this practice, long running teams exist to meet a need of the business, and are closely aligned with their customers. The team has a long running identity and capability, even though the people in the teams may come and go over time. These teams are cross-functional, collaborative and dedicated to a business area which provides the team with a single queue of prioritised work.

They have a stable capacity to do work ,which they are familiar with since they have normalised their process and do not commit to deliver more than they have capacity to take on.

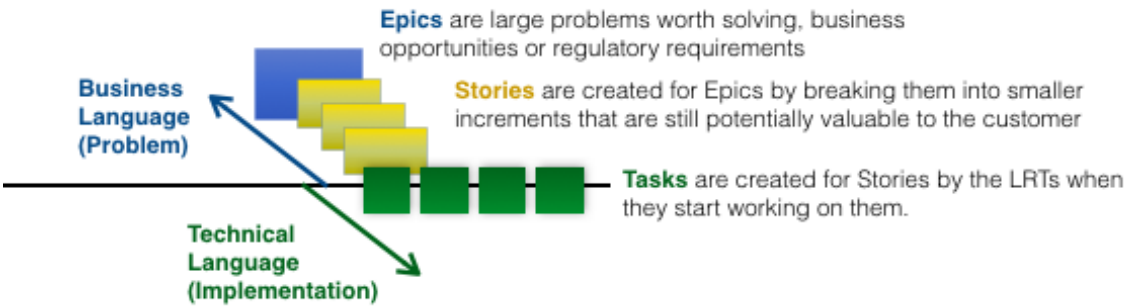
The cost of these teams is fixed, because the team is fixed. They own Stories end-to-end and are measured by throughput - how much valuable work they can deliver over time.

2. High availability service teams

These teams provide expertise to Cross-functional feature teams for technologies or systems that are consumed by multiple teams. They never own delivery of Stories. They either join the feature team whilst they are working on the related story, or for repetitive kinds of work, they provide the recipes, training and oversight for the feature teams to do the work themselves.

The cost of these teams is fixed, because the team is fixed. Charge-back can happen based on ratio of consumption. The goal of high availability services is that they ensure the feature teams are never blocked. Thus, they are measured not on throughput (which is zero) but on availability.

3. Stories (Epics->Stories->Tasks)



User stories should be used to state the business problem, from the perspective of the customer, without prescribing the solution. Typically a user story, once prioritised, can be complete within an iteration.

When many user stories relate to the same overarching business problem, then the overarching business problem can be defined as an epic containing multiple user stories. Like the user stories, the epic is also stated from the customer perspective.

Tasks are the building blocks of a user story, from the perspective of the person doing the work. Tasks are usually technical and not made visible to the customer.

Some practices that are used in conjunction with Stories are...

- *Story mapping* - to understand how Stories link together, from the customer point of view. Useful for discovery and prioritisation.
- *Story points* - to understand the relative size of stories compared to each other. Useful for uncovering misunderstandings in scope, for prioritisation and for post-delivery reflection.
- *Backlogs* - to track flows of work through teams and at portfolio level

- *Planning poker* - to get a shared estimate of story points for a story in a way that allows all perspectives to be considered.
- *Team boards* - to visualise and manage the flow of work through a team.
- *Portfolio boards* - to visualise and manage the flow of work through multiple teams.

4. Iterations

Continuous improvement optimisation works best when there is a cadence. Iterations can be used to timebox the delivery process and repeat it multiple times, each time making it slightly better than the time before.

It also allows for meetings and interactions to be set up long ahead of time so that there are not delays in syncing diaries.

5. Daily Standup

Those interested in the work get together on a daily basis and share knowledge in a way that allows for blockages to be found and addressed, and for the whole team to support each other constantly.

6. Retrospectives

The team leaves their space at the end of an iteration and reflects on how it went, focusing on what they should keep doing, what they should do differently and things they are unsure about. It is a time for the team to come back together and deal with any cracks that may have formed, so that they don’t become major splits.

7. Mobbing/swarming

All those able to contribute to a particular Story work on it together, with the objective of getting it in production and used as soon as possible. An effective way of minimising waste.

8. Cumulative flow diagrams

These diagrams can be used to plot flow of Stories through an area over time, in such a way that useful flow metrics like lead time, cycle time, work in process and average arrival

rate can be monitored.

Tools

Tools are the things we use to get our jobs done, or to automate away the menial, error prone or repetitive parts. Tools should be brought in to assist us implement one or more practices more effectively.

1. Jira

Can be used to track Stories and automatically generate certain reports like cumulative flow diagrams.

2. Physical boards

Used by co-located teams to visualise the work they have in progress so that they can manage their capacity, find blockages and communicate effectively with each other and people outside the team.

Physical boards map the flow of Stories through the team/portfolio, using physical story cards that contain useful information about each story.

Jira has virtual boards, but they do not encourage communication or highlight when teams have overcommitted their capacity like physical boards do. There is also a tendency of teams to constantly groom and refactor their physical boards to match reality that we have not observed in teams using digital tooling.

Theme - Simplifying the architecture

Due to the organisational history of [Org], there is a legacy architecture of disparate systems that duplicate functionality whilst at the same time often integrate heavily, and often directly (point-to-point), with each other. This causes large amounts of accidental complexity, leading to bloated cost of maintenance, repair and enhancement.

