XSCALE Guide

Release 0.1.0-beta

XSCALE Alliance

April 01, 2019

About this guide

XSCALE Alliance is a learning ecosystem of independent coaches and consultancies de-scaling agile organizations through practices derived from

- Hiawatha's Great Law of Peace
- Rikyu's Mu Hin Shu
- Goldratt's Throughput Accounting
- Holmgren's Permaculture
- Stack's Open Book Management
- and Beck's Extreme Programming

Like Linux, we use open source licensing to collaborate on coaching and training material for pattern languages – not frameworks – for Business Agility, Product Management and DevOps.

We're not like the big training frameworks. We're coaches, not trainers, using XSCALE ourselves to collaborate on engagements, collateral and tools.

This guide is one of the results of such a collaboration. In it, we collected patterns and practices that were most useful to us.

We hope this guide will help you to build self-directing portfolios of self-managing streams of self-organizing teams of your existing staff.

We hope this guide will help you to run a self-propagating transformation. Build a slender but uncompromised capability, then grow it.

So that your organization can be like pods of dolphins, not dancing elephants.

Sincerely,

Your friends from the XSCALE Alliance

CHAPTER 2

XSCALE Principles

View article on our wiki

XSCALE is an acronym for the principles of an Agile Organization. Also a language of best practice for Agile organisations focused on exponential growth. As a method for 3rd Generation Agile, XSCALE extends the Agile Manifesto principles to Product Leadership, Portfolio Leadership, Culture Leadership and Holarchy.

XSCALE practices are fully compatible with Spotify, {{SAFe}}, {{LeSS}}, {{Nexus}} and {{DAD}}, and have been applied successfully at scale as a standalone method of generating completely Agile Organizations.

e**X**ponential Return
Simple Design
Continuous Throughput
Autonomous Teams
Learning: triple loop
Ecosystems Thinking

CHAPTER 3

XSCALE Structures

View article on our wiki

{{$:/structures-image}}

CHAPTER 4

XSCALE Roles

View article on our wiki

There are two kinds of Servant Leader role in an XSCALE Organization: Coach and Leader. In general one coach pairs with one leader per the Leadership as a Service practice pattern. Each carries distinct responsibilities to reinforce their squads' autonomy.

Coach Responsibilities

- Determining the Last Responsible Moment for the team to make a decision
- Structuring and scheduling ceremonies
- Quality-assuring the outcome of each ceremony

Leader Responsibilities

- Making the decision if there's no unanimity in the Last Responsible Moment
- Prioritising the input to each ceremony
- Representing outcomes of ceremonies to other teams

Leader and Coach are functional roles, not job titles, so it is possible for someone to play more than one such role at the same time. It is also commonplace that doing so is sub-optimal as it overloads an individual while imbalancing power relationships in their team.

For the specific responsibilities of leaders and coaches per squad see XSCALE Coaches and XSCALE Leaders.

CHAPTER 5

XSCALE Metrics

View article on our wiki

{{$:/metrics-image}}

CHAPTER 6

XSCALE Practices

6.1 YAGNI

View article on our wiki

YAGNI

6.2 Seven Samurai

View article on our wiki

To do

6.3 Leadership as a Service

View article on our wiki

> Power tends to corrupt, and absolute power corrupts absolutely. Great men are almost always bad men. – Lord Acton

The most enlightened civilisations in history regulated the ambitions of their leaders through systems of checks and balances. Perhaps the most successful example is the Iroquois Fire Keepers' Consensus.

> Accurate communication only happens between equals – Robert Anton Wilson

When a leader doesn't serve their team, but instead requires service, team members don't dare communicate accurately with the leader or, for fear of being quoted, with each other. As scaling the responsibilities of leadership eventually overwhelms any individual, leaders are obliged to delegate accountability to subordinates via Command and Control, which restricts teams from responding autonomously to change and so generates Cultural Debt.

Therefore,

Supply leadership only when a team requires it to achieve a decision in a timely manner. To assure this we split decision-making responsibilities between two roles. Here we'll call them Coach and Leader.

1. If the rest of the team are unanimous about a decision, the Leader role doesn't get to decide. Servant leaders lead through influence, not authority.
2. Only when the Coach judges the team can't reach a decision before the Last Responsible Moment - then the Leader decides. The squad abides by any such decision until and unless its other members unanimously overrule it.
3. A Leader can only lead their own squad. For the sake of autonomy, no Leader is permitted to

make decisions for any other team. teams, which are fully capable of representing themselves.

