Software engineering process transformation in multi-component environment

Final Capstone Project

Project Supervisor: S. Potapov

Viktor Shchepanovskyi | MDT21 | March 27, 2021

Kyiv School of Economics

ABSTRACT

Faculty of Business Education

Master's in Digital Transformation and Business Analytics

Software engineering process transformation in a multi-component environment

by Viktor Shchepanovskyi

In this document, you will follow me in my process optimization journey from the start in April 2020 till the current moment, where many milestones are completed, but new incoming challenges are coming.

We'll team-level team level improvements in process optimization, but as a general part of digital transformation because digitalization itself is not only software solutions itself with improved information flow. It is also a new organizational and managerial toolset to work in the new informational era with its most beneficial results.

Then the company would expand it for a multi-component setup. We'll cover the team synchronization topic with the best performance results and a good fit into initial planning.

And we will finish with the Quality Assurance process on the team and program level as the final step to complete the product with a guaranteed level of quality, and we'll reach it with the best time spent on it.

TABLE OF CONTENTS

INTRODUCTION

Business case level – Function(s). For example, scoping and planning of IT solution, when automation or enhancement of particular business function(s) is required/proposed. Could be part of IT Strategy for a selected business function, implementation of new IT solution from scratch, enhancement of existing IT solution, enhancement of service journey within the chosen business area, development of the long list of digital initiatives with clear prioritization criteria (e.g., ease of implementation/cost of implementation and expected productivity impact) with fact-based recommendation on top priority initiative(s) or solution(s); communication tactics on the functional level.

At the beginning of 2020, my friend and ex-colleague Alex phone me to create a new Mobile Department for his client's company. In previous years Alex works with them as head of the Data Analysis department on an outsourcing basis.

Uproad company has at that time Mobile development located in Polish outsourcing company Nomtek, based in Wroclaw. But company planned to create its mobile department as a general next step of company growth.

It's cheaper and faster to start software development based on the existing team (outsourcing or any other way of collaboration) than to try to create groups and products from scratch simultaneously. Later (after MVP release) stages, it's cheaper to move into own development team creation to avoid paying fees overs direct expenses on salaries to the outsourcing company.

Even with that, the department's start as a Department in Alex company, with somewhat same outsourcing contract, agrees about the whole department's possible buy-in.

Starting with that phone call and his promise about project sustainability, I've started hiring engineers for our new team. I've contacted many ex-colleagues who seek a job at that time and published openings on Djinni and DOU.

And in one month after we have principal agreements with the whole set of future Mobile department employees. Official contracts were signed at the start of April with the start date of April 21st, 2020.

During the first month, we identified areas for improvement, gathered information, and created a basis for future changes. I've formed a separate coalition for each of the directions and drive with them through.

The general direction of all changes was to increase performance, future planning, and release and reach healthy team spirit.

The following chapters will disclosure what happens next, including new team processes configuration, incorporation into existing company's owned/outsourced setup, gaps found, and achievements reached.

It's important to say that each new step was opened by completion of previous; this removes bottleneck on the earlier stage and helps identify it later.

CHAPTER 1. BUSINESS CONTEXT AND DIAGNOSTICS

Let's introduce the business context of the tolling domain, Uproad company, and shared/modern tools and methodologies in that area.

Electronic toll collection (**ETC**) is a wireless system to automatically collect the usage fee or toll charged to vehicles using toll roads, HOV lanes, toll bridges, and toll tunnels. It is a faster alternative replacing toll booths, where cars must stop, and the driver manually pays the toll with cash or a card.

In most systems, vehicles using the system are equipped with an automated radio transponder device. When the car passes a roadside toll reader device, a radio signal from the reader triggers the transponder, which transmits back an identifying number that registers the vehicle's use of the road. An electronic payment system charges the user the toll.

A significant advantage is a driver does not have to stop, reducing traffic delays. Electronic tolling is cheaper than a staffed toll booth, reducing government or private road owners' transaction costs. The ease of varying the tolls makes it easy to implement road congestion pricing, including high-occupancy lanes, toll lanes that bypass congestion, and city-wide congestion charges.

The payment system usually requires users to sign up in advance and load money into a declining-balance account, which is debited each time they pass a toll point.

Electronic toll lanes may operate alongside conventional toll booths to pay drivers who do not have transponders at the booth. Open road tolling is an increasingly popular alternative that eliminates toll booths; electronic readers mounted beside or over the road read the transponders as vehicles pass at highway speeds, eliminating traffic bottlenecks created by cars slowing down to go through a toll booth lane.

Vehicles without transponders are either excluded or pay by plate – a license plate reader takes a picture of the license plate to identify the car, and a bill may be mailed to the address where the car's license plate number is registered, or drivers may have a certain amount of time to pay online or by phone.

Electronic Toll Collection Market size was valued at $6,855.5 million in 2017 and is projected to reach $15,648.2 million by 2025, growing at a CAGR of 10.7% from 2018 to 2025.

Structure of market:

Transponders	77%
Cash	16%
Video	5%
Mobile	2%

The mobile sector, where Uproad operates, forecasted 20% in 2025. Also, charges via Video could be paid using Uproad.

Company (business) introduction

Uproad is a start-up company having the mission to deliver next-generation Mobile applications in the tolling domain for the US consumer market (B2C).

Based on GPS location tracking, the app can provide a cutting-edge user experience to pay for tolls, receive toll alerts, calculate toll pricing for the given route, pay your bills, and much more. Since launching in stores back in March 2020, more than 20k users have installed the mobile app. The company aims to be a Nationwide provider in the US for tolling service for millions of users.

The mobile app is powered by AWS cloud-native backend built from scratch following modern microservice architecture and DevOps best practices.
Uproad is backed by the Kapsch Group, one of the tolling industry leaders in the US and European markets for decades.

The team is about 70 professionals distributed across three countries. Most of the engineering staff is located in Ukraine (Kyiv), making it easy to collaborate, learn from experienced teammates, and have a fantastic team atmosphere and colleagues' support.

As an organization, Uproad consists of the following Divisions:

1. Product:
    a. Product managers
    b. Designers
2. Delivery:
    a. Backend
    b. DevOps
    c. CRM
    d. Mobile
    e. Growth, Data Science
    f. Integration & Performance QA
3. Marketing

4. Customer Care

All of the mentioned teams have their C-level representatives and Program/Direction managers. Some of the departments are fully or partially outsourced. But the general movement to own employees in all areas is in place. Even during a covered period of one year, many activities migrated from outsourcing to their service.

Key challenges and inefficiencies identified

Team level:

All teams work in their operation cycle, with only one synchronization on general release train movements within two weeks sprints and two sprints per release.

In the Back end, the team measured by Ideal hours by counting how many ideal working hours were spent in the office. At the end of the sprint calculation of the ideal to actual hours' coefficient, the team has calculated the following sprint capacity. They have grooming performed by the Product manager; the group's technical lead provides all high-level estimates. Estimates for sprint planning filled in during planning sessions with team Scrum master.

CRM team counts only business working hours, no coefficients to convert them to something. Following Sprint, capacity is equal to 2 weeks of business time, multiplied by the number of engineers who work in the department. The Product Manager performs grooming, and he also provides high-level estimates. Estimates for tasks are corrected with the team on planning.

DevOps, Growth, and Marketing team have no planning or sprints and work in continuous Kanban.

The last Mobile team worked on deviated scrum basis, measuring in business hours but planning with Fibonacci margin numbers. I guess to keep at least something from Agile/Scrum methodology. They have grooming performed by the Product manager; the team's technical lead provides all high-level estimates.

Such Fibonacci margins from the start give us a clear look that something went wrong there and some time could be extracted from development spent – just because engineers on measuring Stories between 13 and 21 hours, with real value 15 as an example – would prefer almost always 21. And then, based on Parkinson's law, will spend all these 21 hours on the task.

Scheduling related tasks between teams:

Work on Epics was continued in its Stories for teams. And connected or related tasks were implemented simultaneously by few teams without testing how it works together. Only preliminary interface agreements were approved before implementation started.

During development sprints, and even right after implementing Epic's final element, it did not check. Team testers inspect only component-based parts.

Testing how it works with each other starts only after two sprints on the Integration testing stage. All issues found goes to current teams' sprints with the highest priority and takes a lot of time for branch and context switch to fix it.

This late issue recognition increases the hidden development cycle to more than a month after the team's features were implemented.

Time spent on mocking data and removing mocks for component testing and closing teams' sprints wasted on behalf of this process.

Quality assurance:

As the outcome of the previous process, double testing of application or services areas was a common cause.
Teams were testing parts of features on component level, and after one month, the integration team tries almost the same features and scenarios but with components connected.

But an even bigger disaster was with regression testing. Classic bug-searching system of quality assurance increments a massive number of new tests for new features to test scope each Sprint. And all of them should be rechecked manually before each release.

The growing number of test cases means a growing linear amount of time spent on this testing. In a few releases (or few months) after MVP release, it became impossible to test integration and regression scope in one month time period. Simply said, the range of features developed during the month requires more than a month to be tested.

There were no usual replacements of features, like in established projects. The trend of increasing test scope for manual testing became a disaster. So only two ways were considered to resolve the problem in the short term: increase the number of testers and skip some tests for areas without changes.

There were some beginnings of test automation from a strategic perspective, but no strong focus neither on them nor on unit tests quality.

Business requirements and use cases

On the **team level** there were formulated following target requirements to the final development process:

1. Performance, the velocity of team and company
2. Forecasting predictability, reliability
3. Work perception by employees – healthy work conditions and work-life balance

Use cases to be covered:

1. Grooming and Pre-grooming (technical analysis) stages
2. Planning of Sprint
3. Inter-team planning
4. Employee reporting about things done
5. Team reporting about things done
6. High-level estimates
7. Demonstration of team results
8. Retrospective on team results
9. Application of measured complexity and uncertainty to following sprint scope
10. Defining buffer

Scheduling related tasks between teams, targets:

1. Time to release for feature
2. Time to release scope of features
3. Quality, number of issues on each stage
4. Avoiding double work and double testing

Use cases to be covered there:

1. Inter-team planning
2. Team planning
3. Relations and connections administration
4. Scopes integration

5. Defining relations and dependencies buffers

Quality assurance requirements targets:

1. Avoid double testing
2. Reduce time spent on regression
3. Utilize current team resources
4. Use current automation Framework whether it's possible
5. Quality should not be degraded

Use cases to be covered when we change quality assurance procedures:

1. Test specification
2. Test execution
3. Development of Unit tests
4. Coverage requirements
5. Development of automation tests
6. Schedule of work of tester during Sprint
7. Time framing for learning

Current state analysis

Despite issues and inefficiencies found, processes were working in general and delivering main production goals successfully.

When we try to change something, we always should consider that it's better to avoid the breakage of current achievements. New solutions must comply with the ongoing strengths of the project and processes.

Current (at that time) **Team level mobile department process strengths:**

1. Predictability: team deliver features at the estimated time because it reserves more than needed time for it
2. Healthy work-life balance: because there was no need to work overtime, even some free slots frequently appeared
3. Alignment with other components activities: based on predictability and business hours usage
4. Focused on areas separation by features directions: some people work only in dedicated areas, which produce faster execution there

Weaknesses:

1. Team velocity: It looks like team performance could be better, and some underutilization present
2. Fragility to team vacations and leave outs: connected with the last strength point, this is also a weakness because in a case when the team lost one teammate, subsequent work in some direction stopped or performed very slow.

