Ssh! We are adding a process…

Mark Striebeck, Google Inc.
mark.striebeck@gmail.com

Abstract

*Google is very successful in maintaining its startup culture which is very open and engineering-*centric. *Project teams don't have a project manager, but organize themselves and communicate directly with all stakeholders. Most feature decisions are made by the engineering teams themselves. As well as this works for products like search, gmail … it creates issues for the AdWords frontend (AWFE) application. AWFE is much more product management and release date driven then other Google applications. This presentation discusses how we carefully introduced agile practices to coordinate the AWFE development teams and made the process more efficient and predictable.*

1. Introduction

Google is known for its startup culture and its efforts to maintain it. In terms of the Agile Manifesto Google is almost entirely on the "left hand side" (with the exception of "Working Software"). Traditionally, project teams do not have a project manager, but organize themselves and communicate directly with all stakeholders. Even now, where Google has more then 6000 employees in numerous offices around the world, Google is still very engineering driven. Many new product ideas come from the 20% projects[*] of its employees.

The overall mindset at Google is to have as little as possible standard processes as possible. The reason is that the individual engineering teams will know best what is right for them. Upper management on the other side has trust in its engineers that they would not abuse this autonomy but do what is best for their project and the company.

AdWords is different. Being a B2B application, means that it needs much more business input/direction then other consumer-oriented products. Also, updating AdWords is a much bigger effort then updating any consumer product: all features have to be translated in many languages, sales material has to be updated, support has to be trained, and external communication about major features has to be prepared (forums, blogs and mails).

Therefore AdWords had a few standards:

- From the initial product idea, the product manager together with a UI designer and usability specialists creates almost final UI mockups. These mockups are used for a final project review by senior management and then given to engineering for implementation.
- During the whole project lifecycle, the product manager holds weekly meetings with all stakeholders (engineering, QA, UI, support, marketing). These core team meetings are the main communication channel. All major product decisions are made or at least discussed here. Lots of change requests come from these core team meetings during the project lifetime.
- Although, the core team sets initial release dates (with input from engineering), the final release date is determined by engineering progress and quality of the code. Given the scale of AdWords (number of users, business relevance, load, infrastructure); a small bug can have very severe consequences. Therefore features are rather delayed then released with insufficient or even unknown quality.

This level of process worked well in the beginnings of AdWords. But the AdWords product development outgrew this ultra lightweight process

- The application code consists of more then 500KLOC
- The engineering team is distributed in 5 offices worldwide – there are constantly 15-20 major projects ongoing plus maintenance and small improvements.

And the AdWords application and development team is still growing…

The unpredictability of launch dates caused more and more concern. Nobody wanted to lower the high

[*] Every Google employee is encouraged to spend 20% of his/her time on a personal project. This project should not be too closely related to the employees' actual work.

quality standards. But the initial release dates needed to be more reliable and delays should at least be known and communicated much earlier.

Because of its size and complexity AdWords has fairly large management team (for Google standards). In order to be effective the management team needed much more visibility into the projects and their status.

Finally, the rate of change is very high in AdWords. Teams who work on a project for a few months might find that they have a lot of cleanup to do before they can finally launch. Not so much because of code integration issues (the AdWords team runs a fairly comprehensive continuous integration suite) but because of feature changes. Often, projects that run for a long time have to play catch-up with all the feature changes before they release. In a few cases this lead to significant delays.

2. First agile attempts

Trying to introduce a process in a start-up environment such as Google often meets resistance. Because of the googley way of developing software, many engineers simply do not believe that any formal process can have a benefit but will only slow them down.

When I took on my first projects at Google I was just a few months with the company. The engineers did not know me at all. But it was interesting to see how the Google culture helped me here: A big part of Google culture is trust. This goes through the whole organization. And although I was new to Google and AdWords, the engineers and PMs trusted me that I would do the right things. Or better: they trusted the people who hired me that I am someone who would do a good job.

