From Chaos to Kanban, via Scrum

Kevin Rutherford[1], Paul Shannon[2], Craig Judson[2], and Neil Kidd[2]

[1] Rutherford Software Ltd, 26 Hamble Way, Macclesfield, Cheshire SK10 3RN, UK
kevin@rutherford-software.com
[2] Codeweavers Ltd, 11 Imex Technology Park, Trentham Lakes South,
Stoke-on-Trent, Staffordshire, ST4 8LJ
paul.shannon@codeweavers.net

Abstract. Since late 2007 the software development teams at Codeweavers UK have been incrementally improving their ability to deliver motor finance and insurance web services. This two-year journey has taken the company from chaos to kanban-style single-piece flow, including Scrum briefly along the way. This paper charts that journey, showing the benefits gained from a simple "inspect and adapt" cycle in which the teams tackled their biggest problem at each stage.

Keywords: agile, kanban, scrum, continuous improvement, inspect-adapt.

1 Introduction

Codeweavers is a UK business of approximately 20 people, delivering motor finance and insurance web services. The software development team comprises 8-10 co-located developers. At the start of this process Codeweavers was losing money on a monthly basis. In addition to the development team Codeweavers also has a web designer, customer support team, server/database administrator and a sales team - all working with the Managing Director.

The development team predominantly use a Microsoft .NET environment with C#, developing using Visual Studio on Windows.

1.1 Defining the Chaos

In the beginning Codeweavers consisted of two teams of around 4 developers. Each team worked on a separate set of products using completely separate codebases. There was little cross-over between teams and each developer had his own to-do list and worked on tasks independently.

Tasks were given to each developer as the Managing Director and Development Director required, with their respective backlog of tasks being known only to them. Development tasks were split between 50% fixing problems or small changes and 50% spent on new features, speculative work and manual testing.

This attitude to planning and development resulted in no predictability, which frustrated management and clients because deadlines were vague and frequently missed. Stress levels increased following each deployment because limited, manual testing and write first, test later development let bugs slip out to customers unnoticed.

A. Sillitti et al. (Eds.): XP 2010, LNBIP 48, pp. 344–352, 2010.

Developers often needed to ask the Managing Director for tasks to work on when their todo lists were empty - this often resulted in work on speculative ideas which rarely added value to the business. There was a build-up of legacy code as a result and developers had no way to plan or design for the future.

2 Timeline

The state of chaos at Codeweavers worried both the development team and management alike, so ideas were discussed and new avenues explored.

In the final quarter of 2007 the development and planning processes at Codeweavers could be described as Complete Chaos (see §1.1). Development was slow as the low software quality meant "fire-fighting" was consuming development hours and work wasn't being done on the business' core value streams.

> *Response:* After investigating other planning techniques we decided that Scrum would best address our problems so a developer from each team attended a Scrum CSM course with the Development Director. The next day the newly Certified Scrum Masters introduced Scrum to the rest of the team and together we followed the handbook meeting schedule after merging all of our to-do lists into a backlog. We took tasks from emails, Outlook task lists, hand written lists and from memory, and put them into Scrumworks software, sorted by urgency. We used planning poker cards for estimation and had separate planning/estimation/stand-up meetings per team, with a joint retrospective. We also adopted Test Driven Development and Pair Programming using books and online resources to help us help each other.

In subsequent months we varied the sprint length from 1 week to 3 weeks to 2 weeks; we were spending a lot more time in meetings because we were closely following the Scrum methodology; tasks were hidden away inside Scrumworks and the backlog was inspected only rarely. Test driven development was progressing but the initial learning curve was steep.

> *Response:* To address our desire for better coding technique and to get some direction with regard to planning we hired an Agile Coach. We dropped Scrumworks, instead opting for a task board and paper cards. The teams had separate boards, and work was broken down into User Stories written in the "*As a ... In order to ... I would like ...*" form. Each board was organised with columns for "Not Started", "In Development" and "Done"; cards were moved into "Done" when the card's development work was complete and unit tested. We installed CruiseControl on a continuous integration server. Planning poker was dropped in favour of "pair hours" as a unit in planning which shortened meetings. Instead of holding a planning meeting every Tuesday, say, the team opted for a "just in time" planning meeting whenever the "Not Started" column on the board looked nearly empty.

2.1 Development Focus

All these changes enabled the team to regain some momentum and fresh energy. By 2008 Q3 the feeling in the teams was good and our planning and development

techniques were working as developers were happy and user stories were being completed on time.

There was notable tension due to team rivalry as the teams developed different personalities. Support was becoming more difficult because there was too much specialism and knowledge silos in each team. Fundamental knowledge was duplicated between the teams and code was not shared effectively.