My targets were improving performance and reducing fragility; without losing predictability, work-life balance conditions, and expertise on research-based topics.

Current (at that time) **Scheduling connected tasks between teams strengths:**

1. Feature documentation preparation is attached to only one calendar milestone – there were no separate dates for each component
2. Simple managerial planning: all related tasks were inside the same initial period of implementation, and this was easy to track
3. Relatively fast speed of delivering the first version of the feature to internal testing – because all connections were ignored on component level stage

Weaknesses:

1. Performance: a lot of time spent on mocking connections before implementation and cleaning up the project after implementation.
2. Late issue detection: in the worst case, some bugs found one month after the whole feature development was finished
3. Context switching time spending: starting from time required to re-setup locally version of an application which could be much different from the current one; to time spent on remembering what happens one month ago and what is the logic behind code written

Based on mentioned above, my targets were improving performance and time to market delivery speed without losing simplicity in release planning and featured scheduling from the managerial point of view.

Current (at that time) **Quality assurance strengths:**

1. Cheaper on early development stages: in salaries of manual and automation testers
2. Faster on early development stages: because no time needed to code automation tests, just run manual regression testing
3. Simple testing environment: easy to follow and perform introduction how to work with it for new employees
4. Guaranteed level of quality

Weaknesses:

1. Expensive on later development stages: in time and money spent on all related activities
2. Slower on later development stages: there is no way to accomplish tests in-suite except manually check all of them

3. Challenging to motivate testers to do their job with the same level of inspiration each month, if they manually recheck almost the same things

Based on that, my targets were to decrease cost and time spent on regression testing in the long run, remove ineffective repetitive work parts for testers respecting the required level of quality and without a significant increase of complexity of the system.

Available tools to measure the impact of changes

Team level changes metrics:

1. The number of tasks and their complexity has done over a period
2. Satisfaction survey results
3. Number of incomplete tasks at the end of the Sprint
4. Number of bugs found on regression stage

Tools to measure these metrics are:

1. Jira Mobile board: state at the end of the Sprint, reports
2. Google docs forms for surveys
3. Jira boards and statistics on regression stage

Scheduling related tasks between components metrics:

1. Time to release for feature
2. Time to release for features scope in the release version
3. Number of bugs found on regression stage
4. Team satisfaction survey, the part about collaboration with other teams

Tools to measure these metrics are:

1. Jira reports and statistics
2. Additional calculations based on Jira reports in Redash and Excel
3. Google docs forms for surveys

Quality assurance changes metrics:

1. Time spent on regression run
2. Time spent on tests run during Sprint
3. Expenses summary
4. Work satisfaction survey results

Tools to measure metrics:

1. Jira reports and statistics
2. Additional calculations based on Jira reports in Redash and Excel
3. Google docs forms for surveys
4. Financial statements for alternative options

CHAPTER 2. OPTIONS ANALYSIS, PLANNING

The modern managerial toolset contains many strategies and development systems to be applied in cases like ours. Despite using two weeks sprints, we were not limited to apply only time-based solutions. For example, Kanban is also suitable for a team methodology where the general release process is scaled agile with release train.

Considering the main project constraints and circumstances of uncertain research and development software engineering domain, the most efficient from my side was using one of the Agile methodologies on the team level. Such change was supported by the calculation of total capacity per team per release on the program level. This provides us a possibility to evaluate cost-benefit or impact-effort analysis on features planned.

Scheduling of tasks related to one feature between teams proposed to be aligned to "As late as possible", without mock or fake data usage. Using it, we avoid a lot of dead time investments into obsolete intermediate code and reduce context switching cost. Also, early bug detections were achieved this way.

Later on, after the implementation of the suggested changes, we faced issues with testing time growth. This bottleneck was decided to be resolved by movement from an anti-pattern ice-cone testing scheme with domination of manual testing to ideal (as state always to be directed in) pyramid with the strong fundament of unit tests in its basis.

This change was most difficult because it requires additional education and a new automation setup fighting against growing from each spring regression scope.

Let's dive into more details about why chosen options fit the best for a described problem resolution.

Proposed options and suggested solutions

Team level: Agile / Scrum

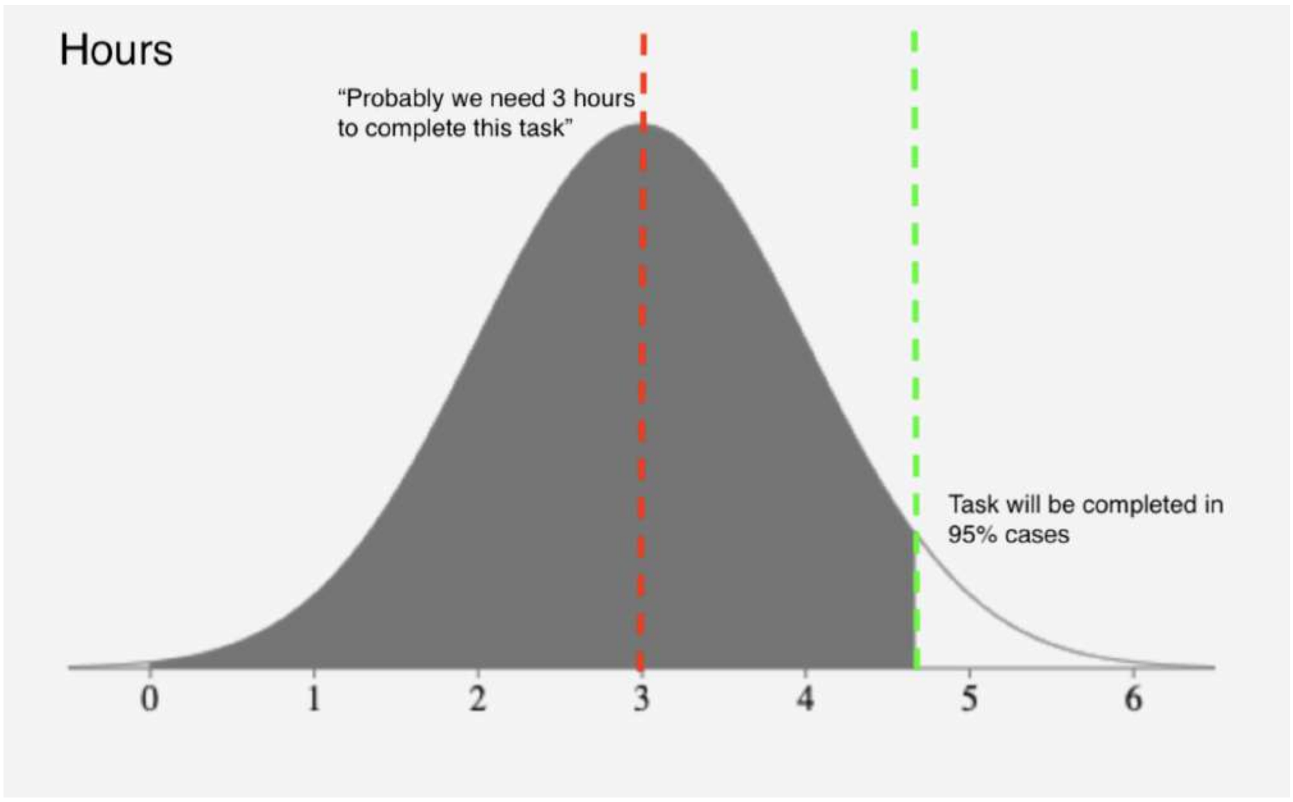
Let's make a short journey into how an uncertain task is forecasted and measured for future research work on it. For example, an assignment requires an average of 3 hours to be completed with a good quality level. And a little bit more than six with the execution of an engineer from this domain.

The probability chart of "work done at" showed on the following charts with normal distribution. This statement strongly relies on Central Limit Theorem.

In probability theory, the **central limit theorem** (**CLT**) establishes that, in many situations, when independent random variables are added, their properly normalized sum tends toward a normal distribution (informally a *bell curve*) even if the original variables themselves are not normally distributed. The theorem is a critical concept in probability theory because it implies that probabilistic and statistical methods that work for normal distributions can apply to many problems involving other types of distributions.

Let's proceed with the following charts and stages describing the time needed to have work done.

Stage 1: How engineers estimate effort

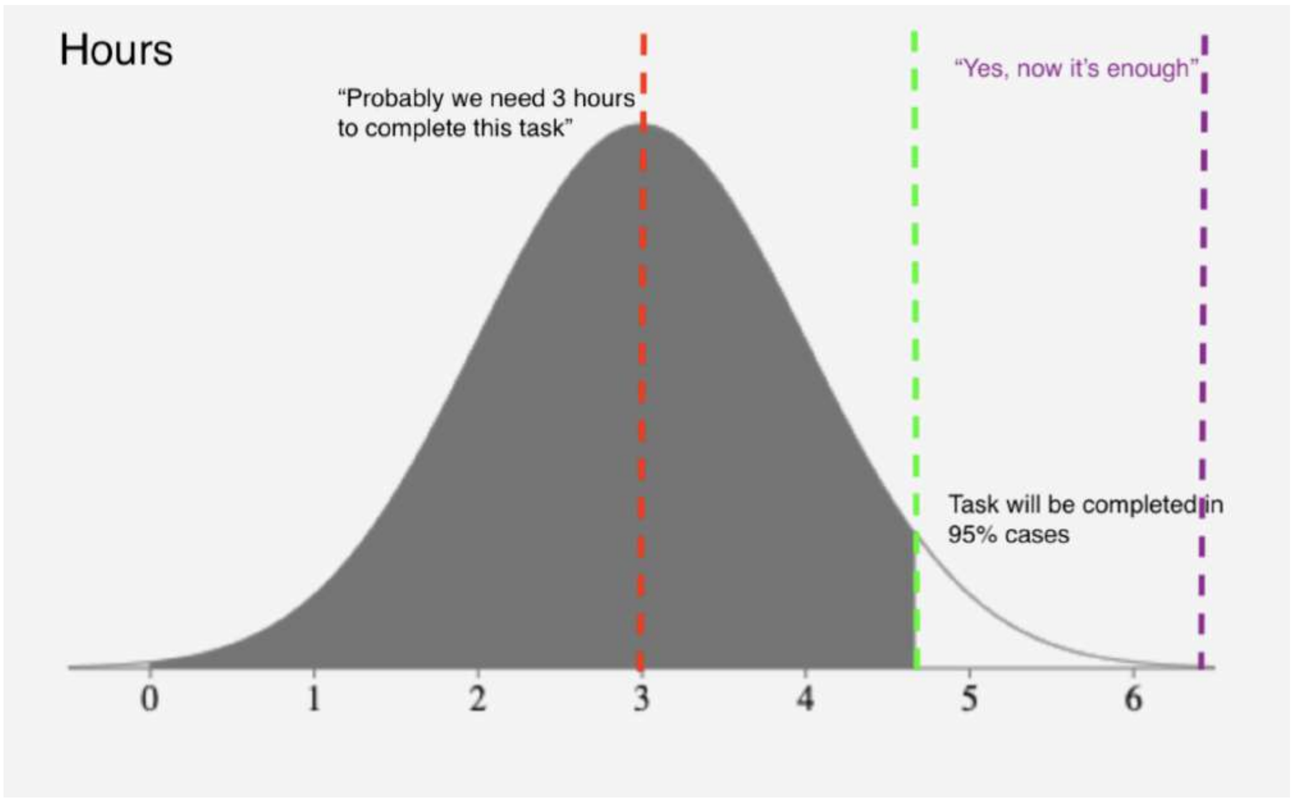


- Estimates given by experts are mean, not 95% quantile
- We don't know dispersion for the exact task

Most probable, we as managers want to have something better than a probability that the task will be completed in time in 50% of cases. But engineers make simple average value from what they work before on estimation of the given assignment.
The easiest solution for this case is to get value *3/2 using the 3-sigma rule. But not efficient from a performance perspective.