So, my strategy was to get as little involved as possible in the actual coding part and to start with a few practices that would just help us to track progress and show issues. Then we would introduce individual agile practices to fix such issues during development. I decided to start with the following practices:

- A release backlog and burndown charts [1]. These two tools provide high visibility into the development progress for the project team, but also outsiders. Using simple wiki pages to store the backlogs allowed the engineers to update their progress in very little time. I decided to measure the burndown rate by tasks complete, not feature complete. Measuring progress in feature complete has many advantages but also forces a team to change their development process a lot. It was one of the areas where I decided to rather introduce this practice later in order not to overwhelm the team.
- In past projects I made very good experience with estimating features/tasks in points [2]. Especially in an environment like AdWords, where engineers are often interrupted by meetings or tech talks, real time estimates are a problem. If the burndown graph tells us that we are implementing 3 days of work per week then it often leads to discussions what the team is doing the other 2 days. Or people try to match their updates to real days. Points are a good abstraction layer that avoids any such discussion.
- Scope changes are included in a controlled way by first estimating them, adding them to the backlog. Here, the burndown charts helped tremendously to get a quick assessment of the impact.
- A weekly development checkpoint meeting to plan the next week and work on scope changes. These checkpoint meetings were attended by the engineers, QA, PM and UI. At this point I did not introduce real iterations. My personal experience was that changing to iteration-based development is a significant change for developers and QA. It sounded too heavy to introduce at this point.

For the adoption of these practices, I tried very hard not to implement anything top-down but to get buy-in from engineers and the product managers. The initial changes sounded reasonable to the engineers. Because I was managing several projects, I could not be too closely involved in the development activities itself. This probably worked to my advantage – the engineers realized quickly that I would not try to tell them how to do their job, but that I only structure the project in a certain way which was not too intrusive. Also, one of the goals was to keep the self-organizing character of teams intact. After all, this is a big part of Google culture and our agile adoption approach would have failed if we had severely impacted it – no matter how successful the projects would have been.

This approach also helped me to work with several projects at the same time. Many meetings regarding UI, features, design… took place without me. Only when we discussed scope, scheduling or planned the next steps, I was there and was usually leading the meeting.

2.1. The guinea pig projects

Changes at Google are often done in some kind of guerilla approach: one project team adopts something new. If it works, other project teams get interested and will try it as well. Therefore, we started only with two projects:

Project A: This was a very new piece of functionality which did not overlap with existing features. The UI was fairly complex; the engineering team consisted of new recent college graduates working in a remote office.

Project B: This project was a simplified version of AdWords. It was heavily integrated into existing features (we basically had to think about every other feature and had to integrate or disable it). The team consisted of experienced engineers. Some of which had already work for some time at Google, others were new to Google).

2.2. The first process steps

In both projects, we used the UI mockups to generate the release backlog by dissecting the screens into individual features. This pre-development process is very well established at Google and it seemed too complicated to make this part more agile.

The release backlogs were stored in wiki pages which made it very easy for engineers to update them. From these wiki pages we automatically generated burndown graphs to visualize the project progress. The concept of giving status updates in work left and not in work completed was initially strange to both teams. But the engineers quickly realized the advantage.

As stated earlier I did not introduce iterations at this time. Instead I installed weekly checkpoints with the development team (PM, UI, Engineering and QA). In these checkpoint meetings, we discussed progress, additional feature requests and other issues. Additional features were estimated and added to the release backlog. I extended the burndown graphs and used a variable floor to indicate the scope changes. The graphs gave us quick feedback what the estimated impact of these additional features was.

