|  |  |
| --- | --- |
| **SCRUM: The Story of an Agile Team**  Scrum is one of the most heavily used agile techniques. It’s not about coding; instead, it focuses on organization and project management. If you have a few moments, let me tell you about the team I work with, and how we adopted Scrum techniques. | **SCRUM: История гибкой команды**  Скрам – одна из наиболее широко используемых техник гибкой разработки. Она не касается кодирования; вместо этого она фокусируется на организации и управлении проектом. Если у вас есть некоторое время, позвольте мне рассказать вам про команду, с которой я работаю, и то, как мы применили техники скрам. |
| **A Little History**  *“Scrum’s roots actually extend beyond the Agile era.”*  Scrum’s roots actually extend beyond the Agile era. The first mention of this technique can be found in 1986, by Hirotaka Takeuchi and Ikujiro Nonaka, for commercial product development. The first official paper defining Scrum, written by Jeff Sutherland and Ken Schwaber, was presented in 1995.  Scrum’s popularity grew shortly after the 2001 publication of the [Agile Manifesto](http://agilemanifesto.org/), as well as the book [Agile Software Development with Scrum](http://enva.to/UDiTUu), coauthored by Ken Schwaber and Mike Beedle. | **Немного истории**  *“На самом деле корни скрам тянутся за пределы эры гибкого программирования.”*  На самом деле корни скрам тянутся за пределы эры гибкого программирования. Первое упоминание об этой технике может быть найдено в 1986, в работе Хиротака Такеучи и Икуджиро Нонака о разработке коммерческого продукта. Первая официальная статья, определяющая скрам, написанная Джеффом Сазерлендом и Кеном Швабером, была издана в 1995 году.  Популярность скрам возросла после публикации в 2001 году [Agile-манифеста](http://agilemanifesto.org/), а также книги [Agile Software Development with Scrum](http://enva.to/UDiTUu), соавторами которой стали Кен Швабер и Майк Бидл. |
| **A Few Facts**  Scrum defines a set of recommendations, which teams are encouraged to follow. It also defines several actors – or roles, if you prefer that terminology – together with an iterative process of production and periodical planning. There are several tools, which accommodate the Scrum process. I will reference a few in this article, but the most powerful tools are the white board and sticky notes.  *“There is not, and never will be, a list of “Scrum Best Practices,” because team and project context trumps all other considerations. — Mike Cohn”*  *The Roles*  Everything starts with the pig and the chicken. The chicken asks the pig if he is interested in jointly opening a restaurant. The chicken says they could call it, “Ham-and-Eggs.” The pig answers, “No thanks. I’d be committed, but you’d only be involved!“  That’s Scrum! It specifies a concrete set of roles, which are divided into two groups:   * Committed – those directly responsible for production and delivery of the final product. These roles include the team as a whole, its members, the scrum master, and the product owner. * Involved – represents the other people interested in the project, but who aren’t taking an active or direct part in the production and delivery processes. These roles are typically stakeholders and managers. | **Несколько фактов**  Скрам задает набор рекомендаций, которых команде следует придерживаться. Также он задает несколько актеров – или ролей, если вы предпочитаете такую терминологию – и итеративный процесс производства и периодического планирования. Существует несколько инструментов, позволяющих приспособить процесс скрам. Я буду ссылаться на некоторые из них в этой статье, но наиболее мощными являются белая доска и липкие заметки.  *“Нет и никогда не будет списка «Лучших практик скрам», потому что команда и контекст проекта превосходят все остальные соображения. — Майк Кон”*  *Роли*  Все начинается со свиньи и курицы. Курица спрашивает свинью, интересна ли ему идея совместного открытия ресторана. Курица говорит, что они могли бы назвать его «Яйца с ветчиной». Свинья отвечает: «Нет, спасибо. Я буду приверженным, а ты будешь лишь участвовать!»  Это и есть скрам! Он определяет конкретный набор ролей, которые поделены на две группы:   * Приверженные – непосредственно ответственные за производство и предоставление конечного продукта. Эти роли включают команду в целом, ее участников, скрам-мастера и владельца продукта. * Участники – представляет остальных людей, заинтересованных в проекте, которые не принимают активного или прямого участия в производстве и предоставлении. Эти роли обычно закрепляются за акционерами и менеджерами. |
| **This is How We Started**  Everything depends on dedication and good will. If you want your team to be efficient, productive, and deliver on time, you need someone to embrace some form of Agile techniques. Scrum may or may not be ideal for you, but it is surely one of the best places to start. Find that someone on your team who is willing to help the others, or you, yourself, can take on the responsibility of introducing Scrum.  *“You may ask why you should care how another team, like mine, does Scrum. You should care because we all learn how to do Scrum better by hearing stories of how it has been done by others – especially those who are doing it well. – Mike Cohn”*  The talented team I work with already knew a lot about Agile. We switched from Waterfall development to a more agile process, and released quite frequently. We successfully managed to release every three to six months, having a decently low number of bugs after each release.  But, still, we were far from what we can achieve today. We missed the process, or rules, that would force us to change our perspective on the product and process. That was the moment when our team manager introduced us to Scrum, a term we, at that time, had never heard of.  