Stage 2: How engineers change tries to keep own safety - fail performance:

- Put time estimates to cover almost 100% of cases
- "work expands to fill the time available for its completion" (Parkinson's law)



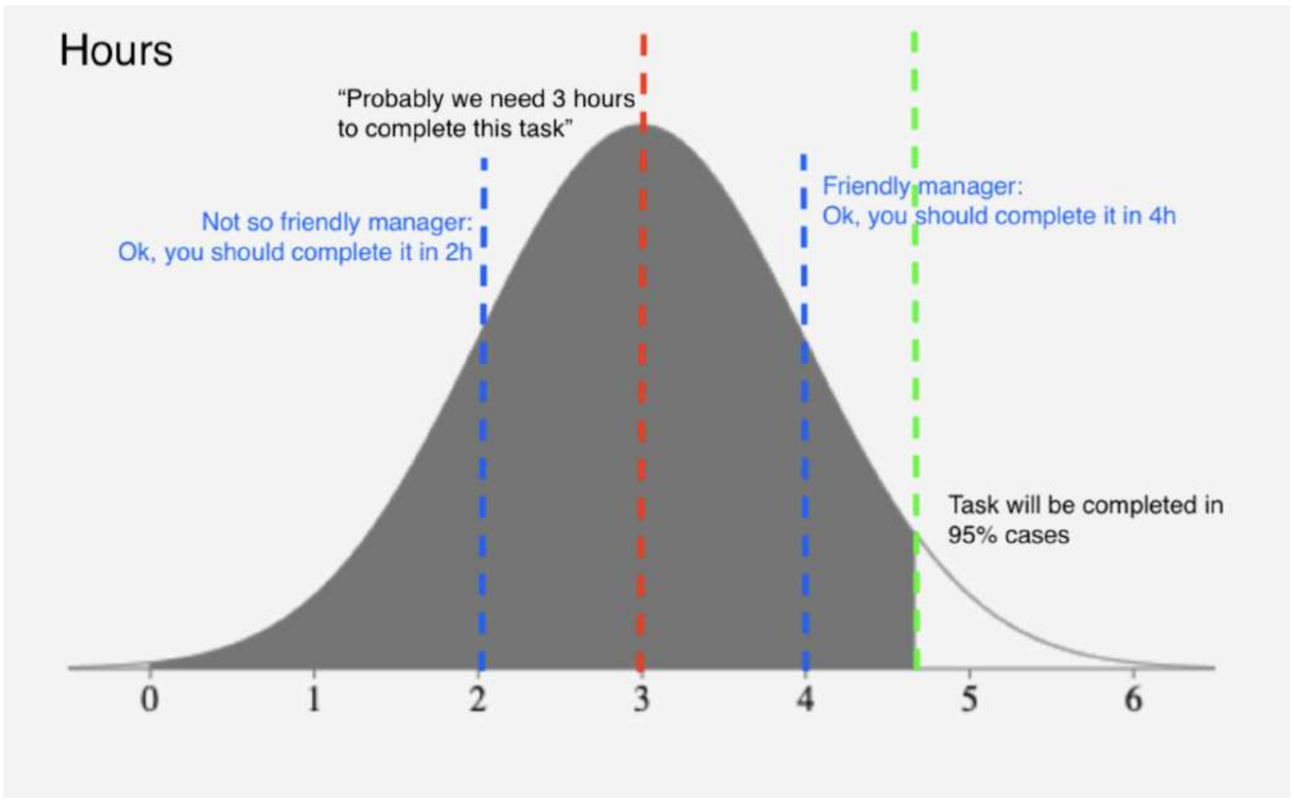
The second fact will break our desire to keep reliability reasonable with a need for tight estimates for better performance.

Even with excellent team spirit and total trust inside the team and between engineers and managers, it's hard to stay and work another 1-2 hours a day when the total amount of estimated time on tasks completed is more than 8 hours.

That's why managers try to push on engineers, whether it's possible. We are moving to the next step.

Stage 3: How managers try to rush - and fail forecasting

- Great target - better performance
- Bad outcome - no predictability at all
- Maybe overtime work to achieve the desired result in time
- More bugs and time spent on them – there is no work in research without bugs when you are rushed and hurry
- Depends on the manager – "good" or "bad", with different stage of rush, right or even left from average – confidence of time slot for work to be done with desired quality is not met

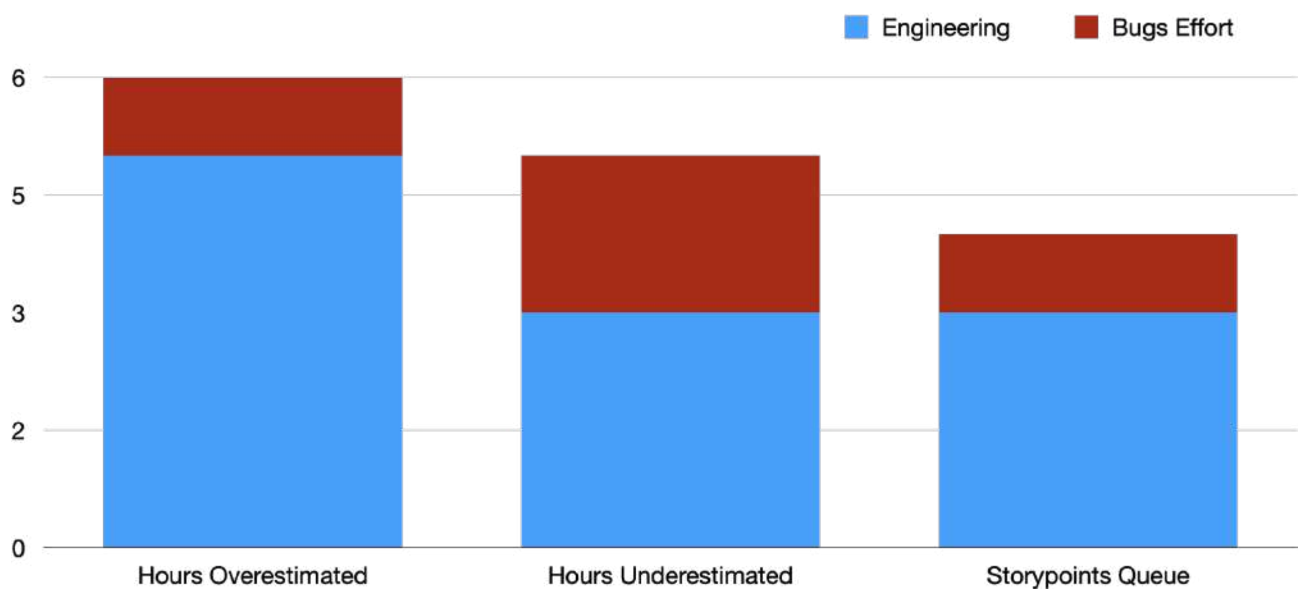


How usage of agile/scrum with story points resolve issues described above:

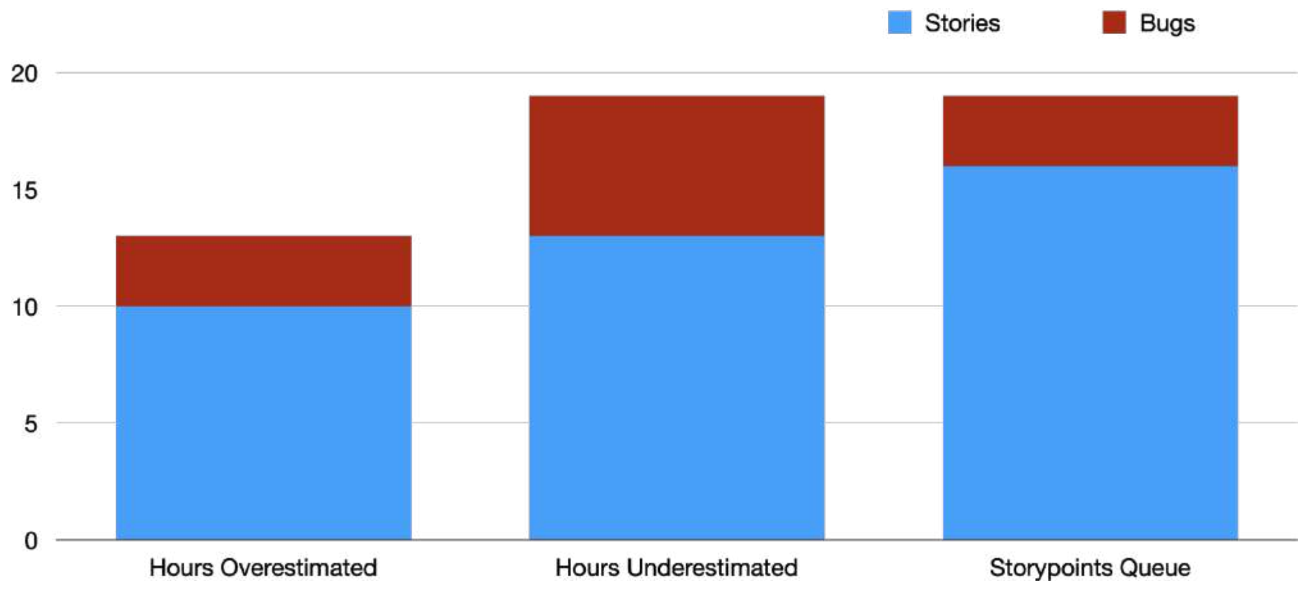
1. No conversion to hours, no linear formula – we removing the perception of work in story points as hours spent on work
2. Estimating both complexity and uncertainty – because both take time on the research and implementation stage
3. Calculating sprint capacity based only on previous sprints results, plus buffer size to strengthen predictability
4. An employee works at his own pace, take the next task if previously completed
5. No time micro-management, regular supporting activities already in scope – reduce time on control and process is easier to manage

Now let's look at our key indicators:

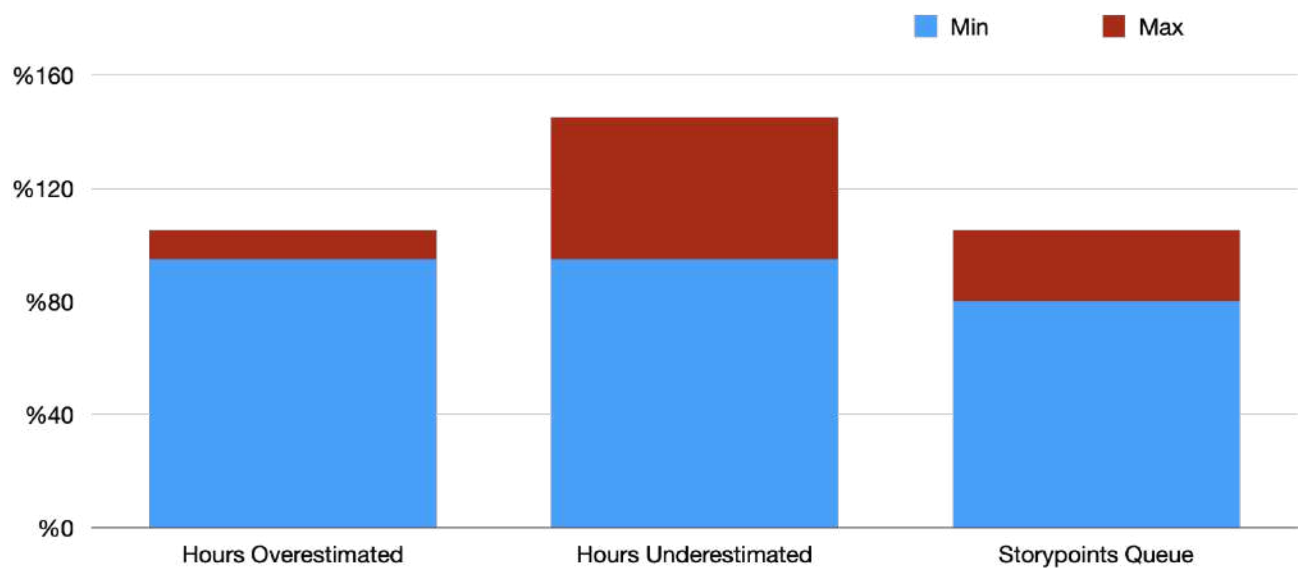
Performance, hours to complete a task (same example 3 hours on average, six-max in domain)



Sprint scope (in the middle, example on same data inputs) in the number of tasks completed:



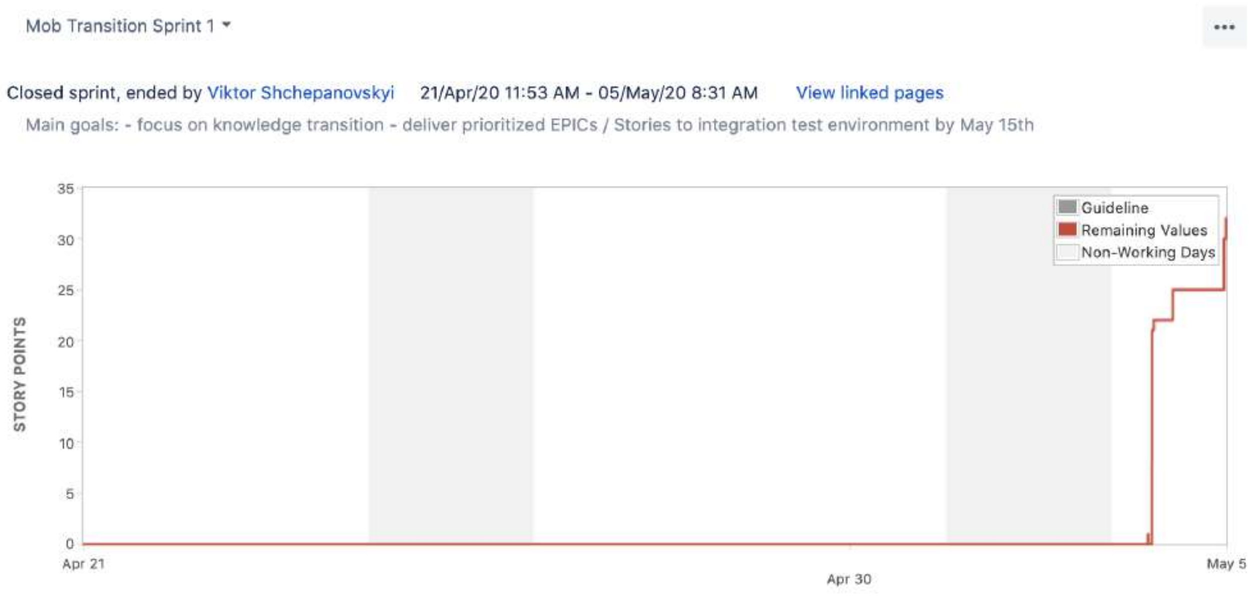
Forecasting, estimated time arrival for Sprint scope, % of the time of Sprint



As we see on comparison charts, the chosen methodology is aligned with our obligations and produces a stable workflow for performance improvements.

Agile/Scrum integration roadmap

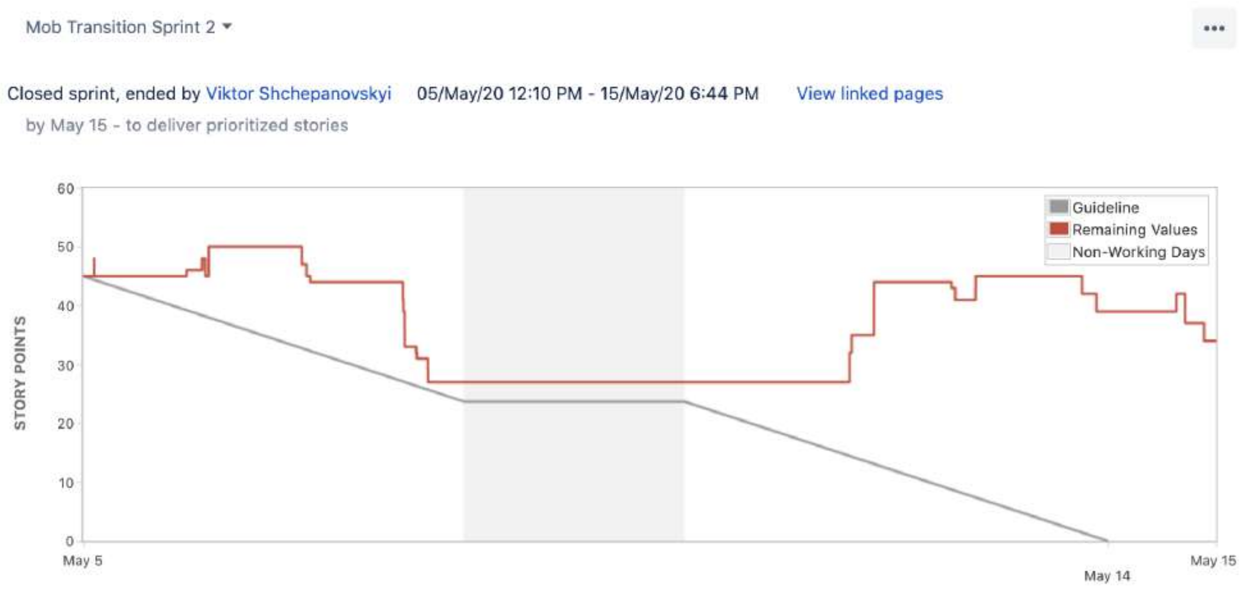
Stage 1: no forecasting, estimation of effort spent at the end of the Sprint


Mob Transition Sprint 1

Closed sprint, ended by Viktor Shchepanovskyi 21/Apr/20 11:53 AM - 05/May/20 8:31 AM View linked pages

Main goals: - focus on knowledge transition - deliver prioritized EPICs / Stories to integration test environment by May 15th

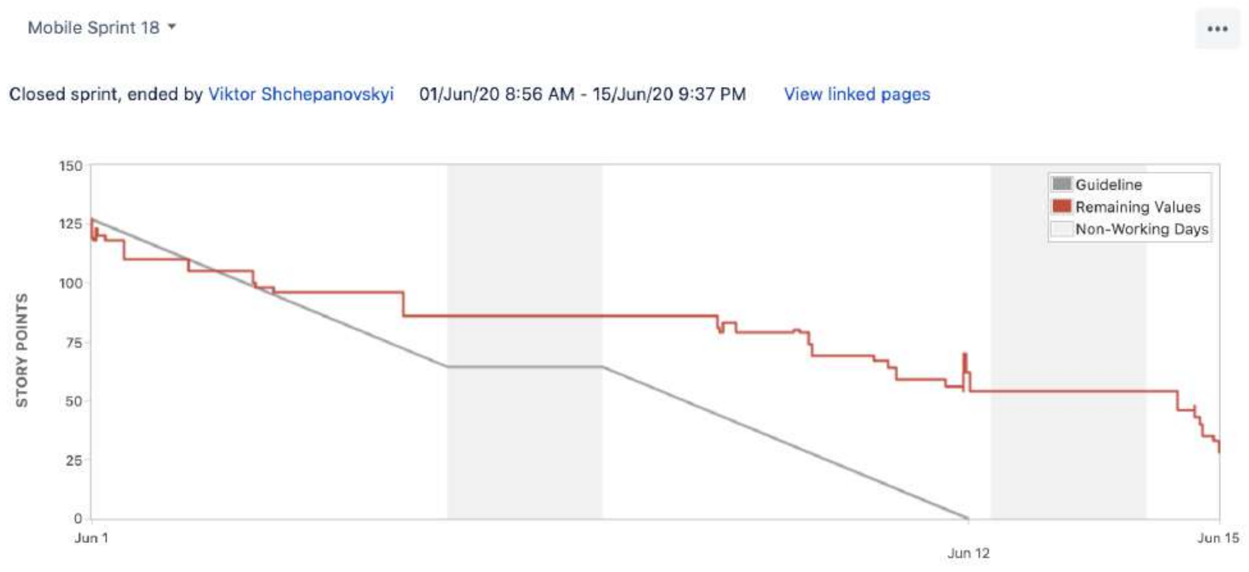
Stage 2: grooming and planning sessions added, but committed scope not defined; tasks added from backlog during Sprint


Mob Transition Sprint 2

Closed sprint, ended by Viktor Shchepanovskyi 05/May/20 12:10 PM - 15/May/20 6:44 PM View linked pages