Whilst macro level simplification is being targeted through initiatives like [REDACTED], we believe it is imperative to tackle smaller levels in parallel.

Our assertion is that anyone who writes a line of code is, de facto an architect because they are making small design decisions with each line of code they write or change. Without

everyone who writes code understanding the principles of good design and architecture, each small string of poor decisions together weave a binding it is impossible to escape from. A binding that will also trap initiatives like [REDACTED], no matter how well they is implemented in their early stages.

[Org] therefore needs to grow a focus on software craftsmanship, which includes a deeper understanding of the business domain, in the people that are producing and integrating software intensive solutions for the bank.

Values

> *Values are the qualities we believe we should optimise for. They can be used as measuring sticks when deciding how to apply principles, or which path forward to take (eg. will X change give us more Simplicity or less?).*

Some of the values that could be used to meet the need above are…

1. Simplicity

Architects and architectures should be valued for their simplicity, not their complexity. The simplest solution that could possibly work should be tried first, to get cheap, early, safe to fail feedback on what is really needed. Good architecture is always grown from a simple system that works.

It is important to note the distinction between simple and simplistic.

2. Flexibility

Flexibility is important because good architectures do not make all the decisions up front, but rather allow for them to be delayed until the last responsible moment, when you have the most information available to make a choice.

Flexible architectures isolate systems and subsystems in ways that they can be removed or replaced without causing ripple effects into the rest of the system.

3. Automation

Prefer automation for repetitive tasks. Much of the manual work required to check, install, deploy and fail-over systems can be automated and free people up to do the work that

computers are not good at.

4. Virtualisation

Virtualisation is the process of encapsulating a system from the underlying system on which it runs. This allows the encapsulated system to not have to worry about where, how and what the system it runs on looks like.

For server virtualisation this means being able to create, destroy and move around servers directly from software. For application virtualisation this means being able to load balance how many instances of an application are running at a given time directly from software.

Virtualisation allows you to treat systems like cattle, not pets. Creating and destroying them according to the demand in real time.

Principles

> *Principles are general truths or propositions that could have numerous ways of being applied. They describe dynamics of how work works in an particular environment. They are commonly expressed as heuristics that provide boundaries within which concrete practices are applied.*

Some of the principles that could be used to optimise for the values above are...

1. Three Domain Approach

Following a Domain Driven Design approach, systems can be generally categorised or decomposed into three domains; Core, Supporting and Generic.

Core Domain

The parts of the system that are high ROI, high IP, business specific parts - your competitive advantage. These parts should be addressed by Long running team(s) of the best people employed and embedded in the business.

Supporting Domain

The parts of the system that are bespoke, but low ROI and low IP. The parts of the system that need to exist, but you can't just buy them off the shelf. These parts should be addressed by approaches like Co-sourcing, Consultants or leveraging off other parts of the

FirstRand group.

Commodity Domain

The parts of the system that are commoditised, infrastructural and have no competitive advantage. They can bought off the shelf, or gotten from open source or outsource providers.

2. Software Craftsmanship

We believe, and have experienced, that there is a direct correlation between the skills and mindset of those producing software and the successful delivery of software systems. Treating software development as a craft means cultivating and maintaining skills in the same way as you would for other crafts - by providing mentorship and guidance from more experienced craftsmen, and time to practice and improve your craft.

3. Everyone who changes a system is an architect of that system

Anyone who writes a line of code, or changes the design of a system is an architect of that system because they are making design decisions.

4. High Cohesion, Low Coupling

This is a fractal rule that applies as much to high level systems as it does to single classes in an object orientated language.

Cohesion refers to the degree to which the elements belong together, so we should strive for high cohesion and keep all related things together as far as possible. Things that change together should stay together. Things that change quickly and often should be kept separately from things that change slowly or never.

Coupling refers to the degree to which the different elements depend on each other, all elements should be independent as far as possible, that’s why low coupling. If a system needs to deal with another system whose model is undesirable or inapplicable to the model you want within your own system, use an AnticorruptionLayer to translate to/from that model and yours.

Data should be shared between systems rather than behaviour or state, since this creates tight coupling.

5. Deployed systems treated like cattle not pets

Systems (servers, applications etc) should be able to be created and destroyed as required, automatically, from scripts. They should not require massive amounts of custom, one-off deployment and setup in order to run.

Practices

> *Practices are the structures we use and things we do in order to get work done. To be applied effectively they should be supported by one or more principles that are well understood by all the people implementing the practice.*

Some of the practices that could be used to apply the principles above are...

1. Continuous integration

2. DevOps

3. Containerised deployment

4. Distributed version control

5. Test Driven Development (TDD)

6. Pairing/Swarming/Mobbing

7. Automated checking

Tools

> *Tools are the things we use to get our jobs done, or to automate away the menial, error prone or repetitive parts. Tools should be brought in to assist us implement one or more practices more effectively.*

Some of the tools that could be used to support the practices above are...

1. Git

2. Docker + Kubernetes

Links

1. http://www.twitter.com/kevintrethewey
2. http://spinemodel.info/assets/casestudies/AlignmentToCorporateStrategyUsingSpineModel.pdf
3. http://www.drivenalliance.com/