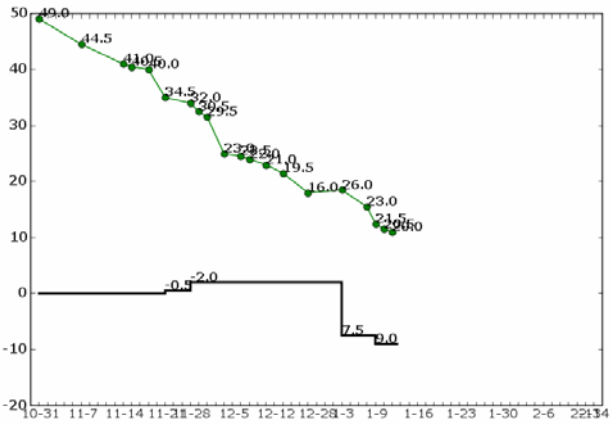

Table 1: Burndown graph with variable floor

Although I did not try to implement an immediate testing of implemented features, I wanted to get away from the purely phased approach where testing starts after development is finished. To push for this, we setup staging servers that were rebuilt on a nightly base with the latest code. These staging servers were used for testing the application but also for UI walkthroughs[*]. Usually, they are performed towards the end of a project when the system is nearly complete. But because we staged the application early on and implemented end-user features (from the UI mockups) we could start with these UI walkthroughs much earlier and gather important feedback for the further development.

2.3. Issues to overcome

In both projects we faced similar issues:

Customer / Product Owner concept

Most agile processes have this role which is responsible for features, prioritization and ultimately scope vs. release date decisions. It is usually an individual or team outside of the development team. But at Google, many of these responsibilities rest with the team leads. The product managers usually have more then 10 projects at the same time. This does not give them the bandwidth that the product owner role requires. Also, they trust the tech leads and UI designers enough that they will make good decisions (often, when I asked a product manager for prioritization of a feature, he turned to his tech lead and simply asked “what do you want to do?”).

This gives the planning and prioritization meetings a different dynamic. Often, the tech leads do not see the need to make such decisions during the planning meetings as they know that they will be involved enough during development itself that they can make such decisions at a later point. I usually drove the team to make at least those decisions which are necessary to create good effort estimates and priorities for the backlog. I always wanted to leave the weekly checkpoint meetings with good updates to the release backlog.

Retrospectives

For me, frequent retrospectives [3] became such an important part of software development that I tried to install them in the weekly checkpoints from the

[*] UI walkthroughs are live demonstrations of the system with the whole core team to gather feedback and uncover usability issues early enough.

beginning. It would have helped a lot with improving our process through constant feedback.

But both teams were not (yet) used to having a formal development process. The weekly retrospectives usually turned into a status report from the last week but very little about the process itself. This was aggravated by the engineering centric culture at Google. When an issue comes up, most engineers at Google only consider technology to fix it.

After a few weeks, I silently dropped retrospectives from the weekly checkpoints. I decided to wait until the teams embraced the concept of a development process and that they own it and could change it to fix problems.

Constant scope increase

In both projects, the scope increased significantly (more then 30%) during development. Interestingly, these scope changes were not the result of additional feature requests by the product managers. Most additional tasks were the results of oversights during the release planning:

- The engineering team missed features in the UI mockups when we created the release backlog
- Integrations into other AdWords features were overlooked. Also, the rate of change in AdWords is very high. During development others areas of the application changed and we had to change our integration as well.

Most of these additional tasks could not be down prioritized for a later release but had to be added to the release.

In both projects, this lead to several postponements of the release date as no other feature could be dropped from the first release.

Although, there was considerable frustration about these delays, both project teams and management appreciated that we at least knew about these postponements early enough and not just the week before release. The burndown graphs gave a good visualization and the release backlogs made it easy for everyone to understand what was left to be implemented.

> *The backlog was handy as things came up over time and as we dived deeper. One function was to not loose the line items but more important it was useful for the team to see how many un anticipated issues cropped up and have a good snap shot in time.*
>
> *Product Manager*

2.4. Working with the remote team

As stated earlier, project A was implemented in a remote location. The rest of the core team was in our headquarters. Initially, I was concerned how that team would react to my leadership – if they would appreciate it as much as the other team or if the would regard it as a heavy-handed approach from headquarters.