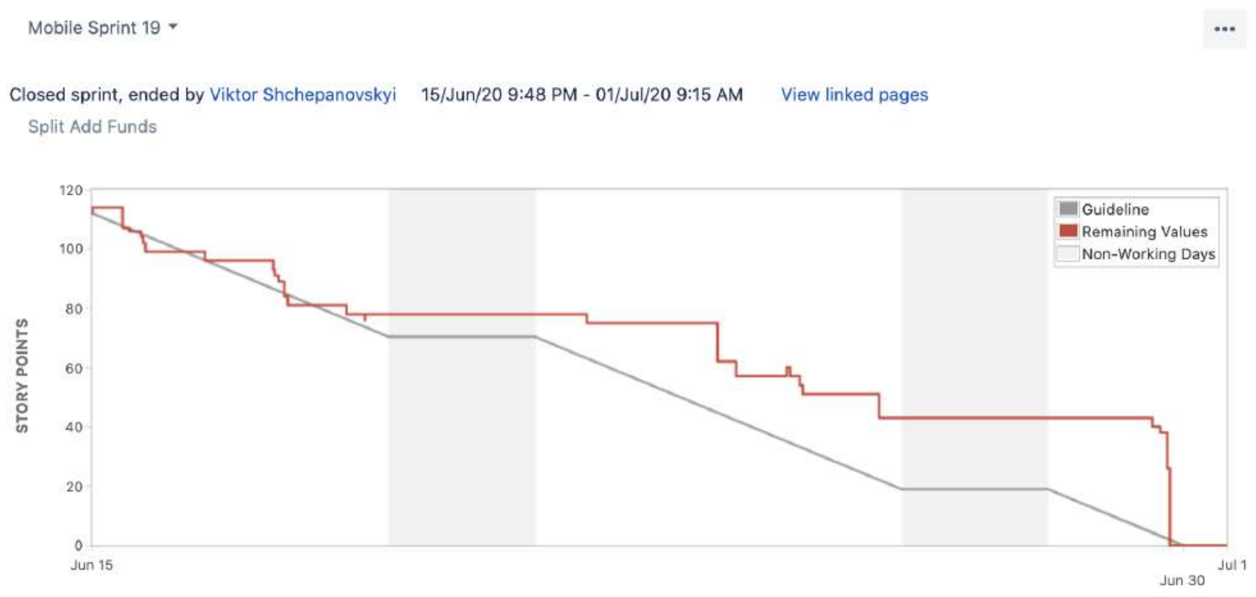
by May 15 - to deliver prioritized stories

Stage 3: scope defined, but buffer size is unknown


Mobile Sprint 18

Closed sprint, ended by Viktor Shchepanovskyi 01/Jun/20 8:56 AM - 15/Jun/20 9:37 PM View linked pages

Stage 4: buffer size covers uncertainty


Mobile Sprint 19

Closed sprint, ended by Viktor Shchepanovskyi 15/Jun/20 9:48 PM - 01/Jul/20 9:15 AM View linked pages

Split Add Funds

Scheduling connected tasks between teams: deep development process sync on ALAP

As late as possible, the ALAP methodology of synchronizing business activities relies on the connection between them on the latest expected timeframe.

Let's wrap up on how to sync was designed before to measure where improvements could be:

1. Each team works for two Sprints (2 weeks each) and deliver their parts separately
2. Definition of Done – component tested on mock data
3. After this period starts the integration stage, where all team deploy their parts to a dedicated environment, and Integration Testers check that pieces fit each other
4. If something broke – bugs created and added with the highest priority to current development Sprint

Issues present there:

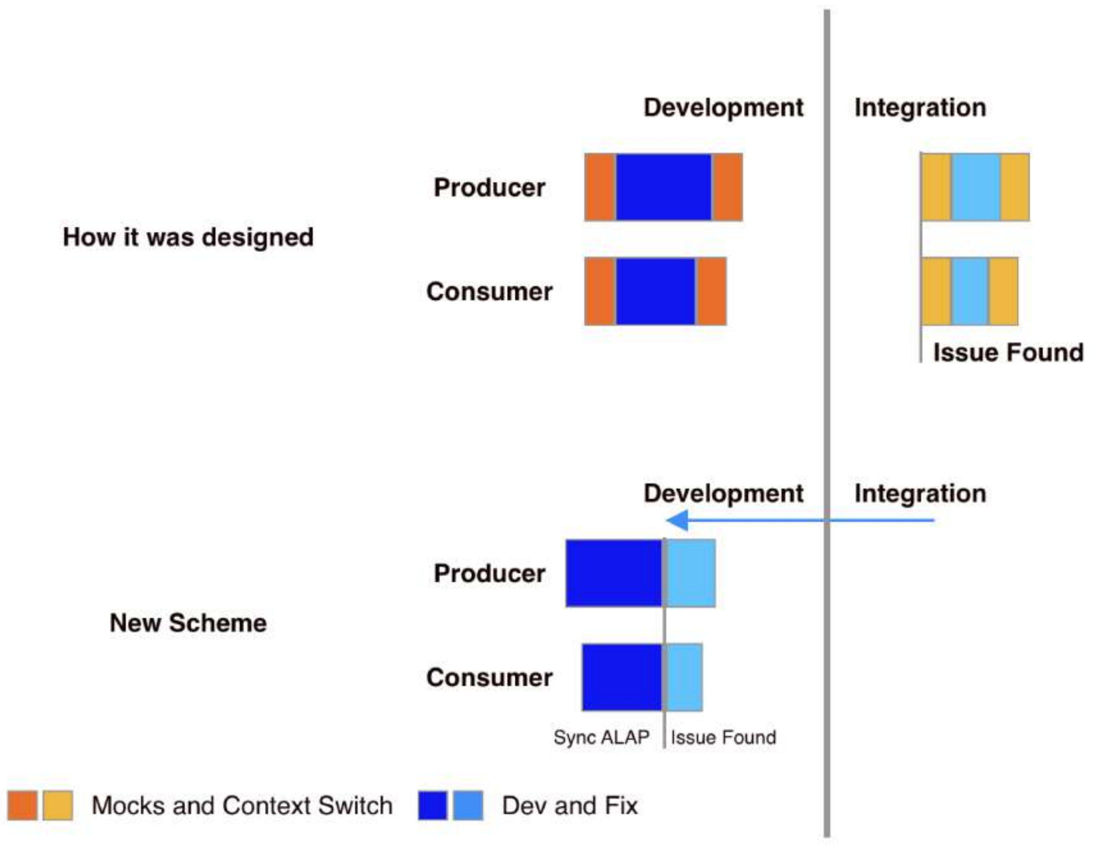
1. API contract defines an interface but does not always describe the best logic behind it.
2. If there is an integration issue: a bug was created, but there is still a problem to address it correctly.
3. Context switching is time-consuming, same for mocks.
4. Every team reports successful completion of their work, but the whole product doesn't work at the end of the period.
5. Problems are revealed at the latest stages. A more significant time gap between these stages produces a more extensive amount of time spent on fixing.

The primary solution for this case is to move the integration process itself from the testing stage to development:

1. Don't use mock data, no time to be invested there
2. Use real when it's ready
3. Manage dependencies between components during development
4. Fix issues or redefine API expectations during simultaneous work on both components
5. Schedule tasks using ASAP with priorities for the producer, ALAP for consumer
6. Do nothing on integration QA check stage, proceed with regression instead

Let's look at the timeline of how the development of dependent tasks performed before and after integration shift with ALAP:

1. Better performance numbers – no time spent on obsolete temporary code for mocks
2. More reliable, because problems revealed in early stages
3. The lower number of bugs inserted during Sprint to fix what also increases predictability of product development and gives the possibility to decrease the buffer size
4. Happy employees with a lower number of contexts switching and disruptions



Designed methodology aligned with our obligations from current state analysis and provided a clear route for managing the delivery of complicated Epics.

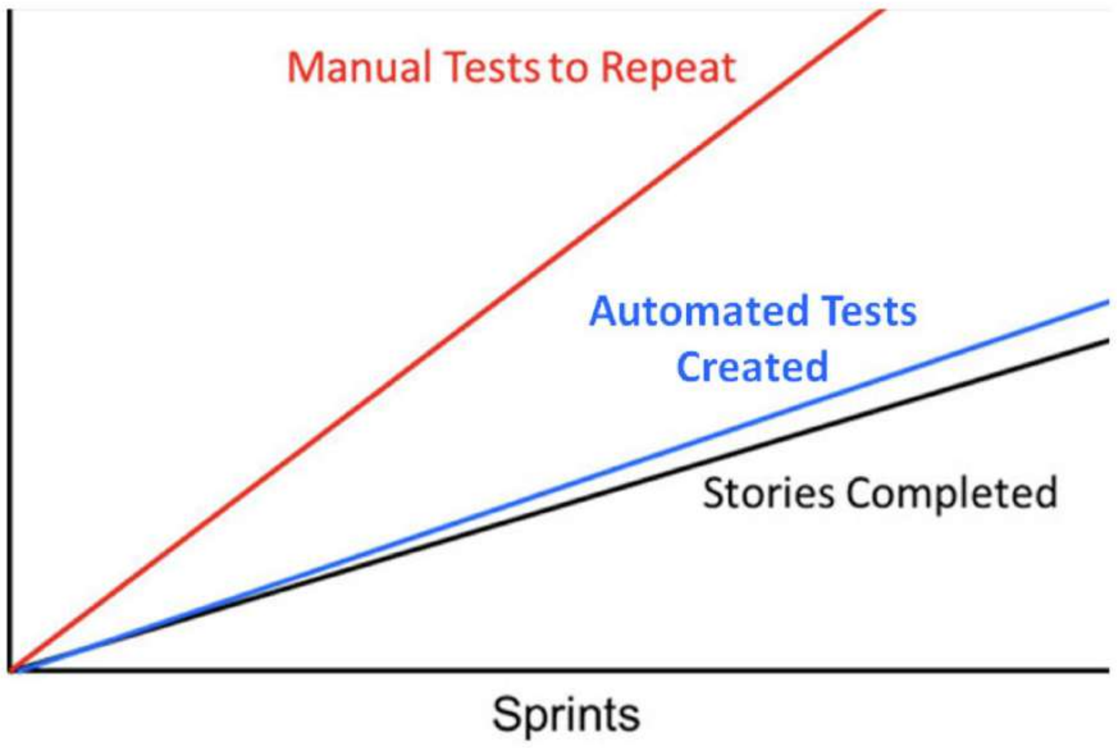
Quality assurance: Agile quality pyramid

The current state of tests scope in the project had the following structure:

- Some unit tests
- Some behavioral tests
- Some UI automation
- The enormous scope of tests for manual regression testing

I guess it easy to understand that word "some" should not be present in valuable parts of the quality assurance scope. But it's a widespread situation in the early years of software development in almost every company.