Where a team is obliged to report into a Manager, the decision only constitutes advice from the team. Where teams composit in a Holarchy, there need be no intermediary managers.

Where an agreement must span more than one team, and disagreement or dependency cycles make this inefficient, teams align by Team Representation. To keep alignment relevant to current circumstances, they regularly Brighten The Chain. Real servant leadership is more proactive than simple checks and balances. Its essence is generating consensus.

In order to assure that Leaders are passionately committed to the team's success, they should be treated as Directly Responsible Individuals. This places a leader under tension; on the one hand they represent a specific responsibility; on the other, they're forced to lead through influence rather than authority. To succeed in the role requires vision, passion and empathy - all the attributes of a Servant Leader.

6.4 Throughput Diagram

View article on our wiki

- In Lean Manufacturing, the progress of a Kanban is tracked over time using a {{Cumulative Flow Diagram}} (CFD). CFDs are superior to traditional Agile burndown/burnup charts because they make it easier to see causes of Waste, Stress and Instability.
- To represent Throughput the diagram must account for the time taken up in analysing the Features in an Epic and the Stories in a Feature, not just the delivery and integration workflow per story, and continuously represent the components of Return as well.
- If a Value Stream is trying to hit a calendar date It's important to be able to see by simple visual inspection of diagrams when the feature point budget for a feature has been exceeded by Story Point estimates, and that consequently a release must be refactored.

Therefore,

Feature Points provide a simple basis for this kind of accounting. The cost of an Epic over the life of a release is the sum of its Feature Points. And as each Feature is delivered by just one Squad at a time, the Feature Point cost of a story is simply its Story Points divided by Squad Velocity.

A CFD that represents all these metrics is called a **Throughput Diagram**. The topmost "return" line can represent revenue or an Epic's Critical Number. As with CFDs, Throughput Diagrams can be added together to represent the flows of whole Portfolios.

{{$:/metrics-image}}

6.5 Pirate Canvas

View article on our wiki

Pirate Canvas

6.6 Business Bingo

View article on our wiki

Feature Squads use a simple collaborative estimation method called {{Planning Poker}} to determine the relative effort needed to deliver different stories. These {{Story Point}} estimates can be converted into dollars and time by combination with delivery throughput measurements called {{Velocity}}. But they're not a suitable input to release planning because:

- They don't and were never intended to estimate a Feature Budget. To get at a budget estimate you'd have to multiply them by the Story Velocity, which inevitably and intentionally varies from team to team and, within a single team, from time to time.
- You can't produce story point estimates without breaking down Features into Stories. If you wait until you've done that for all your features, you've stepped out of Agile-land back into Waterfall-land to have an Analysis phase. But if you don't commit to some kind of Feature-level release plan you'll wind up with the Business rebelling because of a lack of predictability, which will also take you back to Waterfall-land.
- There are plenty of costs associated with delivering a Feature that aren't captured by Story estimates, especially to do with Feature Integration Testing, System Integration Testing, Marketing and Opex. These must be taken into account when budgeting Features for a release plan.

Therefore,

Business Bingo is a simple way to quickly determine the budget (time * resource) and relative ROI (Royal Cod) of a feature set without waiting for its features to be broken down into estimable stories. It bases estimates and priorities on the historical costs of a set of features delivered in previous releases and takes costs of analysis and operation into account too. It also provides a simple method to "monetize" Story Points in terms of feature budgets in Feature Points.

Best of all, **Business Bingo** is easy, fun to play, and quickly aligns business and technical stakeholders. It works like this:

1. Write Fibonacci numbers from 1 to 89 on cards and lay them out in a row across a large table. There's nothing magical about Fibonacci numbers - we use them because they consistently lead people to argue in terms of trade-offs - is feature A really as big as feature B + feature C, and so on.
2. Select three previously delivered and deployed features with well documented costs, one small, one medium and one large. Call these probes. Describe each probe in story-normal form commensurable with the roadmap features you want to estimate and represent their complete delivery costs - including documentation, hotfix, opex, the whole nine yards - in terms of Feature Points. Place the three probes under the Fibonacci numbers that match their respective magnitudes in feature points.
3. Pick a feature from your Acceptance Matrix. Compare it with the probes, starting with the middle one, to evaluate its relative size in Fibonacci multiples of feature points.
4. As you add features, sort them into the appropriate Fibonacci column. Continue to compare features this way until there are none left to compare. If the estimators cannot agree on the Fibonacci number for a feature, split it into pieces they can estimate separately.
5. Take features that wind up in the last 3 columns and break them into pieces you can estimate separately. This granularity limit greatly improves the overall quality of the estimates.
6. To estimate relative business value, you simply pick a different set of 3 probes - one for an existing deployed feature that the PO says has low business value, and then one that's critically important to business function, and then one roughly in between. Place them at 3, 13 and 55, respectively, and the rest of the Bingo game runs as above.
7. All Product Squad members collaborate in a 3rd pass to calculate Risks.
8. Record all three numbers on each feature card as input to Royal Cod prioritization.

There's no dollar quantification of the return here, but we've found business stakeholders quickly converge on which features are worth more than which. And the conversations they have in getting to agreement are extremely illuminating - the technical team members need to listen carefully and ask questions to make certain they share the business context.

ToDo: update this description to account for return estimation via {{Pirate Canvas}}.

6.7 Release Refactoring

View article on our wiki

- Release planning works to maximise Product Market Impact and thereby business Throughput by reconciling feature priorities and calendar dates with quantified Release Goals across the Epic Landscape.
- Business, design and technical stakeholders self-organize into Product Squads and Portfolio Councils to collaborate on tradeoffs for this purpose using Leadership as a Service and Open Book Management.
- When solution assumptions fail, market conditions change, new business, design or technology constraints are identified, or other significant learnings show up, feature priorities and release plans must be adapted to them immediately to maintain Throughput.

Therefore,

Release Refactoring is a rapid consensus game that enables quick numeric trade-offs between different feature-sets while respecting calendar dates and budget limits for a release goal.

1. This is a game for the entire Product Squad - business stakeholders make the decisions but design and tech stakeholders question them and answer questions.
2. Using the Royal Cod prioritization derived by Business Bingo, lay out all available Features in columns grouped by Epic.
3. Pick the first column. Pick the feature at the top of the column. Let business stakeholders say whether the Release Goal can be satisfied for this Epic without including that feature. If they disagree, use Leadership as a Service to resolve the matter.
4. Continue down the column, feature by feature, until the business stakeholders agree that one of the features isn't critical for the Release Goal for this Epic.
5. Call all the features above that one **bronze** and all the ones below them **silver**.
6. Now ask the business stakeholders whether any of the silver features could be deferred till the following release without impacting the Release Goal. Call the ones that can be deferred "gold".
7. Total how many feature points are in each of bronze, silver and gold levels for that Epic.
8. Total how much ROI are in each of bronze, silver and gold levels for that Epic.
9. Repeat the above steps for all columns.
10. If you're working to create a release plan for a particular date, determine how many feature points correspond to that constraint. Now simply select the combination of bronze, silver and gold groups with the maximum ROI within that number.
11. Let stakeholders do some fine tuning and trade-offs of features from different epics to respect COD issues. But make certain they start with the bronze/silver/gold tranches so keep this conversation manageable.
12. If fitting to a continuous delivery funding model rather than a specific date, use Getting Features Done instead.
13. Keep the refactoring board on an Information Radiator to assure continuous alignment of authorities and for context for the next Release Refactoring session.

Release Refactoring is played whenever new features are added to the Acceptance Matrix, whenever Sprint Planning indicates a feature's budget is blown, or whenever the PO calls for it. Because this is such a quick game it's also possible to play using set-based Simple Design in an R&D Stream mode to evaluate alternative product plans to evaluate possible responses to changes in market conditions.

CHAPTER 7

Colophon

This is version 0.1.0-beta of the XSCALE guide. It is based on content from our wiki, available at https://xscale.wiki.

This guide is available in the following formats:

- website: https://serra.github.io/xscale-guide/
- pdf
- epub

7.1 License

7.2 Attributions

We've used many open source tools to create this guide, most notably:

- TiddlyWiki
- Sphinx
- PyTiddlyWiki
- Pandoc
- PyPandoc
- Panflute