To my surprise I did not encounter many issues with this project. Only providing tools to get more visibility into development progress and facilitating planning meetings seemed to be the right level to give the remote team enough room to work mostly autonomously. Also, I could make myself very useful in facilitating lots of communication with other engineers in our headquarters. The team realized quickly that I indeed tried to help the project progress and not to control them remotely.

3. Adding agility – one practice at a time

3.1. Daily standup meetings

Both project teams initially rejected the idea of daily standup meetings [4]. They were seen as an unnecessary overhead.

But during development we discovered issues in the weekly checkpoints from the past weeks:

- QA tested unfinished features or was not sure how to test new features
- Engineers who worked on related features worked on the same refactors. The AdWords engineering team has a very healthy culture of constantly refactoring the code. The downside is that two engineers who work on related features often start to improve the same code.
- Engineers could not continue with their implementation because they depended on a task from another engineer. Often enough, the other engineer was not aware of this dependency.

It was clear to everybody that these issues could have been avoided had the team communicated earlier. At this point it became easy to convince both teams to try out daily standup meetings and to include QA in these meetings.

The first standup meetings were quite lengthy. Everybody had a lot to talk about and had problems to focus just on a quick status update (“done”, “to-do”, and “issues”). But after a few days nobody had a big baggage anymore and everybody realized that there is not much to talk if you restrict yourself to the past 12 hours and next 12 hours. Several issues were resolved or at least uncovered during these meetings. After a couple of weeks, both projects did not need a reminder

anymore but made the standup meeting part of their daily routine.

3.2. Small steps – Completely finishing a feature/task

In project A, the progress looked very good. Initially, we estimated 3 weeks for a set of screens. When we did low-level estimates, we came to 40 points. After the first week, the team did 8 points – in the second week 7.5 points. I looked as if the initial estimate was too low and the team would need 5 instead of 3 weeks.

Interestingly, the tech lead of the team was convinced that the screens could still be implemented in 3 weeks (i.e. all remaining 24.5 points in 1 week!) quote: “It just does not feel that much anymore”.

After week 3, the team was not done. The team implemented another 9 points. The velocity looked very stable: ~8 points per week.

To my big surprise, the tech lead announced in the core team meeting once again that his team will be done in one week…

> *It took me some time to learn to trust the burndown graph and to question my gut feeling when a feature would be finished.*
>
> *Tech Lead*

The fourth and fifth week showed a significant drop in velocity: 4 points and 2.5 points! It turned out that the team did not completely finish the tasks: tests were not written, code was not reviewed (which is mandatory at Google), features were not completely integrated. This caused the burndown graph to go down because we did not measure progress in finished features, but in tasks.

This caused a further delay and the screens were finally implemented after 7 weeks. This additional delay caused some concern with the core team. To avoid this situation I added a green/yellow/red color coding to the burndown charts to indicate how many tasks are new/started/finished. This made it very clear if velocity was high because many features are partially finished or if the team completely finished a feature before moving to the next one.

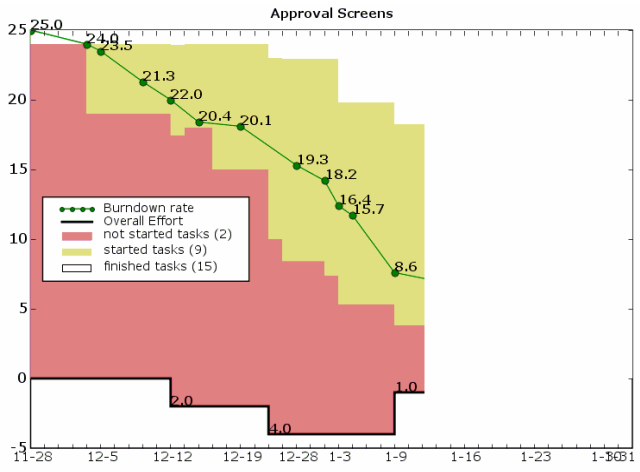

Figure 2: Indicating started and finished tasks

The team responded very positively. It was quite a shock for the engineers to see that up to 80% of all tasks were in a ‘started’ state. They started to keep the corridor of started tasks as small as possible.