It leads to the next problem with the growing of scope in the project:

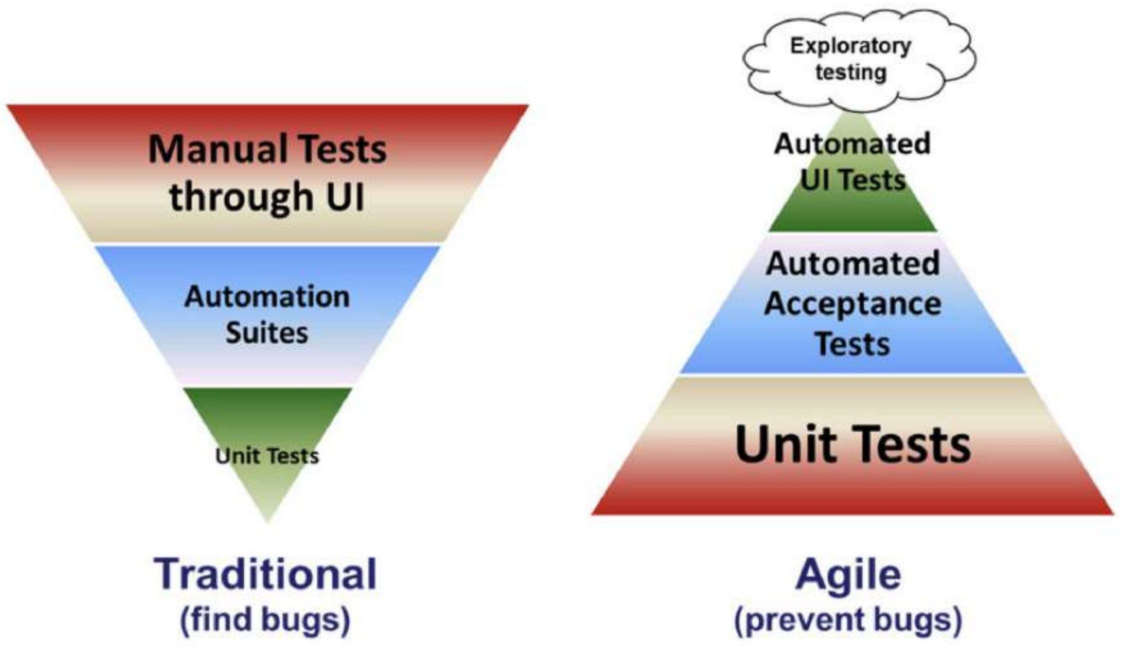


Which produce the following sequence:

1. The number of features increases every Sprint
2. And the number of test cases just adds to the common scope
3. The total number of cases grows as arithmetic progression

4. In just a few iterations, you need more time spent to run tests manually than to implement new features

Let's look at and compare the current state and desired



Migration from left to right could be complex, but it resolves future problems.

In general, migration items is following:

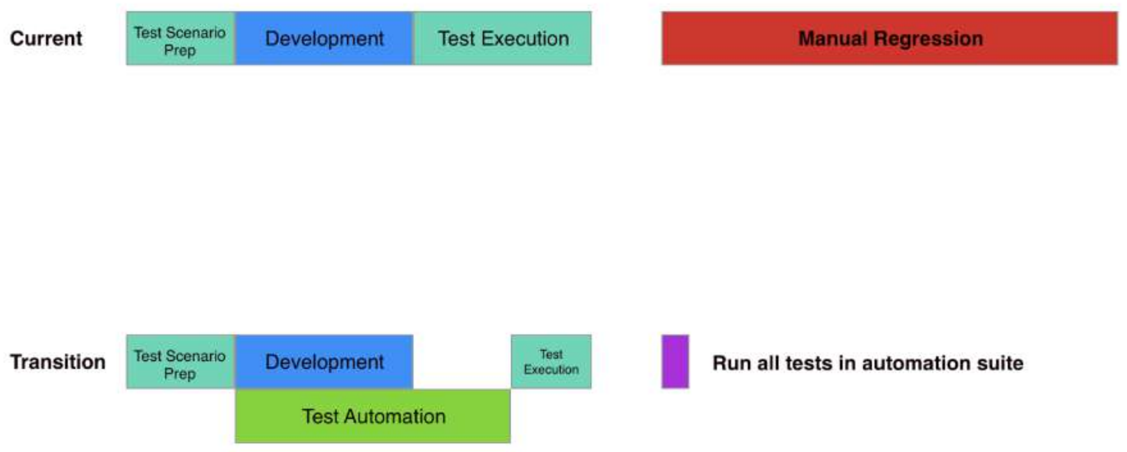
1. Make unit tests strong base for regression
2. Review quality of other test layers and complete them
3. Move from finding bugs to predicting bugs
4. Automate everything
5. Setup learning course for current testers team to migrate into automation area

Evaluation of targets reached:

1. Performance: We could spend time effectively when not lose it on massive manual regression
2. The cost of regression testing reduced dramatically
3. Time to market reduced
4. Reliability increased – time required to run on automation suite is more predictable than for manual
5. Team morale improved a lot because they wouldn't repeat the same checks every month

6. Also, it's a possibility for personal growth for testers

Let's review how testing activities are applied before and after planned change on the timeline



Designed testing scheme aligned with our obligations from current state analysis and reduced time spent on regression testing. Also, it gives us the possibility to run regression more often on-demand.

Coverage roadmap

	2020	**Q1 2021**	**Q2 2021**	**Q4 2021**
Unit tests	60%	70%	75%	85%
Behavioral tests	20%	30%	40%	70%
UI tests	20%	40%	50%	70%

IT architecture for the solution

Agile / Scrum on team level – just Atlassian Jira as the primary tool.

Same for tasks relations management – Jira connection and additional communication between teams with Slack and Zoom.

Testing automation has a more complicated setup:

- First layer: unit tests on each component, based on typical unit tests suite for the platform;
- Second layer: tests specification in TestRail;
- Third layer: Automation Framework, created on Java with Maven build and test scripts, using Appium for Mobile and Spring for Backend integration.

IT solution insights

The essential insight from all performed works and toolset usage is simultaneous movement into technical improvement and organizational process improvement streams.

Precisely for this case, there was no magic SaaS that resolves these issues and improved performance.

But we have a marvelous instrument both from the Software and Managerial sides to achieve performance improvements, measure them, and move forward.

Atlassian Jira and Confluence is a great tool to track a complicated project with different streams. Still, the main limitation is managers who operate on Epics and Tasks.

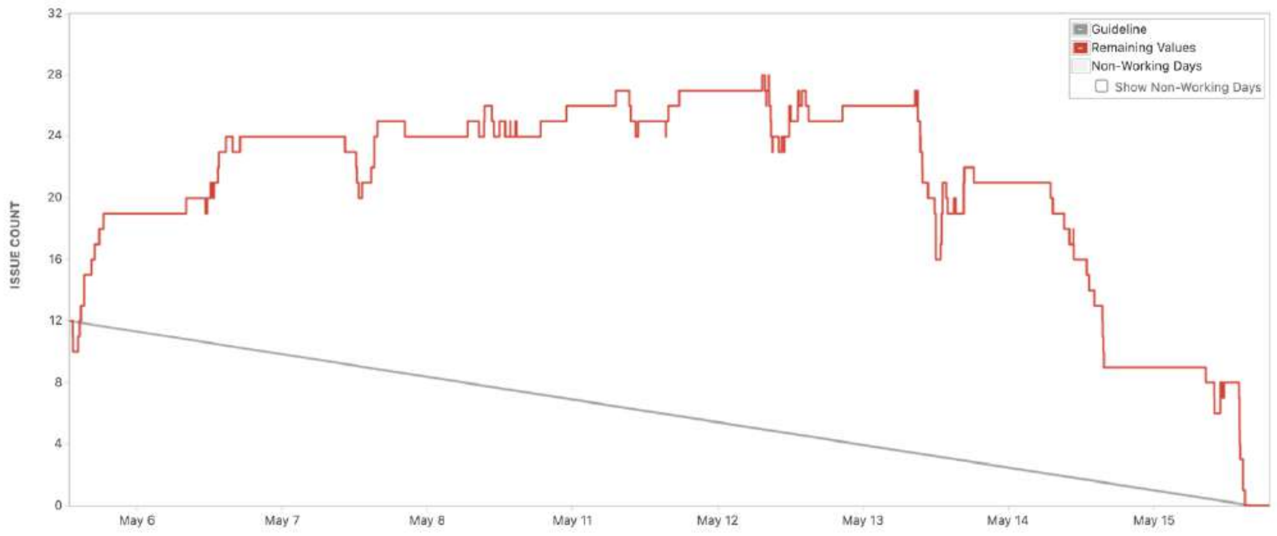
From automation stream insights, we've got a clear understanding of how important is the proper set up of quality assurance people and tools in the start of the project to be confident with quality in later stages.

Analytical calculations

Agile / Scrum integration

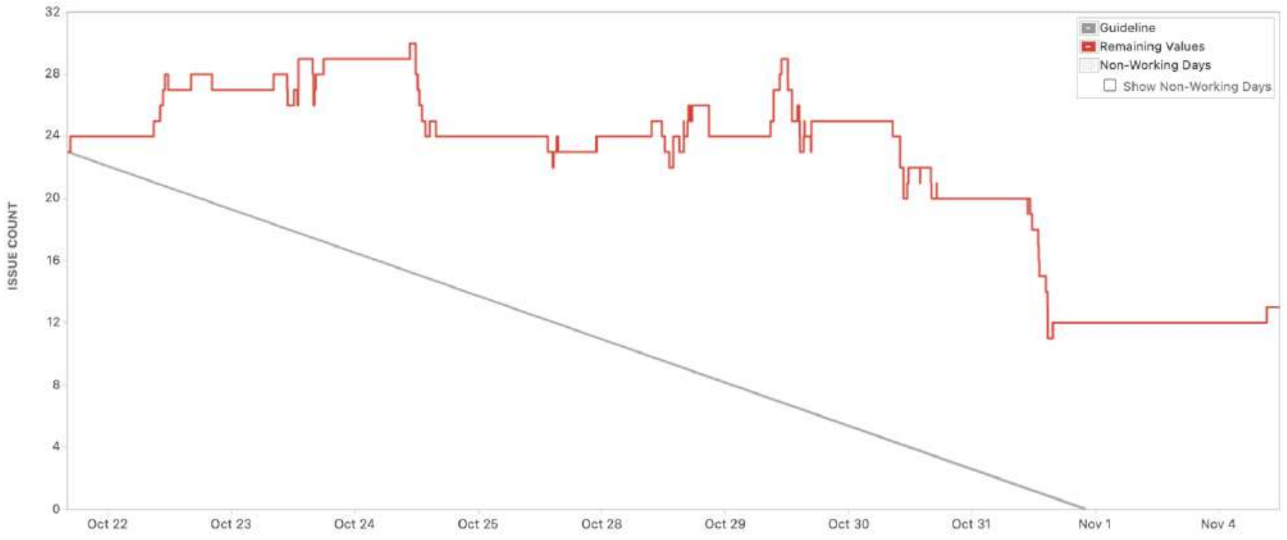
It's impossible to compare business hours spent and Story points completed in Sprints for the previous team and ours. But we know that the distribution of complexity of tasks inside sprints didn't change a lot. We could try to compare the number of tasks planned and completed based on Jira burndown reports.