> *Response:* We began swapping one developer between teams for a week at a time.

2008 Q4 brought problems with our planning. The User Stories we used on the work in progress board were large, technical, development tasks and were difficult to demonstrate, thereby increasing the length of our feedback loop. Slow progression of stories through the development process was demotivating.

> *Response:* Simplification seemed to be the answer so we moved from user stories in favour of smaller engineering tasks. These promoted better team working and more frequent demonstrations, helping us fail faster and address quality issues earlier. Tasks moved independently across the board, much as the User Stories had done.

2.2 Time to Merge the Two Teams

Swapping one person each week between the two teams was not popular, and did little to break up the knowledge silos. Duplication of design and planning meetings, code overlap and the use of two boards all indicated that two separate teams may not be ideal. Our retrospective meetings, although involving both teams, were too specific so one half of the team would sit silently while issues were discussed between the others.

> *Response:* A major customer had an urgent request that involved just one team's codebase. We decided that the two teams would need to merge to get our customer's urgent requirement delivered in time, and organised the work as a special-case workshop event. With the extra energy brought about by the over-dramatisation a feeling of "all hands on deck" swept through the team and a single focussed development was started. We merged the two teams into one large team with a single board, creating a single plan to swarm on. To ensure further integration each pair consisted of one member from each team, which also helped with knowledge sharing. When the emergency was over the effectiveness of the single team approach was much applauded and the decision was made to continue as a single unit.

Although we'd made regular changes to our planning procedure it was noted that we were still spending too much time in meetings, particularly in planning.

> *Response:* We addressed this problem directly and stopped estimating task duration. Instead we ensured that tasks were designed so that each card covered a similar amount of work.

2.3 A Focus on Quality

By 2009 Q2 the single team was now the standard practise but we still noticed skill silos. A lack of focus emerged as developers had a tendency to refactor legacy code

rather than delivering value. A disconnection arose between development and the business and the team didn't notice the amount of wasted time/yak shaving.

The work in progress board worked effectively for development but visibility up and down-stream was reduced. The board only had 3 columns and once a task was done it was effectively "thrown over the wall". This resulted in partially developed features being deployed, or conversely, a finished feature to be held up for deployment by code under development.

A kaizen activity on our value stream highlighted that most of our stream was never looked at as it wasn't on the task board.

> *Response:* We mapped our value-stream beginning with a customer order and ending with the income following a successful deployment. This helped us to expand the columns on the task board to add in space for internal acceptance, customer acceptance and live. We also included buffer columns between each acceptance state so that we knew when tasks were ready to move.

Tasks were still not getting attention outside the development column as it became clear the current board layout was more environment based that state based. Adding to this we found deployments were not approved for weeks as downstream activity caused a build up with the Product Owner, and the whole process was bottlenecked by approval.

> *Response:* We adapted morning meetings to use a pull system that pulled cards from the right - to help us focus in these meetings we moved the meeting to the right hand side of the board, helping us see the tasks from the customer's point of view. We were then able to take on customer approval tasks providing additional value where normally the product owner would have to take over. We added WIP limits[4] to the board with low WIP limits in the acceptance columns and a limit of 1 in each buffer. When a pair became free they would move to the board's downstream end and search upstream for work. No new tasks were moved to development until all reasonable effort had been applied to customer and internal acceptance.

2.4 Moving to Minimum Marketable Features

The downstream bottleneck into live deployment disappeared. We started measuring mean task flight time at this point and over the next month it dropped from 29 days to 6 days, then stabilized at 10-12 days.

In 2009 Q3 a key member of the management team left the company. This meant our Product Owner had less time to focus on his role within the development team. He was responsible for internally approving completed tasks and the development team handled customer acceptance.

Deployments were still not going smoothly, because quality wasn't seen as a priority. Our 10-stage board meant the feedback loop was very long: occasionally a task would be rejected in one of the downstream acceptance stages and moved back to development; this would lead to partial features in the codebase, and thence to either deploying incomplete features or holding up deployment while waiting for delayed tasks.

> *Response*: To solve the above problems with co-dependent tasks we began batching them into Minimal Marketable Feature-Sets (MMFs). Quality was moved upstream and the team was responsible for approving the quality of an MMF before moving it to customer acceptance. This alleviated the bottleneck with our Product Owner as we employed "corridor testing", demonstrating smaller chunks of work to him when he was available rather than at the end of the feature. The board changed to reflect this and consisted of “Development”, “Done”, “Demo”, “Live” and we removed the WIP limits as the discipline was no longer needed. Individual tasks moved from “Development” to “Done” (which continued to have its old meaning); when all of the tasks in a batch were Done, the whole MMF moved into “Demo”, and then either into “Live” or back into “Development”. At this time we might have 2 or 3 MMFs in flight at any one time, often working on different products.

