# Leadership Presentation – Louis Thorp

## Previous Leadership Examples

The following examples are taken from my time at my last employer, Third Bridge.

**Background** – Third Bridge’s line of business and client facing applications were developed over a 12-year period as a monolith in PHP/Symfony/MySQL without following SOLID principles. These systems were not unit testable and suffered from regression overheads as a result. Some Selenium front end testing was already in place when I arrived, however only covered a handful of user journeys. To improve maintainability, reuse and reduce the regression overheads, my team adopted a no new on old mantra, with all new applications developed in AWS serverless with React on the front end. Similarly, non-trivial business logic changes to the monolith were moved to a new services layer, with existing and planned services mapped with the other engineering teams by domain. Lightweight data synchronisation put in place between the monolith's MySQL database and the service layer where required, with the goal of retiring the monolith once the service layer matured.

**Communication** – an upgrade of the monolith from PHP 5.6 to v7.2 was required in order move to a supported version of PHP. This was a non-trivial upgrade due the number of dependencies which were unavailable in PHP 7 and needed to be replaced as well as PHP language changes. A cull of disused code (~500k lines) was performed to reduce the surface area for the upgrade. Close coordination with product, QA and the business was required to identify areas which could be culled, which was in turn validated through the release of deprecation warnings to confirm the same. Over 150 user stories were delivered for the upgrade. QA and regression testing of these in a single big bang release was too high a risk and would have made localisation of any defects slower when compared the more incremental approach which was taken. To this end, tickets were classed by those which could be released to production prior to upgrade provided they were PHP 5 backwards compatible and could pass testing on both PHP5/7. Two thirds of tickets were progressively released into production prior to the upgrade in this way, with the remaining third of tickets being functionally tested, prior to a two-week regression test cycle given their regression testing requirement.

Discovery and grooming of the majority of the tickets required for the upgrade was largely completed one month into the three-month upgrade effort. As initial ticket discovery wound down, it became clear that from bottom up estimation of functional QA and final regression testing required would miss an associated CircleCI/Terraform platform upgrade which was a dependency for the PHP upgrade. The sheer volume of changes required to be tested required a revised approach to the QA efforts, even after drafting additional QA resource from other teams and expanding the number of journeys covered by front end testing with help from an external partner. Planning showed that we would be unable to complete functional testing in time for the infrastructure upgrade and that regression testing could not be completed in the two-week code freeze window in which regression testing had to be completed. I took the decision to redirect developers onto testing as their upgrade work was code complete. The team was motivated to adopt a delivery rather than role based mind-set and concerns about dev testing quality needed to be addressed. Working with dev and QA, we revised our grooming approach to ensure the acceptance tests of upgrade tickets were robust enough for the developers assist with their QA. The upgrade was completed successfully and on time with only single digit production defects.

**Staff development** – A mid PHP developer was hired into my team shortly before my arrival. Through 1-1's and observations, e.g. churn on code reviews, it was clear the developer’s ramp on was not at the trajectory required for his level. This was not helped by his pure PHP background and our move to AWS serverless. I gave this feedback in 1-1’s explaining that his ramp needed to improve in order to pass probation. We reviewed internal levelling guides which documented the expected behaviours of each level of engineer at Third Bridge and I prepared a 30/60/90 day induction plan with objectives to bridge the gaps he was facing. These included such things as Node.js training, AWS certification, the use of tech specs and peering on changes for areas where domain or tooling were unfamiliar. In addition, the team revised existing coding standards and a documented revised approach to code reviews, helping reduce churn on code reviews and improve their effectiveness. These objectives were tracked in 1-1's with a step-wise approach reviewed weekly with the developer. He made good progress and justified his probation period being extended. By the end of this, the he was achieving at the level expected and passed probation.

**Agility** – one issue faced when joining Third Bridge an under resourced product team, with most relatively new to the business and domain. Senior developers often were required to help with analysis as the team struggled to maintain a sufficient pipeline of tickets. Further, requirements where frequently not being missed during user story grooming requiring rework. Team morale was low as this was an ongoing issue several months prior to my arrival. To help mitigate the issue while the product team grew and matured, I took the lead in initial grooming of tickets before wider review/grooming by the team, allowing the senior devs to focus what they did best. Definitions of done for user stories were reviewed with the team and product to ensure that tickets had the correct level of detail to allow dev and QA to progress minimal guidance. New approaches to requirements discovery were taken, with the business being brought into grooming sessions and techniques such as event storming used to thoroughly tease out requirements. Additionally, during sprints and at the start of an epic, a ‘3-amigos’ approach was taken in which the assigned QA, product and dev team members reviewed the associated user stories, caught any late discovery and to ensured requirements were clearly understood by all prior to build and testing. This same group then reconvened once an epic was QA complete for a demo and acceptance purposes. Lastly, epics were assigned a development team feature lead who helped provide continuity throughout delivery particularly for team members who joined the epic after the three amigos kick-off.