This person took the role of the Scrum Master.  *The Scrum Master*  The Scrum Master is easily one of the most important roles. This person is responsible for creating a bridge between the Product Owner (defined below) and the Team (also defined later). This person usually possesses a strong technical knowledge, and actively participates in the development process. He or she also communicates with the Product Owner about the User Stories, and how to organize the Backlog.  The Scrum Master coordinates development processes, but he does not micro-manage (the team is self-organized). At the beginning of the process, however, the Scrum Master might micro-manage part of the team, in order to improve their team interaction and self-organization techniques.  Scrum Masters have more responsibilities, and I’ll cover them as we discuss this process. | **Вот как мы начинали**  Все зависит от самоотдачи и доброй воли. Если вы хотите, чтобы ваша команда была эффективной, продуктивной и давала результаты вовремя, вам нужно, чтобы кто-то применил в некотором виде техники гибкой разработки. Скрам может быть, а может и не быть идеальным для вас, но несомненно одна из лучших, с каких можно начать. Найдите в команде кого-то, кто готов помогать другим, или же сами возьмите на себя ответственность за введение скрам.  *“Вы можете спросить – почему вас должно заботить то, как другая команда, вроде моей, применяет скрам. Вас должно это заботить, потому что мы все понимаем, как сделать скрам лучше на основании историй о том, как его применили другие – особенно те, кому это хорошо удалось. – Майк Кон”*  Талантливая команда, с которой я работаю, уже много знала о гибкой разработке. Мы переключились с водопадного на более гибкий процесс и выпускали релизы довольно часто. Мы успешно выпускали их каждые три-шесть месяцев, со значительно меньшим количеством багов.  Но все же, мы были далеки от того, чего можем достичь сегодня. Мы не могли найти процесс, или правила, которые заставили бы нас изменить наш взгляд на продукт и процесс. В это время менеджер команды познакомил нас со скрам, термином, который мы, на тот момент, никогда раньше не слышали.  Этот человек взял на себя роль скрам-мастера.  *Скрам-мастер*  Скрам-мастер – одна из самых важных ролей. Этот человек несет ответственность за создание моста между владельцем продукта (описан ниже) и командой (также описана ниже). Этот человек обычно обладает прочными техническими знаниями и активно участвует в процессе разработки. Он или она также общается с владельцем продукта, получая от него истории пользователей и обсуждая, как организовать журнал.  Скрам-мастер координирует процесс разработки, но не занимается микро-менеджментом (команда самоорганизуется). В начале процесса, тем не менее, скрам-мастер может заниматься микро-менеджментом части команды для улучшения их техник командного взаимодействия и самоорганизации.  Скрам-мастера выполняют и другие задачи, я расскажу о них при описании процесса. |
| **Introducing the Term “Sprint”**  Personally, we don’t have any problem with three to six month releases, but I originally couldn’t imagine such a frequent release schedule. I thought it was too fast, and didn’t provide us with the necessary time to integrate and debug our products. But then, our Scrum Master introduced us to the term, sprint.  *“Sprint: a basic unit of development in Scrum. It can take between one week and one month, and the product is in a stable state after each sprint.”*  That sounds outrageous! Stable every week> Impossible! But, in actuality, it’s quite possible. First, we reduced our production cycles from three months to one-and-a-half months, and, finally, to a single month. All of this was accomplished without changing our style. However, you’ll need to introduce some rules for a sprint less than thirty days. There’s no magic rule-set here; the rules must benefit your project.  If I recall correctly, the first significant change in our development process came by the introduction of sprint planning.  *Sprint Planning*  This is one of the several meetings that Scrum recommends. Before each new sprint, the team, product owner, and scrum master plan the next sprint. This meeting can take a whole day, but shorter sprints likely only need a couple hours or so.  Our process is mostly reviewing the product backlog, and deciding upon a subset of user stories that will be included in the next sprint. These decisions are made by direct negotiations between the team, represented by the scrum master, and the product owner.  *The Product Owner*  *“Setting the direction of a product by guessing which small feature will provide the most value may be the most difficult task.”*  This person is responsible for defining the User Stories and maintaining the Product Backlog. He or she is also a bridge between the team and higher management. The product owner evaluates the requests from stakeholders, higher management, users, and other feedback (like bug reports, user surveys, etc).  He or she prioritizes this backlog, providing the maximum value to the stakeholders in the shortest possible time. The product owner achieves this by planning the most valuable user stories that can be completed in a timely manner. This may sound sophisticated – it is! In fact, setting the direction of a product by guessing which small feature will provide the most value may be the most difficult task of the whole process. On the other hand, sometimes it’s rather easy. You may have a thousand users asking for a specific feature. In these cases, the correct choice is obvious.  *“If those users represent a large portion of you user base, providing that specific feature increases loyalty.”