Overall, this was a very healthy learning experience for the team. It showed them the difficulty that we tend to have when trying to estimate a release date instead of deriving the release date from effort estimates and progress. It also showed them that we can only measure progress well, if we completely finish tasks and not leave small bits and pieces around which sometimes turn out to be much larger then we thought.

3.3. Spikes

In the weekly checkpoint meetings we often discovered that tasks took much longer then initially estimated. Or the team had problems with estimating a new feature.

Initially, the engineers just wanted to pad estimates for such unknown tasks. Often enough, these padded estimates were much too high or still too low. And everybody could see that they lowered the usability of our burndown graphs significantly. So, we added in a spike (an investigative task) to help determine what the effort for the implementation would be. Especially when the scope continued to grow, everybody realized the value of getting a better estimate of implementing a feature before actually starting to work on it.

4. Release experience

The two projects had somewhat different releases:

Project A)

The team had fixed many bugs already during development, only few bugs were discovered in the final integration test phase. It was a very smooth launch.

Project B)

Because of the integration into all other AdWords features, QA found many issues during development – most of them through exploratory testing [5] (i.e. not really tied to a particular product feature). The team tried to keep the bug backlog under control but did not want to fix all bugs. When we came close to launch, we had to review the bug backlog several times and down prioritize many bugs. Until a few days before launch it was not clear if we could fix enough bugs to release it.

At least the team did not encounter any issues that required a complete redesign of some area – which could have easily happened for such a far reaching feature.

Still, the overall release experience was very positive. Both projects were very successful in production and had very few issues.

5. Feedback and next steps

I held post-mortem meetings with both projects. In these meetings I focused the teams on listing positives and negatives and not jumping to discuss solutions immediately. From the overall list, the teams selected the worst issues and best practices to keep:

Positive

- Project Management and tools (burndown charts and backlogs)
- Early QA and availability of a staging server
- Teamwork and collaboration

Negative

- Unclear or non existent prioritization
- Felt as if team missed release date several times
- Too risky at end because of bug backlog (Project B)

It was very encouraging that both teams found the overhead of maintaining and updating the release backlogs worth doing.

> *Burndown charts made it easy to see when were making progress, and gave us a nice sense of satisfaction and completeness.*
>
> *Engineer*

And, furthermore that the process did not impact the great teamwork and collaboration that Google teams have. Also, the effort of maintaining a dedicated staging server was appreciated. The engineers from both teams were very positive about the early testing and feedback by QA that the staging server afforded.

> *I think it took some time getting used to the approach of testing so early in development, and also making sure that QA and dev were on the same page. I think that our daily standups and also having QA co-located with dev has helped greatly here.*
>
> *Engineer*

6. The second version

From the feedback of the post-mortem meeting I tried to modify the development process further to address the worst issues.

In both teams I gave at this point a presentation about a full Scrum process [6]. During the first projects there were many tech talks at Google about agile development (by internal and external speakers). Both teams got very interested in it. They could see that their practices fit into agile development but heard a lot about other practices too. Also, the very positive feedback of my project management style and tools showed me that the engineers trusted me and my guidance. In both teams we discussed which additional practices to adopt:

Product/Release Backlog

To address the prioritization issue, I worked with the product managers of both projects to organize their requirements in prioritized lists. It took a little bit of time for them to get used to it, but was not a major effort. The core team members liked the backlogs a lot. It gave them much more visibility and input into development. Initially, there was still the desire to make each feature high priority. But soon everybody realized that even if a feature is not included in the current iteration, it will still get done fairly soon.