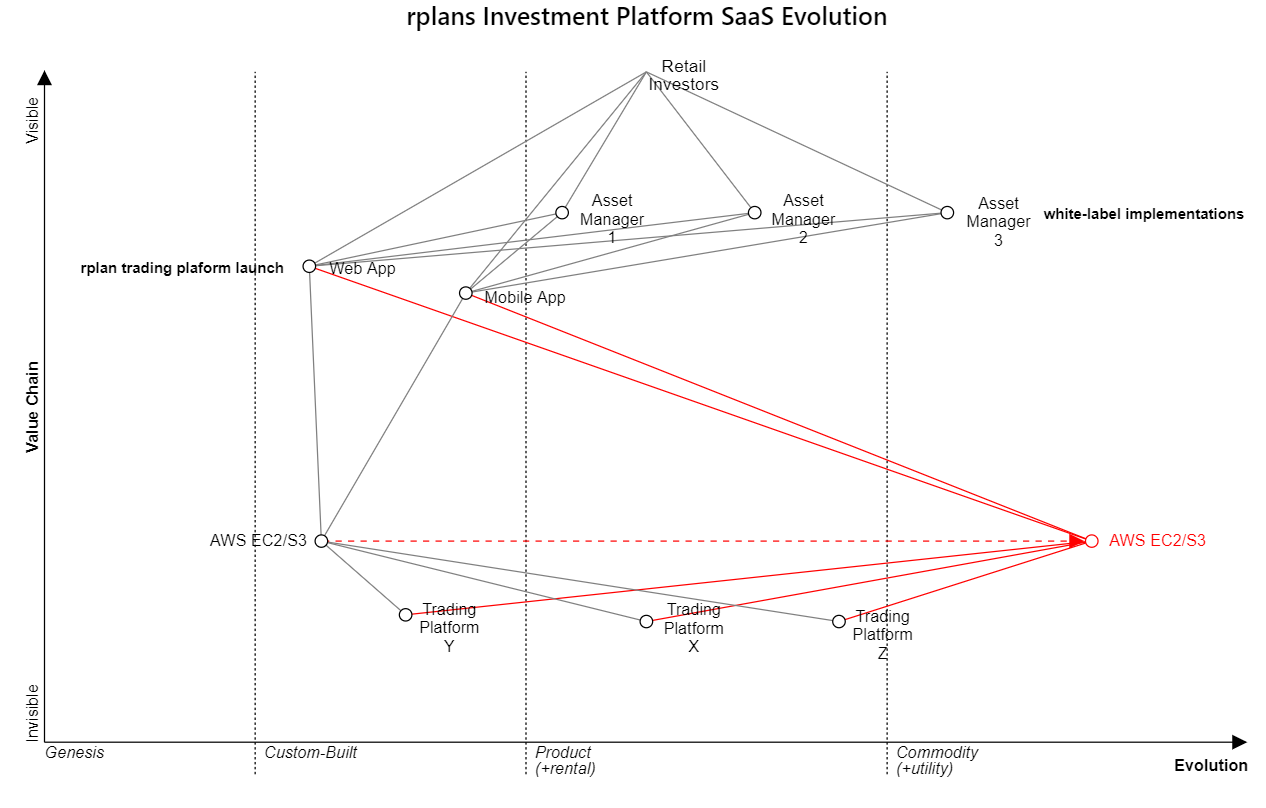
**Planning and resourcing** – my team was regularly missing release dates due to inadequate estimation, planning and user story grooming. Therefore, I moved the team from Kanban to Scrum to improve our short-term planning and overall team processes which served to improve our predictability in the short-term. I worked with my team and the wider engineering teams to agree a lightweight t-shirt sizing approach for the estimation and scheduling of changes in the mid to long term. This allowed teams to provide clearer delivery dates, indicative costs to the business for prioritisation of the roadmap and also helped tease out cross-team dependencies early.

Separate to this, as a wider engineering teams we faced a shortage of PHP dev resource shortage and wanted reduce proportion of contract developers. To help with this, I established an internship and graduate programme which took third year students. These were ramped on our tech stack and core systems before being reassigned to other engineering teams where the PHP resource was needed the most.

**Technical debt** – My team’s support of both legacy and new serverless applications posed challenges with the volume and prioritisation of technical debt which had been incurred historically over quite a period of time. I established an ‘architectural steering group’ made up of the most senior developers in the team. The group prioritised a separate tech debt backlog created for this purpose, performed initial ideation, before review and initial estimation by the wider team to drive arbitrage with product team and business. Agreed tickets were then moved into existing workstreams, groomed and delivered as part of these. This allowed non-blocking pre-existing issues encountered during sprints to be properly prioritised by both engineering and the business without affecting existing commitments. These simple changes allowed reduce tech debt faster, improved sprint success, helped communicate value of the non-functional tech debt changes to product and the business.

## Wardley Map

My previous employer rplan was established to provide a B2C investment platform for retail investors, before pivoting and white labelling their own platform for UK asset managers.



<https://onlinewardleymaps.com/#YiS9JohicaoUwfh6AG>

## Pseudocode Description of a CI/CD Pipeline for a Serverless Function

Assumptions

* CircleCI – for build automation
* Slack build channel (or similar) – for visibility of what being deployed

Trigger build via Slack command using slackbot

AWS Lamba triggered - validate parameters, e.g. target environment

Get AWS credentials from CircleCI credentials manager

Authenticate with AWS

Retrieve secrets via AWS CLI

Get containers from AWS container registry

Trigger build jobs

Run unit/integration tests

If tests successful

Store AMI in S3

Deploy package via Terraform

Slack success notification

Else

Slack failure notification