A bottleneck became apparent when gaps in the flow appeared as the development team felt unable to prioritise tasks or create new ones due to a lack of information, while the Product Owner was still too busy.

> *Response:* The Product Owner attached a value to each task card by adding 1 to 5 dollar signs. The team knew which cards to do next as those with more dollar signs were worth more to the company and helped us work towards the overall goal of making money for the business.

We became reluctant to deploy features due to lack of confidence in quality. MMFs were moving through customer acceptance quickly but our own confidence halted them. Support issues and bugs were not being reduced as we had hoped and time was being wasted combating these issues directly following deployments.

> *Response:* We began to introduce automated regression tests. We thought this magic bullet would ensure that anything we hadn't unit tested would be covered and bugs would be caught before deployment to the customer. It didn't work though, and the amount of time spent on automating these tests was not justified by the benefits. We had "false positives" on one hand and additional work when regression tests randomly failed on the other. Following a retrospective on the subject, we decided that manual regression testing was a better solution as our codebase and understanding was not ready for automation. We created test plans and an overall agreement to do more manual testing was reached.

In 2009 Q4 incomplete MMFs could still get deployed. We were measuring flight time of tasks but not that of MMFs which created pressure to get tasks out resulting in a severe lack of code quality. Task cards could still be co-dependant and there was no improvement in legacy code.

> *Response:* Acceptance criteria and swarming on quality had partially addressed the continued deployment of bugs but this wasn't enough. Our first change was the introduction of Single Piece Flow. At any time the team would have exactly one MMF in flight. Increased slack time meant we had more time for refactoring legacy code and manual regression testing. By measuring waste (recording time spent when our “Blue Lights”[1] were off)

> we identified problems in waiting for deployments so the board was refined to reduce columns; ending with: "In Development", "Done" and "Customer Acceptance". Tasks were moved to "Done" when the feature was complete and the code had been refactored to improve quality but would only move to "Customer Acceptance" when all tasks on the MMF were Done and the whole team was happy with the quality of both the feature and the code; if one person was unhappy we swarmed on the quality and didn't deploy.
> Responsibility driven design[2] through CRC[3] sessions was our next addition which helped test coverage, code quality and design. During each CRC session we recorded interactions between objects and started TDD by testing the responsibilities and interactions on each CRC card in the code. We used mocking to ensure only the methods on the card were under test.

Although pairs swapped daily there were still specialisms within MMFs and new recruitment meant knowledge sharing became a priority

> *Response:* We introduced a "least knowledgeable developer" rule. During daily stand-up meetings the developer who had worked on an area the least would volunteer for that task and they would select a pairing partner based on who had the most knowledge. The least knowledgeable developer would then start the task as the driver in the pair, ensuring no work was done without both developers having a full understanding of the code they were producing.

The "To-Do" column just ahead of development suddenly emptied one day, partly because a potential sale failed. We discovered that the product owner had no process for deciding which of the many other customer requests to prioritise. We introduced a new board upstream of development, representing a cost/benefit matrix to help sort cards for business requests before they were moved onto the development team. Although prioritisation was now adequate on the cost/benefit board for a month, there was no visibility of priorities upstream so the development team didn't know which MMF to bring in next. This became most apparent when the Product Owner was not available due to a spate of sales meetings.

> *Response:* We introduced a new value based hopper board to feed the development board with MMFs. Cards were prioritised by placing high value cards at the top and cards that are ready to be developed to the right. We now knew where effort should be placed to get MMFs ready for development; this information was previously known only to the Product Owner.

A period of frequent changes meant that the development team had stabilised so our process improvements were extended to the server admin and database admin as wasted effort was evident in that team.

> *Response:* A conscious effort was made to move that team into the MMF WIP board. All the database or server admin tasks were added to cards and tied to MMFs. This worked initially and knowledge was shared as developers worked cross-team, however, it was proving hard to maintain as the two conflicting work styles did not coordinate well and we deemed that perhaps the admin team needed an update to their own process first and we should revisit the integration later.

2.5 Increasing Flow

With an ineffective attempt at alleviating bottlenecks around the development team we decided that we'd focus again on the speed of MMFs moving through the development board.

We noticed we were not getting features out as fast as we wanted, waste wasn't being reduced and frustration was building when waiting for builds and deployments, causing a loss of focus in the value stream.

> *Response:* Using one of our whiteboards, we started recording reasons why we were wasting time, and then the amount of time wasted. This enabled us to start swarming on build and deploy times which halved the amount of waste. We also noticed that legacy code was a bottleneck in one particular product - our broker finance application - which was measurably slower to work on as this code had largely been produced in the early days of the company.