Iteration based development

This was the hardest practice to introduce. Without practical experience it is hard to explain why iterations are better than scheduling the whole release at once and adding to it when necessary.

But with the feedback about missing deadlines and too many bugs, I could explain how an iteration based approach would address these. The concept of not only implementing but also testing and completely fixing features within the same iteration sounded very appealing to the engineers. Although, they were

somewhat skeptical of this high-quality approach, both teams wanted to give it a try.

The teams soon realized the advantages. The planning meetings became much more focused than the weekly checkpoint meetings from the previous projects. No time was wasted with discussing the same feature for 5 weeks but never implementing it. Or to discuss and design a feature that finally gets dropped.

We agreed to start with 2 week iterations. This synchronizes well with the 2 week release cycle of AdWords. We are finishing the iterations with the code freeze for the next push. This means that a high-priority feature that gets put on the product backlog can be implemented and release within 4 weeks without any interruption.

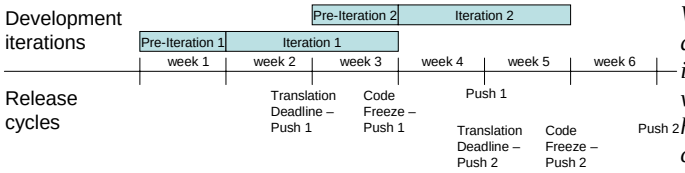

Figure 3: Synchronized development iterations and release cycles

Retrospectives

After the previous projects, both teams had some experience with a defined development process and that they can influence/change it. I started the iteration planning meetings again with a retrospective and this time it was much more fruitful. Most contributions were about how we develop our application and how we can improve that.

Review of iteration features with core team

In the first projects, we reviewed the application by clicking through it during the core team meeting and collected some feedback. Now, with the iteration based development we do these reviews at the end of each iteration and only on newly implemented features. This made the reviews more focused and gives us feedback early enough so that we can integrate it in the next iteration.

Testing tasks for features in same iteration

In order to test features in the same iteration as they are developed in, we added testing tasks to the iteration backlog. The QA engineers were asked to provide effort estimates for these tasks so that they can be included in the burndown chart.

Overall, the teams could see how these process changes would address the negative feedback from the post-mortem meetings. Both teams did not fully understand how these practices would work together but agreed to give it a try.

At this point I took on a third project where I implemented the new process from the beginning. The product manager of this team was from Project A, the QA engineer from Project B. This made the adoption much easier. Also, many people in AdWords had heard about how I ran my projects and the barrier to try it out was considerably lower.

6.1. The world is better, but …

Overall, the more agile processes worked really well. Everybody noticed that the additional structure comes with very little overhead and fixes many of the issues that we had before.

> *We're still getting up to speed on the iteration-based development. It's been nice for development, now that our iterations are in sync w. AdWords code freeze cycle. It was hard at first for UI/PM, but has gotten easier as PM has assembled farther projecting roadmap, to give UI a clue what will be needed for a coming iteration.*
>
> *Tech Lead*

After a month or two, both product managers realized that they need to establish a requirement process that ensures that we not only implement little bits and pieces at a time but keep the overall release. This is an issue that I had with previous agile teams. I could persuade the product managers to dissect their releases into smaller chunks and prioritize them.

For these I created release burndown charts to track when they will be finished. At this point I started to measure progress on the release level in features complete. At this point it was very easy to convince the teams that this is the right measurement as it would give us a much better guidance where the release is.

The teams first thought that it was strange to have one iteration burndown chart and one release burndown chart. But after a few iterations they saw the benefit of both. The iteration burndown to guide actual development efforts. And the release burndown to guide the overall release planning.