*  But again, this is a difficult choice. What if you could increase your user base by 100% by implementing a different feature? So, you can either increase your current users’ loyalty, or increase the user base. What is the correct choice? It all depends on the current direction of the business, and the product owner must decide where to take the product.  In the company I work for, these decisions propagate to the team. It’s not a requirement of the Scrum process, but it is especially useful with new teams. Sharing information goes a long way in helping everyone understand why some decisions are made, and why seemingly obvious features may be delayed or dropped. | **Знакомимся с термином «Спринт»**  Лично у нас нет никаких проблем с периодичностью выпуска от 3 до 6 месяцев, но сначала я не мог представить такой частый выпуск. Я думал, это слишком быстро, нам не будет хватать времени на интеграцию и отладку продуктов. Но затем скрам-мастер познакомил нас с термином «спринт».  *“Спринт: базовая единица разработки в скрам. Он может занимать от одной недели до одного месяца, при этом продукт находится в стабильном состоянии после каждого спринта.”*  Это звучит чересчур! Стабильный релиз каждую неделю > Невозможно! Но, на самом деле, это вполне возможно. Во-первых, мы уменьшили наши циклы разработки с трех до полутора месяцев, а затем и до одного месяца. Все это было достигнуто без изменения нашего стиля. Однако вам придется ввести некоторые правила для спринта короче тридцати дней. Не существует магического набора правил, они должны быть полезны для вашего проекта.  Если я правильно помню, первое значительное изменение в нашем процессе разработки произошло с введением планирования спринта.  *Планирование спринта*  Это одна из нескольких встреч, которая рекомендована скрам. Перед каждым новым спринтом команда, владелец продукта и скрам-мастер планируют следующий спринт. Эта встреча может занять целый день, но более короткие спринты снижают это время до пары часов.  Наш процесс – в основном обзор журнала продукта и выбор подмножества историй пользователя, которые будут включены в следующий спринт. Эти решения принимаются прямыми переговорами команды, представляемой скрам-мастером, и владельца продукта.  *Владелец продукта*  *“Установка направления развития продукта решением, какое небольшое свойство принесет наибольшую ценность, это, пожалуй, наиболее сложная задача.”*  Этот человек несет ответственность за определение историй пользователя и поддержку журнала продукта. Он или она также являются мостом между командой и высшим управлением. Владелец продукта оценивает запросы акционеров, высшего управления, пользователей, и других источников обратной связи (сообщений об ошибках, обзоров пользователей и т.д.).  Он или она расставляет приоритеты журнала, предоставляя акционерам максимальный результат за минимальное время. Владелец продукта достигает этого планированием наиболее ценных историй пользователей, которые могут быть выполнены в разумное время. Это может звучать мудрено – так и есть! На самом деле, установка направления развития продукта решением, какое небольшое свойство принесет наибольшую ценность, это, пожалуй, наиболее сложная задача во всем процессе. С другой стороны, иногда это довольно легко. У вас может быть тысяча запросов от пользователей об определенной возможности. В таких случаях верный выбор очевиден.  *“Если эти пользователи представляют большую долю вашей пользовательской аудитории, предоставление этой возможности увеличивает лояльность.”*  Но опять же, это сложный выбор. Что если вы можете увеличить вашу пользовательскую аудиторию на 100% введением новой возможности? То есть вы можете либо увеличить лояльность нынешних пользователей, либо увеличить число пользователей. Каков правильный выбор? Все зависит от текущего направления бизнеса, и владелец продукта должен решить, куда направить продукт.  В компании, где я работаю, эти решения предлагаются команде. Это не обязательно при реализации скрам, но особенно полезно для новых команд. Раскрытие такой информации может многого стоить с точки зрения помощи в понимании, почему приняты те или иные решения, почему кажущиеся очевидными функции откладываются или отменяются. |
| **The Planning Board**  I remember the morning it happened: I arrived at the office, only to find our scrum master preparing a makeshift planning board with A4 paper and transparent tape. I had no idea what he was doing. As with every morning, I prepared a pot of coffee, and waited to see.  When finished, a white board was placed on the wall. It had several columns, and morphed into a rectangular shape. Several colored sticky notes peppered the “board.” That was two years ago.  The board now accommodates the Lean Development Process that we use today. Remember, Agile is all about change and adapting to change. Never blindly follow the rules.  C:\Users\Владимир\Desktop\scrum-story-of-agile-team-PlanningBoard.jpg  So, what’s on that board?  *Columns for the Development Process*  There are four main columns:   * Release Backlog – Where all the stories reside for the current release. Yes, the product is ready for release after each sprint, but that doesn’t necessarily mean that we actually release it. We typically have five-day sprints. * Sprint Backlog – When we plan, we negotiate what the product owner wants to complete in the sprint. How do we decide what we can and cannot complete? By estimating the difficulty of each story (see below – Estimating stories). The end result is a set of stories moved from the release backlog to the sprint backlog. The team concentrates on finishing those stories in the upcoming week. * Working On – This one is simple. When team members take a story, they add it to this column, to show everyone what they are working on. This is meant for employee control, but rather for communicating with team members. Everyone should always know what their teammates are working on. In the above image, the small bookmarks stuck on the sticky notes contain my team members’ names. * Done – Complete all the things! Yes, this is where completed stories go. However, it’s important to define “what is done.” At the end of an ideal sprint, all stories and bugs from the sprint backlog should be contained within this column.   **Tip:** Many teams like to split the Working On column into several others to better define different stages of work. We split ours into Design, Development and Testing. Feel free to invent your own steps, according to your process.  *The Definition of Done*  What is done? When can you confidently state that a story is complete? How do you know?  “Done” must be clearly and precisely defined, so that everyone knows when a story is complete. The definition of “done” can differ from team to team, and even from project to project. There is no exact rule. I recommend that you raise this issue at a team meeting, and decide what determines a complete story. Here are some ideas that you might want to consider:   * Create a minimalistic design. * Create a GUI mockup. * Use TDD or ensure that there are unit tests. * Write the code. * Let another team member manually test your story. * The whole system can be built and compiled with the new code. * Functional or acceptance tests pass as expected, after the new code is integrated into the system.   There are multiple ideas that can be included in the definition of done. Take what you consider to be necessary for your team and use it. Also, don’t forget to adapt this definition over time. If you notice that your definition is becoming outdated, you may consider removing some elements or adding necessary, but frequently forgotten, ideas.  In the picture above, the green sticky notes describe what we considered to be done, for each part of the process. | **Доска для планирования**  Я помню утро, когда это произошло: я пришел в офис и увидел, что скрам-мастер подготавливает временную доску для планирования с помощью листов A4 и прозрачного скотча. Я не понимал, что он делает. Как и каждое утро, я приготовил кофе и ждал, что будет.  Когда он закончил, белая доска была помещена на стену. В ней было несколько колонок, сама она была прямоугольной формы. «Доску» усеивали цветные липкие записки. Это было два года назад.  Сейчас доска приспособлена под бережливый процесс разработки, который мы используем сегодня. Помните, гибкая разработка заключается в изменениях и приспособлении к ним. Никогда не следуйте правилам вслепую.  C:\Users\Владимир\Desktop\scrum-story-of-agile-team-PlanningBoard.jpg  Итак, что на этой доске?  *Колонки процесса разработки*  Существуют четыре главных колонки:   * Журнал выпуска – В нем находятся все истории для текущего релиза. Да, продукт готов к выпуску после каждого спринта, но это не обязательно означает, что мы правда его выпускаем. Обычно мы используем пятидневные спринты. * Журнал спринта – Когда мы планируем, мы обсуждаем, какой продукт владелец хочет получить по результатам спринта. Как мы решаем, что мы можем и что не можем закончить? Оценивая сложность набора историй, перемещенных из журнала релиза в журнал спринта. Команда концентрируется на завершении этих историй в ближайшую неделю. * В работе – Это просто. Когда члены команды берут историю, они добавляют ее в эту колонку, чтобы показать всем, над чем они работают. Это нужно для контроля подчиненных, но еще больше – для общения с членами команды. Каждый должен всегда знать, над чем работают члены его команды. На изображении выше небольшие наклейки на записках содержат имена членов моей команды. * Сделано – Завершите все штуки! Да, сюда перемещаются завершенные истории. Однако важно определить, что значит «сделано». В конце идеального спринта все истории и баги из журнала спринта должны содержаться в этой колонке.   **Совет:** Многие команды предпочитают разделять колонку «В работе» на несколько других, чтобы лучше определить различные этапы работы. Мы разделяем нашу на «Дизайн», «Разработка» и «Тестирование». Вы можете придумывать собственные шаги в зависимости от вашего процесса.  *Определение термина «Сделано»*  Что значит «сделано»? Когда вы можете с уверенностью сказать, что история завершена? Как об этом узнать?  “Сделано” должно быть определено ясно и точно, чтобы все знали, когда история считается завершенной. Определение «сделано» может отличаться в разных командах, даже в разных проектах. Нет точного правила. Я рекомендую поднять этот вопрос на встрече команды и решить, что определяет завершенность истории. Вот некоторые идеи, которые вы можете захотеть учесть:   * Создать минималистичный дизайн. * Создать макет интерфейса пользователя. * Использовать разработку через тестирование, чтобы убедиться в существовании юнит-тестов. * Написать код. * Дать другому члену команды вручную протестировать вашу историю. * Вся система должна быть построена и скомпилирована с новым кодом. * Функциональные тесты или тесты приемки проходят как запланировано после того, как новый код интегрирован в систему.   Есть множество идей, которые можно включить в определение «сделано». Возьмите то, что считаете необходимым для вашей команды и используйте. Также не забывайте адаптировать это определение во времени. Если вы замечаете, что ваше определение устарело, можете попробовать удалять или добавлять элементы.  В изображении выше зеленые липкие записки описывают, что мы желаем сделать на каждом этапе. |