Let's look at average sprint examples from the previous team:



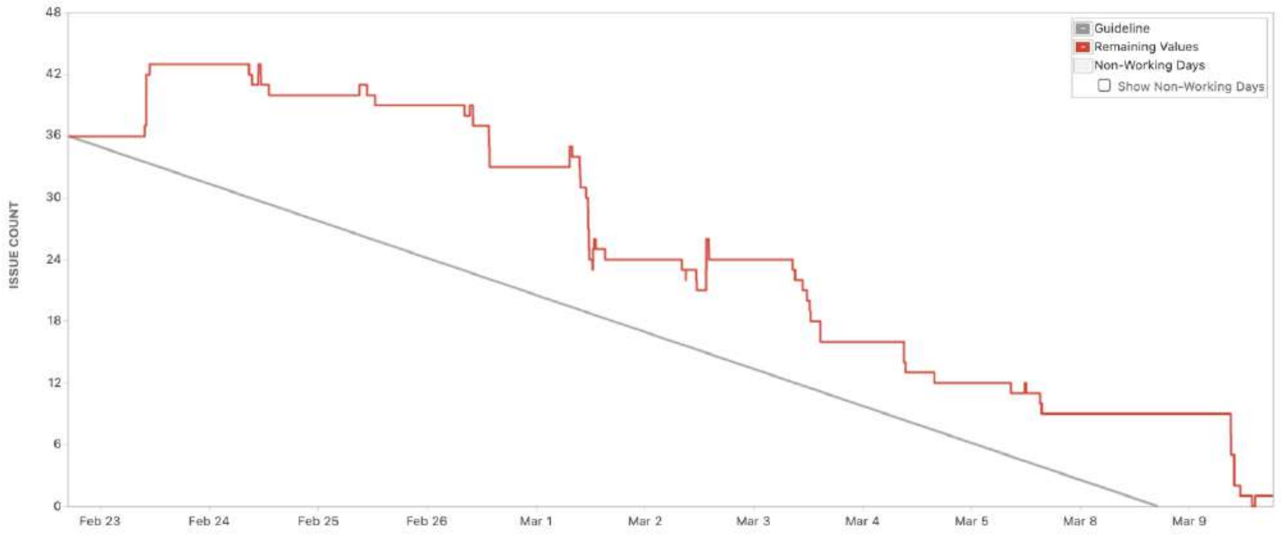
It looks like Sprint's start scope was not properly planned, and a lot of tasks were added during Sprint. Not good practice at all, but at least they accomplished development goals. The top margin of issue count is 24 tasks.

Second chart:



Here we see that team was very accurate with scope preparation, have 24 tasks at the start of the Sprint, which is typical for them—but not finished Sprint successfully with 12 tasks leftovers.

Let's look at our usual Sprint (there is a holiday on March 8th):



The top margin is 42, starting scope is 36 tasks which are average capacity when we expect that some urgent requests will be added.

Comparing 24 and 36 tasks on average per sprint capacity planned with the same complexity distribution – we could conclude that performance was improved more than 30%.

But this is not the only effect of integration of Agile/Scrum, but also the impact of changes on the program level with scheduling tasks.

Scheduling connected tasks with ALAP

Here we will operate with average values:

Let's take average time to complete a task that needs API from the backend team. It is ~2 days based on internal team calculations. Time spent for mocking data and removing mocks in the future on cleanup is 4-6 hours.

Also, context switching from current project state to integration state and environment with dive into what happens takes ~1 hour. Moving back to the current state is 0.5-1 hours.

The average time to fix some issues is 1 hour, with the average number of cases per story – 1.5.

Let's calculate how much time we will spend on integration on later and earlier stages:

1. Late: 16 + 4 + 1.5 + 1.5 = 23 hours min (25.5 max)
2. Earlier: 16 + 1.5 = 17.5 hours

The performance improvement ratio is 24-31% on related tasks. In independent studies, we stay with the same values.

Quality assurance automation

We'll just mention here that automated regression run will take 4-6 hours on average, comparable to 1-month time for manual regression run for the whole test case base with the participation of 4+ manual testers.

Some investments for education are needed, but in general, time exchange from manual testing to the creation of automated test scenarios should not affect sprint velocity at all.

Finance

Based on our department financial statements, the budget for a year for our team is $523 000 (per the first year). Let's be a skeptic and suggest that performance improvement from stage 2 is already included in 30% improvement for the first task.

That means we produce the same engineering product outcome as a larger team or same team on a more significant period and cost bigger by 30%.

Direct "profit" of process improvements (without QA automation) is $156900.

QA (integration team) manual effort on regression is four testers per year, let's put skeptic salary $2000. Then we achieve 2000 * 4 * 12 = $96000, or 18% of Mobile team annual budget.

Costs and resources needed

Agile / Scrum integration:

No additional costs.

Scheduling related tasks with ALAP:

No additional costs.

Quality assurance automation:

Cost of training for testers for 6 months (approximately 4 * 2 * 100 = $800, cost of device farm on cloud ($200 monthly). Annual cost is 12 * 200 + 800 = $3200 for current year.

Risk – mitigation matrix

Agile / Scrum integration

Severity \ Likelihood	Low	Medium	High
Low	-	-	Ambiguous task description – fix on flight
Medium	-	Scope changed during Sprint – remove tasks with lower priorities	Scope not completed at the end of the Sprint – use buffer for planning
High	Dependency incomplete – freeze and get another task from the backlog	Vacations, sick leaves – plan according to the projected presence	No issues categorized here

Scheduling connected tasks with ALAP

Severity \ Likelihood	Low	Medium	High
Low	-	-	Ambiguous task description – fix on a flight, agree on changes with other teams if needed
Medium	-	Scope changed during Sprint – remove tasks with lower priorities and without dependencies	Scope not completed at the end of the Sprint – use buffer for planning; Split tasks
High	Dependency incomplete – freeze and get another task	Vacations, sick leaves – plan according to	Deployment delays – proceed work on local environments

	from the backlog, inform program management	projected presence, inform other teams	

Quality assurance automation

Severity \ Likelihood	Low	Medium	High
Low	-	-	Ambiguous task description – align on behavior with engineers
Medium	-	Part of tests become obsolete – regularly cleanup	Scope not completed at the end of the Sprint – move tests execution to next Sprint
High	Automation QA teammate leave company – hire a new one, reduce the scope for a period	Automation toolset become overwhelmed – refactor and cleanup	Impossible to test with automation in some case – test manually

CHAPTER 3. IMPLEMENTATION AND CHANGE MANAGEMENT

The most challenging task in all designed changes is changing how people work and how other managers do their job.
No tangible assets were created or moved. Only process things and some educational courses for testers are primary artifacts after all changes.

Methodology for implementation

Agile / Scrum integration: team level, top-down change

First – I discussed what I would do with the main stakeholders and achieved approval for moving forward.
Second – discussed with the team what's pros and cons; what is beneficial for them with change.
And for the next four sprints, we gradually align our rituals with the desired state.

Scheduling connected tasks with ALAP, program level, bottom-up change

As for the previous case, an agreement with all stakeholders was achieved.
Then we move into one side implementation of the new scheme on the Mobile side.
After two sprints, we were ready to align with other parts of the company on the used scheme.

Quality assurance automation, program level, top-down change

Agreement with stakeholders, with sharing leadership role in this migration with automation QA lead.
We produced a plan for the education process, gap analysis, and gap coverage in the next half a year.
Education parts are connected with regular practice.

Project implementation plan

The implementation plan is following sequence:

1. Integration Agile/Scrum methodology on a team level with double counting for program level
2. Changing measuring capacity procedure on program level
3. Changes in scheduling connected tasks in one direction (consumer only)
4. Application of this change on program level
5. Migration to two-sides relations management
6. Nailing it in common procedures
7. Initiation of migration to automation scheme in one team
8. Involvement of automation team engineers and mobile engineers to education and development process
9. Expansion to one more team
10. Gap analysis for cases left
11. Complete automation suite with end-to-end scenarios

Quality assurance automation, the initial plan for Mobile team:

0. Reserve capacity for QAs education and practice
1. Validate and edit the "First Day "page for automation to be complete instruction for setup
2. Perform introduction session with Automation Team representative and Mobile QAs
3. Setup environments, IDE's, suite, and Repo for Mobile QAs
4. The initial coding session with the Automation Team representative
5. Define the scope for first automation tasks for the Mobile team
6. Validate current Test Scope in the Automation suite
7. Create jobs to cover missing mobile parts in the Automation suite

Change management components

Stakeholders map

Stakeholder	Interest	Influence	Manage
Director of Engineering	Low	Medium	Inform
Delivery Manager	High	High	Partner
Program Manager	Medium	High	Involve
Integration QA Lead	Medium	Medium	Consult
Automation QA Lead	High	Medium	Consult/Involve
Automation QA team	High	Low	Consult
Manual QA team	High	Low	Consult
Software Engineers	Low	Low	Inform

Managing change steps

Step One: Create Urgency
Inform stakeholders from the table above about inefficiencies found. Show clear understanding that the delivery process could be improved without losses. Produce a report about KPIs reached by performed improvements.

Step Two: Form a Powerful Coalition

Create a coalition with the Delivery Manager, Program Manager, and Quality assurance leads (manual and automation)—split responsibilities.

Step Three: Create a Vision for Change

Create a "target state" processes plan for agile/scrum integration, dependent tasks completion, and automation strengthening. Based on it, create a transition plan listed in this project paper.

Step Four: Communicate the Vision

Deliver vision created on the previous step to all listed stakeholders. Communicate with each of them his/her role in ongoing and future processes.

Step Five: Remove Obstacles

Identify individuals, traditions, legislations or physical obstacles, or any other barriers blocking change's path. Rely on available resources to break them down without disrupting any other areas of the business.

Step Six: Create Short-Term Wins

Celebrate first achievements gathered by Agile/Scrum integration; show the successful reduction of the number of bugs on integration/regression stages and productivity updates.
Create educational milestones with an appreciation of obtained knowledge.

Step Seven: Build on the Change

Sustain and cement the processes change for long after it has been accomplished. Keep setting goals and analyzing what could be done better for continued improvement. Continue contributing automation suite test coverage.

Step Eight: Anchor the Changes in Corporate Culture

Incorporate new development, synchronization, and testing schemes into the onboarding process. Promote future improvements based on delivered changes.

Organizational culture

There is no prevalent culture in the company; each team or department is different because it was created as the decentralized scope of parts outsourced in other companies and countries.

This is an issue to operate new features or add new teams because each time, newcomers should be ready to work in a different culture, even when they will work with a diverse group but no other company.

Few times this produces miscommunication or misunderstanding of who or how something should be implemented.

When I joined the company in April, it was straightforward for us to see differences in culture because it directly impacts how people manage other people and how people manage their work.

For example, overtime work is prohibited in my team just because I want to know the typical velocity of a team and appreciate that people spent their time rest and be fresh at work.

In the Minsk team, we have a different situation, when people regularly work overtime, sometimes on weekends. Because they are likely to think that this additional time they worked is a virtue and will be appreciated despite lower quality of work and unpredictable completed scope at the end of the Sprint.

There is a difference even in measuring work: they count each hour and sometimes minutes to control the minor time employees spent. In our team, estimations are not strict and don’t mean that tasks will take exactly described time; At the end of the day, there is no requirement to be at work 8 hours.

Motivational strategies

The company is in the growth stage now; MVP was released just this spring. Based on this, there are minimal applications of motivation theories in the organization.

At C-level, we have no strict directive to manage the team one way, a massive opportunity for independent practice. But on another side, we have no guidance and no visible targets to achieve in people management.

On the team level, I practice some of the elements of the following theories:

Equity: there is a strong correlation between performance and appreciation + salary. As a person with a tremendous technical background, I could evaluate the quality of work done and general employee performance. Based on these characteristics and the 360-feedback person gets or not salary increased. And all teammates know that this is how it works.

Expectancy: From the very beginning, I've got a message for a team that they are expensive but the most qualified people on the labor market, and this was true. Nevertheless, I keep this feeling that the performance and quality of work of a team expected to be much better than average.