An ongoing issue is the QA involvement. I constantly have to push the QA engineers to test features immediately after they are implemented. The reason is that the QA engineers support several projects. And the other projects are not agile, i.e. don't require much attention during development, but a lot at the end. This made it hard for the QA engineers to constantly spend a little bit of time each day on or project to give the engineers the immediate feedback. Right now, both teams question if it is worth the effort to include QA tasks and effort estimates in our planning as it does not seem to have any benefit.

For me, it seems like an extra task of updating a table with data (QA estimates) that's not of significance for me. But I'd really like to know if it's helpful to others. So far, most of the estimates have been 0.1 points.

QA Engineer

Finally, the teams do not try to create a releasable product at the end of the iteration (which is even harder because of the QA issue mentioned above). There are always tasks half implemented, not tested, need review… For now, I am not pushing too hard on this. The teams completely implement enough features per iteration that we can release those with the next AdWords update.

6.2. The project manager is dispensable

Recently, I went on a 3 week vacation. I was concerned how the teams would continue with the agile process during my absence and reminders and reinforcements of our agile practices.

But it turns out that the teams embraced the process enough to continue it even without any reinforcement. Iteration planning meetings happened, backlogs were created according to previous velocity, and daily standup meetings took place …

7. Where are we going from here

With the success of three project teams we are now prepared to make much bolder steps. Everybody in AdWords had at least heard about the advantages of the agile approach. Resistance at this point will be much less.

- Establish backlogs and burndown charts as status reporting standards in AdWords. Even if teams do not adopt other agile practices, these practices are easy to implement and provide a very good visibility for the teams themselves but also management and other outsiders.
- Other managers voiced interest. With a shadowing approach I will guide them through the agile process and try to give them enough experience to implement agile practices in their projects by themselves
- A few projects involve teams from several AdWords departments (frontend, backend, NetAPI…). Such projects always required much more management attention. As great as Google engineers and tech leads are, coordinating and synchronizing a teams efforts with other teams often distracts tech leads too much. We will either try to coordinate these teams as one big team (one backlog, one burndown chart) or use the "Scrum-of-Scrum" [7] approach.
- During the first months at Google I heard from other departments who are using some agile practices or full-on Scrum/XP processes. To support this effort we started a grouplet[*] that focuses on agile development. We just recently started this grouplet and the initial response / interest was overwhelming – not only from engineering, but also other groups (QA, Product Management, UI, Usability)
- The UI development and usability part of our development projects is still very frontloaded. Almost all of this work is done before development starts. A few usability experts and UI designers showed interest in making this also part of the iteration-based development.

8. Summary

With the help of an experienced agile leader (scrum master, XP coach…) it was possible to carefully introduce agile practices into Google - an environment that does not have an affinity to processes in general. Instead of introducing a grand new process, individual practices could be introduced either to fix observed issues or just to "try them out" – the development teams realized the advantages very soon.

Along with these practices came a visibility into the development status that gave the approach great management support.

All this could be done without destroying the great bottom-up culture that Google prides itself of. The practices only affect how the projects are structured. Design and implementation remains fully an engineering responsibility. With some modifications, we could even keep the very strong role of tech leads and UI designers.

In keeping the great culture and self-organization of the teams, I could easily manage several projects in parallel. I could continue to rely on all core team members to communicate effectively without introducing any heavy processes.

[1] Controlchaos website - http://www.controlchaos.com/about/burndown.php

[*] Google grouplets are cross-department groups which focus on a specific area of the software development process (there is a tech documentation grouplet, a build tools grouplet…) The members of the grouplet use their 20% time for their participation.

[2] Mike Cohn, “Agile Estimating and Planning”, pp. 35-42.

[3] http://www.retrospectives.com/pages/whatIsARetrospective.html

[4] XP.org website - http://www.extremeprogramming.org/rules/standupmeeting.html

[5] http://www.satisfice.com/articles/what_is_et.htm

[6] Controlchaos website - http://www.controlchaos.com

[7] Mountain goat website - http://www.mountaingoatsoftware.com/scrum/scrumteam.php