| **Populating the Board**  That was the question we asked ourselves. Until this point, we did not use sticky notes for planning. We used software to keep track of user stories and bugs, but, other than that, we used nothing. After lunch, our scrum master presented us with a mountain of colored sticky notes. After preparing a dozen notes, he explained them to us.  *The User Stories*  *“A user story is a short, one sentence definition of a feature or functionality.”*  These represent the main features that we want to implement. A user story is a short, one sentence definition of a feature or functionality. It is referred to as a user story, because it is presented from the perspective of a user. Naturally, the term *user* is the person using our application. This person can be in one or more different categories: a system administrator, restricted user, manager, etc.  An example of such a story might sound something like, “*As a user, I want to share a folder with my teammates.*“  At that point, we did not have a product owner defined, so our scrum master invented these stories. This is acceptable at the beginning, but I **highly recommended** that you separate these two roles. Otherwise, how can the scrum master negotiate the sprint backlog with the product owner?  You may think to yourself, “*Why negotiate? Isn’t it actually better for a single person to decide what to do and when?*” Not quite. The two roles would be influenced by a single person’s views of the system or project. Two people, on the other hand, have a better chance of providing a more objective path for the team, ensuring the end goal (better valuable software) is achieved.  *“The product owner should define user stories; the team should negotiate their execution, and the scrum master represents them.”*  User stories define everything new that needs to be done; they’re represented by yellow sticky notes on our board.  *Bugs, Bugs, Bugs*  *“Tracking your bugs is incredibly important.”*  We also list bugs on the board. Do you see those red sticky notes? Those are the bugs that we need to fix for the next release.  Different teams treat bugs in different ways. Our team mixes the bugs with the stories, but we always begin a sprint by fixing the bugs.  I know of other teams who pile up bugs for a period of three sprints, and spend every fourth sprint fixing them. Others split sprints into 25/75 for bugs/stories. Further, other teams may work on stories in the morning, and bugs after lunch; it simply depends on the company.  It’s up to you to find the best solution for your team, and of course, keep track of your bugs. Write them on sticky notes so that you can track your system’s issues and the fixes for those issues. Tracking your bugs is incredibly important.  *Tasks or Sub-Stories*  *“Tasks are written as simple sentences from the developer’s point of view.”*  Ideally, each story should be short enough to be completed with relative ease, but splitting stories into other stories can prove difficult. Some projects simply don’t afford this possibility. Still, you’ll find large stories that several team members can work on. It’s important to divide huge chunks of work into smaller, easier to manage pieces.  One approach splits big stories into Tasks. Tasks are written as simple sentences from the developer’s point of view. For example, the previous folder sharing story might be divided into several tasks, such as: “Create UI for sharing”, “Implement public sharing mechanism”, “Implement access control functionality”, “Add Access Control checkboxes to the UI”, and so on. The point is that you have to think more about the story every time you break it into smaller tasks. Your team will have much greater control over a story, when you analyze each piece.  We use light-blue sticky notes for tasks on our board. Look in the last column (the “Done” column), and you’ll find our tasks under the user story stickies. That particular story was broken into around four tasks.  C:\Users\Владимир\Desktop\scrum-story-of-agile-team-BurnDown2.png | **Наполнение доски**  Вот каким вопросом мы задались. До этого момента мы не использовали липкие записки для планирования. Мы использовали ПО для отслеживания наших пользовательских историй и багов, но кроме этого мы не использовали ничего. После ланча наш скрам-мастер представил нам гору липких записок. После приготовления дюжины он объяснил их нам.  *Пользовательские истории*  *“Пользовательская история это короткое, на одно предложение, определение возможности или функциональности.”*  Они представляют главные возможности, которые мы хотим реализовать. Пользовательская история это короткое, на одно предложение, определение возможности или функциональности. Его называют пользовательской историей, потому что оно представляется с перспективы пользователя. Естественно, пользователь – это тот, кто использует наше приложение. Этот человек может быть в одной или нескольких категориях: системный администратор, ограниченный пользователь, менеджер и т.д.  Пример такой истории может звучать так: «Как пользователь, я хочу разделять папку с членами своей команды».  В то время у нас не было владельца продукта, так что наш скрам-мастер придумал эти истории. В начале такое допускается, но я **настоятельно рекомендую** разделить эти две роли. Иначе как может скрам-мастер обсуждать журнал спринта с владельцем продукта?  Вы можете задаться вопросом: “*Зачем обсуждать? Не лучше ли дать одному человеку решать, что и когда делать?*” Не совсем. Две роли будут под влиянием взглядов одного человека на систему или проект. У двух людей, с другой стороны, больше шансов прийти к более объективному решению для команды, позволяющему добиться цели (более ценного ПО).  *“Владелец продукта должен определить пользовательские истории; команда должна обсудить их исполнение, их представляет скрам-мастер.”