Goal-setting: This aspect is described in the process of how we manage our work cycles. We have two weeks of Sprints, and each Sprint has significant goals. These goals are indicators of a good job done over the period. At the start of the Sprint, the team commits that they will complete the defined scope.

CHAPTER 4. CONCLUSION/RECOMMENDATIONS

We achieved significant performance improvements at the team level, up to 30%, without any additional costs spent. Even with applied managerial tools for process improvements, it wouldn't be successful without our leadership teammates' involvement and active support of engineers' changes in my team.

Collaboration between teams improved a lot, at least for 24% performance gained. But even more achieved on the communication level, with deeper integration of different development departments we reached better work perception as contributors to common company's goals, instead of being separated and limited in local silos.

Quality assurance was my weakest expert area at the start of this project, but using the best industry practices and guidance of my automation colleagues, I've learned a lot, and we as the company are still obtaining benefits there.

But I will be fair and should say that there are limitations to my work's application methods.

For example, you could not apply agile/scrum to projects with a specific execution time. It does not apply to projects with overwhelming complexity, for which predicted average time could not be used as the target for planning. The tasks with indefinite results passing away to some version of Kanban too.

For deep development-time integration, there is a limitation in the number of components. It works best for 2-3 teams, or let's say to not more than 3-4 relations. If your project development cycles contain more of that, then other development & testing synchronization technics should be applied. With moving away from management in software engineering to (for example) manufacturing, it became more complicated. Process architecture depends on the industry and its constraints and main processes under the hood.

Nevertheless, for many software companies, maybe except corporate giants, this 3-4 relations setup is more than enough.

Testing automation is limited first in the software domain. In physical goods manufacturing, it could be much more complicated to produce machines that could test goods. In some areas like car parts or assembling of complex devices, it's already in use, but much more expensive than our case.

Also, limited application to project stage: it's not required for sure when you try to produce MVP for investors, or vice versa – for an old legacy system which planned to be replaced soon, it could be just a waste of time.

Areas for improvement in this project now separated into the following domains:

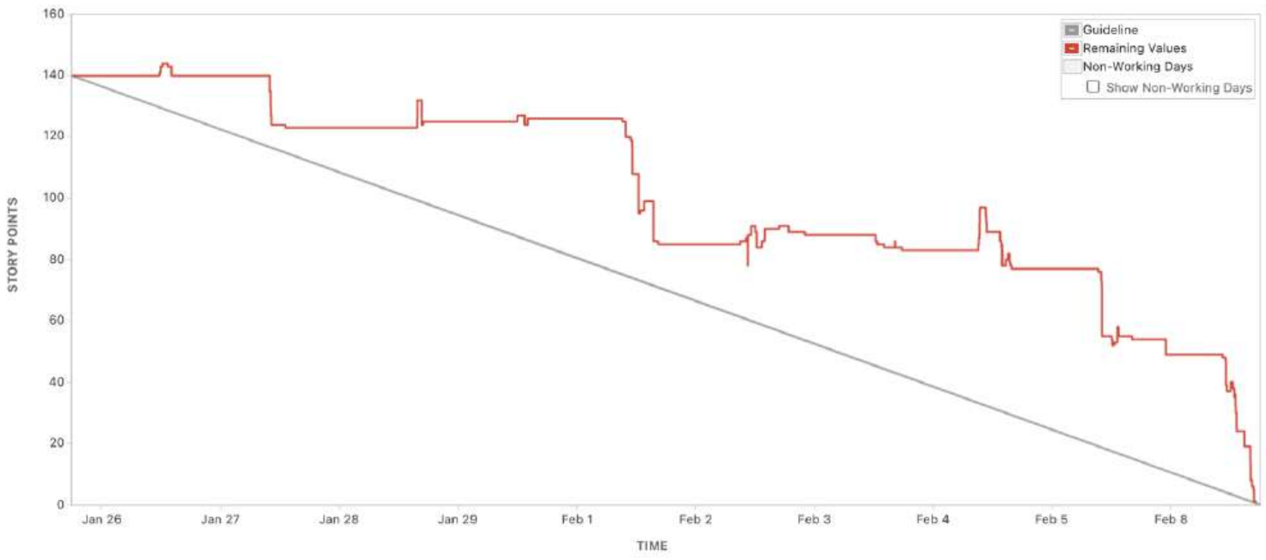
1. Technical research: toll gates detection improvements, battery drain, optimizations
2. Fastlane and hotfix procedures
3. Scale-up structure when the company will grow in number of employees: cross-functional teams, guilds, or any other way we consider the best fit

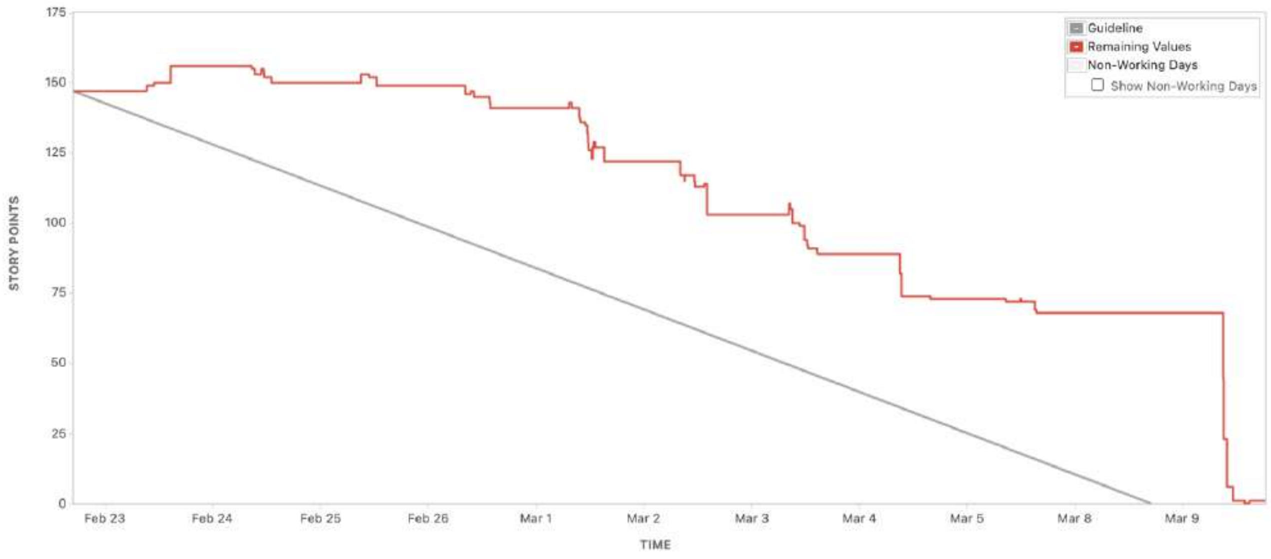
BIBLIOGRAPHY

1. The goal : a process of ongoing improvement - Eliyahu M Goldratt; Jeff Cox
2. Critical Chain - Eliyahu M Goldratt
3. The Scrum Guide - Ken Schwaber and Jeff Sutherland
4. Scrum: The Art Of Doing Twice The Work In Half The Time - Jeff Sutherland
5. Agile Software Development, Principles, Patterns, and Practices - Robert C. Martin
6. Clean Code: A Handbook of Agile Software Craftsmanship- Robert C. Martin
7. Test-Driven: TDD and Acceptance for Java Developers - Lasse Koskela
8. Leading Change - John P. Kotter

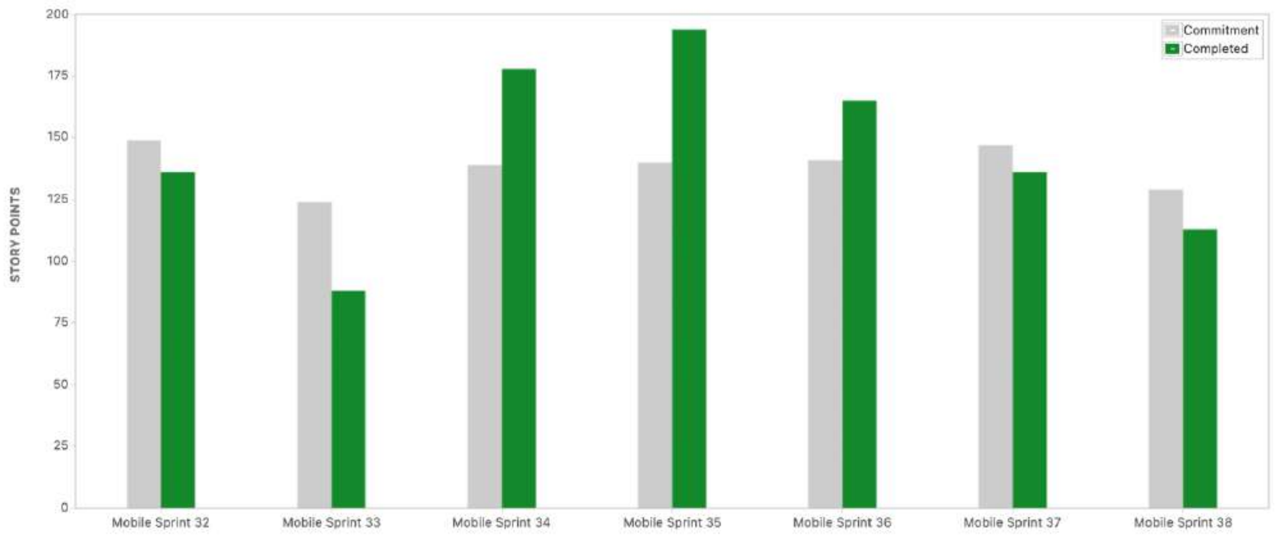
APPENDICES

1. Latest sprints burndowns in Story points





2. Latest velocity report (vacations in 37/38)



3. Expenses structure (2020)

Expenditures (OpEx)

Salaries	USD	456 000
Hardware	USD	20 000
Software	USD	3 000
Office rent	USD	36 000
Insurance	USD	4 000
Total	USD	**519000**

4. Department org structure

	Quantity
Department Lead	1
Product Manager	1
Designer	1
Android Lead	1
Android Engineer	2
iOS Engineer	2
QA Engineer	2

5. Income Statement of Department

Sales	**USD**	**523 000**
Cost of Sales	USD	0
Gross Profit		**523 000**
Gross profit margin		*100%*
Operating expenses	USD	519 000
Operating profit		**4 000**
Operating margin	*%*	*0,76%*
Depreciation/Amortization	USD	**4 000**
EBIT	**USD**	**0**
EBIT margin	*%*	*0,00%*
Interest Expenses	USD	0
Earnings before taxes (EBT)	**USD**	**0**
EBT margin	*%*	*0,00%*
Tax rate		*5%*
Taxes	USD	0
Taxes/EBT	*%*	*0%*
Net income from operating activities	**USD**	**0**
Net income margin	*%*	*0,00%*

6. Future improvements backlog:
    a. Separate releases per platform
    b. Real-time tolling accuracy improvements
    c. Real-time tolling configuration tools updates
    d. Continuous educational program for employees
    e. Migration to the new simplified business model