Towards the end of development on an MMF we noticed that swarming wasn't as efficient as it should be. It clearly needed an optimisation in organisation and communication.

> *Response:* A second stand-up meeting was brought in at 2pm every day to ensure tasks were worked on as things developed. New tasks were spawned regularly if we thought that a concurrent unit of work could be done on a particular part of the feature. Developers were encouraged to stand up and "pull the andon cord", if problems were found, so that we could all effectively swarm on adding value. A practical problem with the second stand-up emerged in that differing lunch times weren't always compatible with the new meeting time so we arranged for a lunch time alarm to sound synchronising developers.

As sales increased and new customers were added to our roster we noted that support loads increased and profitability diminished because clients needed help to integrate our solution into their websites.

> *Response:* A proactive approach was taken here as we started to go out to clients and help on-site. We also invited any new customers to come to the office to learn more about our processes while also getting a tutorial on the integration of our services. We attempted to improve their processes using our experience in addition to improving our support documentation, and improve the service to ease client implementation.

3 Discussion

Looking back over the journey so far, we notice some patterns in the kinds of changes made at different times. Throughout 2008, most changes were improvements to our programming techniques and general team organisation. During 2009 we focussed mainly on improving the flow of value through the organisation, which also involved giving a lot of attention to our approach to quality. In the most recent two quarters the development team's focus has again returned to programming technique, perhaps indicating that the team and business have found a "sweet spot" in their ability to collaborate responsively and effectively.

Our look back over the last two years also reveals patterns in Codeweavers' overall approach to problem-solving:

The team coach often used theatrical means to increase the apparent urgency of the team's problems, effectively "stopping the line" in order to get everyone's attention and perform whole-team root-cause analysis. Gradually the team learned to "swarm" when a crisis occurred, self-organising and stopping development in order to focus for a brief period on a single problem. During these stoppages, some pairs would work on fixing immediate symptoms, while others would analyse and address deeper causes. The team developed a consensus approach to quality that also transferred to development of MMFs: deployment of a feature or fix is now only permitted when everyone is happy with every aspect of its quality. Consequently, defect rates are at a record low, and continue to fall. You can see our current defect count on our support site[5].

Sometimes the team would agree on a process change, but would subsequently find that the change failed to "stick". It turns out that the most successful changes were always those that involved a physical or tactile component. For example, the team found it difficult to remember to focus on pulling value through the whole value stream; then someone suggested that everyone stand near the downstream end of the kanban board during stand-ups, so that all work in process was viewed from the customer's point of view!

Often, discovery of a problem led to a period of data collection, so that the team could decide how and what to tackle based on actual evidence. Sometimes the data collection itself served to create a culture in which the observed problem naturally disappeared.

Finally, as the focus on flow and throughput became more deeply ingrained, the developers' view of the value stream gradually increased. First we looked only at the "in development" column; later we lifted our gaze to look downstream to ensure our code was accepted and deployed to customers. Later still we looked further downstream, helping customers to adopt the new services, and upstream, helping the business decide what was needed and how to prioritise it. Recently the focus is expanding even further, into the company's sales and marketing functions.

3.1 Future Directions

The Codeweavers development team has a number of improvement goals for the coming year, including: solving the problem of automated end-to-end integration tests; expanding the philosophy to cover sales, marketing, administration, customer care and suppliers; reducing the amount of legacy code we have; and reducing defect levels to zero. The team also believes it can probably still halve overall feature cycle times by eliminating more queuing, waiting and waste.

Codeweavers plans to double the size of the development team during 2010, and is confident that the current kanban-based process will scale without problems.

4 Conclusions

Codeweavers has adopted a very simple inspect-adapt cycle of continuous gradual improvement, which has taken us from chaos to kanban, and from loss-making to consistent month-on-month profitability. We are getting better faster by stopping

production and swarming on problems, and it is important to note that every change has been made in response to a genuine business need. We are having unprecedented levels of success with a development process based on lean principles, no estimating, promiscuous pairing and the least knowledgeable developer rule. We are also confident that our lean principles will scale well and support us as we expand their scope to encompass the whole company (and beyond).

References

1. Fox, K.: TOC Stories #2: Blue Light creating capacity for nothing, http://theoryofconstraints.blogspot.com/2007/06/toc-stories-2-blue-light-creating.html
2. Wirfs-Brock, R.: Responsibility-Driven Design, http://www.wirfs-brock.com/Design.html
3. Cunningham, W., Beck, K.: A Laboratory For Teaching Object-Oriented Thinking. In: Proceedings of OOPSLA 1989 (October 1989)
4. http://limitedwipsociety.org
5. http://www.codeweavers.net/support/index.php