*  Пользовательские истории определяют все новое, что нужно сделать; на нашей доске они представлены желтыми липкими записками.  *Баги, баги, баги*  *“Отслеживание багов неописуемо важно.”*  Мы также заносим баги на доску. Видите красные липкие записки? Это баги, которые нам нужно устранить до следующего релиза.  Различные баги по-разному обращаются с багами. Наша команда смешивает баги и истории, но мы всегда начинаем спринт с исправления багов.  Я знаю другие команды ,которые оставляют баги на период в три спринта и тратят каждый четвертый на их исправление. Другие разделяют спринт 25/75 для багов/истории. К тому же, некоторые команды могут работать над историями утром, а над багами – после ланча; все зависит от компании.  Вы должны найти лучшее решение для вашей команды и, конечно, отслеживать баги. Напишите их на липких записках, чтобы иметь возможность отслеживать проблемы в вашей системе и их исправления. Отслеживание багов неописуемо важно.  *Задачи или под-истории*  *“Задачи описываются как простые предложения, написанные с точки зрения разработчика.”*  В идеальном случае каждая история должна быть достаточно короткой, чтобы быть выполненной довольно просто, но разделение историй на другие может оказаться сложным. Некоторые проекты просто не допускают этого. И все же вы найдете большие истории, над которыми могут работать несколько человек. Важно разделять большие куски работы на более мелкие, с которыми легче работать.  Один из подходов – разделение больших историй на задачи. Задачи описываются как простые предложения, написанные с точки зрения разработчика. Например, предыдущая историй о разделяемой папке может быть разделена на несколько задач так: «Создать интерфейс пользователя для разделения», «Реализовать механизм публичного разделения», «Реализовать функцию контроля доступа», «Добавить чекбоксы контроля доступа к пользовательскому интерфейсу». Суть в том, чтобы больше думать об истории при ее разбиении. Ваша команда получит намного больше контроля над историей, если вы проанализируете каждую часть.  Мы используем светло-синие липкие записки для задач на нашей доске. Посмотрите на последнюю колонку («Сделано»), и вы найдете наши задачи под записками пользовательских историй. Эта конкретная история была разбита на четыре задачи.  C:\Users\Владимир\Desktop\scrum-story-of-agile-team-BurnDown2.png |
| **Technical Tasks**  Certain activities must be completed in order to finish a task, story, or the sprint as a whole. These tasks are usually infrastructure related, but you may find that the tasks require changes to the system. This process may or may not be part of a story. For example, you may find that a third party application must be installed, in order to implement your application’s sharing capability. Is this part of our user story? Possibly, but maybe not.  We determined that these tasks should be separated from the actual story. This helped us to better track these extra tasks. On our board, you can find these tasks on green sticky notes. There is one on the sprint backlog, and about three in testing.  *The Technical Backlog*  *“For a young team with little experience with Agile and Scrum, it’s helpful to highlight these tasks with a mini-backlog.”*  This backlog is for infrastructural tasks, such as updating the automated testing system, configuring a new server, and other things, which make our everyday development work easier. These are things that must be completed at some point, but are not directly related to development.  You don’t have to put these items into a separate backlog. In fact, I know of teams who don’t separate them. Our team dropped our technical backlog a few months ago; we decided that infrastructural tasks are as important as other tasks. However, for a young team with little experience in Agile and Scrum, it’s helpful to highlight these tasks with a mini-backlog. So, I recommend that you should give it a try. It may work for you, and, if not, then just put Infrastructural Tasks on you planning board, possibly with a different color. | **Технические задачи**  Для завершения задачи, истории или спринта в целом нужно выполнить определенные мероприятия. Эти задачи обычно касаются инфраструктуры, но вы можете обнаружить, что они требуют изменений в системе. Этот процесс может быть, а может не быть частью истории. Например, вы можете обнаружить, что необходимо установить стороннее приложение для реализации разделения папок в вашем приложении. Является ли это частью пользовательской истории? Возможно, но может быть, и нет.  Мы определили, что эти задачи должны быть отделены от самой истории. Это помогло нам лучше отслеживать дополнительные задачи. На нашей доске вы можете найти эти задачи на зеленых липких записках. Одна находится в журнале спринта, еще около трех в тестировании.  *Технический журнал*  *“Для молодой команды с небольшим опытом гибкой разработки и скрам будет полезно выделение этих задач в мини-журнал.”*  Этот журнал предназначен для инфраструктурных задач, таких как обновление системы автоматизированного тестирования, конфигурация нового сервера и другие задачи, которые делают ежедневную разработку проще. Это вещи, которые нужно сделать, но которые не связаны с разработкой напрямую.  Вы не обязаны помещать эти задачи в отдельный журнал. Фактически, я знаю команды, которые их не разделяют. Мы отменили такое разделение несколько месяцев назад, решив, что инфраструктурные задачи так же важны, как и другие. Тем не менее, для молодой команды с небольшим опытом гибкой разработки и скрам будет полезно выделение этих задач в мини-журнал. Так что я рекомендую вам попробовать. Это может сработать для вас, если же нет – просто поместите их на вашу доску для планирования, возможно, с другим цветом. |
| **The Big Challenge: Estimation**  During a planning meeting, we decide which user stories and bugs from the product backlog (or release backlog in our case) to include in the next sprint. This may sound simple, but, in reality, it’s quite complicated.  The product owner comes forward with a set of stories to work on. This list typically contains more work than what can be accomplished in the sprint. The scrum master, together with the team, has to convince the product owner of what can be done during a sprint. Over time, this process becomes easier, as the product owner learns the approximate velocity of the team. Then again, the team may become more productive with each sprint, thus allowing more stories to be finished. The trick is to have a team who really wants to exceed expectations!  Now, the product owner wants to complete more stories than we can do in a sprint. We need to estimate the amount of work we can do in relation to the stories submitted by the product owner, and we can do this in a variety of ways.  *Story Points*  Story Points are one of the most common methods for estimating stories, bugs or tasks. They are not necessarily the best approach, but they still are a good way to start.  So what is a story point? At the beginning of the process, the team looks for the simplest story they can find on the board. It doesn’t matter how difficult it is, or how long it takes to complete. When they find that story, they set it as the reference story of having one point. In some projects, such a story can be as simple as fixing UI elements in ten minutes; whereas, the simplest story for more complex systems may take two hours for three people to complete. Now that you have a baseline, evaluate the other stories and bugs and assign them points.  This can be more difficult than it seems, and there are several point techniques to better help estimate stories. Here are two of them:   * Use Fobinacci Numbers – 1,2,3,5,8 (and maybe 13, but a 13-sized story smells too big to me). * Use Powers of 2 – 1,2,4,8 (and maybe 16, but this number should be avoided).   You are free to choose whatever you feel most comfortable with. Be agile! Maybe you want to use two points increments, because your tasks are best estimated that way. Bravo to you!  *Semaphores*  Numbers are great, and many technical people love them. You may find, however, that, at some point, programmers begin to associate story points with time. They will think, “It takes me two days to do this. Let’s give it five points”. This is wrong. Estimates go from bad to worse when this happens. Story points should never relate to time.  *“Using semaphore colors may help alleviate this problem. Instead of assigning numbers to stories, the team can associate those stories with colors. Our team made this change a few months ago, and it greatly helped change our point of view.”*  Naturally, each color has a different meaning.   * Green signifies an easy task that can be completed in the next sprint. * Yellow refers to a more difficult task – one that requires more effort from several team members. It also has a high chance of completion in the next sprint. * Red labels are assigned to stories, which are extremely difficult and may not be finished in a single sprint. There should be little to no such stories, but if you adopt one week sprints, five days is a short time.   *T-Shirt Sizes*  Numbers may be ugly to you, colors too artistic. It’s time for t-shirt sizes! This technique suggests giving up on comparing story complexity with time of completion. Simply put, instead of numbers, you use sizes like S, M, L, XL, XXL for your stories.  I personally never felt attracted to this kind of estimation, but, hey, some feel that it’s the best way to go. Try it out, if you feel comfortable with the idea.  The product owner, stake holders, and managers have to know what to expect from the end of a sprint. They must decide if the stories that were worked on should be released, and they have to know when features are ready. It’s not a good solution to release every new feature at the end of a product’s development cycle; releasing the most valuable features on a more frequent basis is a considerably better way to go. To achieve this, they must know what will be available in the short term, and their information should stem from the team. Therefore, the team should estimate, as best as possible, the work they can do in a sprint. | **Большой вызов: оценка**  Во время встреч по планированию мы решаем, какие пользовательские истории и баги из журнала продукта (или журнала релиза, в нашем случае) включить в следующий спринт. Это может звучать просто, но в реальности это довольно сложно.  Владелец продукта дает набор историй для работы. Этот список обычно содержит больше работы, чем можно выполнить за спринт. Скрам-мастер вместе с командой должен убедить владельца продукта, что они смогут сделать за спринт. Через некоторое время этот процесс становится проще, так как владелец продукта осоздает приблизительную скорость команды. К тому же команда может становиться более продуктивной с каждым спринтом, позволяя выполнять больше историй. Фокус в том, чтобы найти команду, которая хочет превосходить ожидания!  Владелец продукта хочет завершить больше историй, чем мы можем сделать за спринт. Мы должны оценить объем работы, который мы можем выполнить по отношению к историям, выданным владельцем продукта, и мы можем сделать это несколькими способами.  *Очки истории*  Очки истории – один из самых распространенных методов оценки историй, багов или задач. Они не обязательно лучше остальных, но с них хорошо начинать.  Так что же такое очко истории? В начале процесса команда выбирает наиболее простую историю на доске. Неважно, насколько оно сложное или сколько времени требует для выполнения. Найдя эту историю, они делают ее основой, придавая единичное значение. В некоторых проектах такая история может быть просто десятиминутным исправлением элементов интерфейса, тогда как для более сложных систем может занять два часа у команды из трех человек. Теперь, когда у вас есть мера, оцените другие истории и баги и назначьте им очки.  Это может быть сложнее, чем кажется, существует несколько техник начисления очков для лучшей оценки историй. Вот две из них:   * Использовать числа Фибоначчи – 1,2,3,5,8 (и может быть, 13, но 13-очковая история кажется мне великоватой). * Использовать степени двойки – 1,2,4,8 (и может быть, 16, но этого числа лучше избегать).   Вы можете выбирать любой способ, какой вам удобнее. Будьте гибкими! Может быть вы захотите использовать прибавление двух, потому что ваши задачи лучше оценивать таким образом. Ну и молодцы!  *Семафоры*  Числа это здорово, и многие технари их любят. Но вы можете обнаружить, что на некотором этапе программисты начинают ассоциировать очки истории с временем. Они думают: «Это займет два дня. Давайте дадим ему пять очков». Это неправильно. Оценки только все портят, когда происходит такое. Очки истории не должны соотноситься со временем.  *“Использование семафорных цветов может помочь облегчить эту проблему. Вместо назначения чисел историям, можно ассоциировать эти истории с цветами. Наша команда перешла к такому способу несколько месяцев назад, и это очень помогло изменить нашу точку зрения.”*  Естественно, каждый цвет имеет свое значение:   * Зеленый обозначает простую задачу, которую можно выполнить в следующем спринте. * Желтый обозначает более сложное задание – то, что потребует больших усилий от членов команды. У нее также высокая вероятность выполнения в следующем спринте. * Красные метки назначаются историям, которые крайне сложны и могут быть не выполнены за один спринт. Таких историй должно быть мало или не быть вообще, но если вы используете недельные спринты, пять дней это мало.   *Размеры футболок*  Номера могут вам не нравиться, а цвета казаться слишком изобразительными. Пришло время размеров футболок! Эта техника предполагает отказ от сравнения сложности истории со временем ее завершения. Проще говоря, вместо чисел вы используете для ваших историй размеры вроде S, M, L, XL, XXL.  Лично меня никогда не привлекал такой вид оценки, но, что говорить, некоторым он нравится больше всего. Попробуйте, если вам это кажется удобным.  Владелец продукта, акционеры и менеджеры должны знать, что ожидать к концу спринта. Они должны решить, нужно ли выпускать истории, над которыми велась работа, и они должны знать, когда будут реализованы возможности. Выпуск каждой новой возможности в конце цикла разработки продукта – плохая идея, выпуск наиболее ценных чаще – намного лучше. Чтобы этого достичь, они должны знать, что скоро будет доступно, и эта информация должна идти из команды. Поэтому команда должна оценивать работу, которую она в состоянии сделать за спринт, как можно лучше. |
| **Measuring Speed of Development**  So you want to see how well you perform in the current sprint? A frequently-used method is the burndown chart:  C:\Users\Владимир\Desktop\scrum-story-of-agile-team-BurnDown1.png  In this chart, we have a five day sprint, and we assumed that we could complete ten points in the sprint. Each value point represents the remaining story points at the end of each day. The green line reveals the ideal path: a steady two points per day. The red line shows our actual performance, or the true speed of development.  It’s not on the planning board picture, but my team used to have an A4 paper sheet positioned above the planning board, with the burndown chart for each sprint. At the conclusion of each day, one of the team members was responsible for calculating the points completed for that day. It’s simple: if programmers move the stories from column to column as they work, finding the remaining unfinished stories is easy.  *“There are no half-done stories. If a story is done, it is done. If it is not complete, then it is not counted on the burndown chart.”*  Of course, you will fail – BIG TIME – at estimating! This is especially true at the beginning. There is no way to avoid this, unfortunately. There is no way to know how many points you can to deliver. Your first chart may very well look like:  C:\Users\Владимир\Desktop\scrum-story-of-agile-team-BurnDown2.png  Our first chart surely looked similar to that. I think we did not even complete half of what we committed to. Why? Well, because estimation is hard. No matter what you do, or how good are you, when someone asks you how complicated something you have never done before is, you will have a hard time providing an accurate estimation. Don’t panic! Try your best. With time, these things become easier. You may become able to estimate with a 70% accuracy at some point for short sprints. If the sprints are longer, your accuracy will likely be less, because there will be more stories to estimate and more variables that can go wrong.  When this happens, you adjust. For the next sprint, take four points. Like this:  C:\Users\Владимир\Desktop\scrum-story-of-agile-team-BurnDown3.png  This is bad again. You were too conservative, and finished early. It’s a natural reaction for the team to adjust, based on the failure of the previous estimation. Still, this is a failure again, but on the other side of the road.  The problem? What do you do after you’ve finished what you were planning for? Another story? How do you put that on the chart? You can’t redraw the original line.  *“When working with burndown charts, it’s recommended to always make an average of the last 3-5 charts, in order to specify the points for the upcoming sprint. Of course, at the beginning, you don’t have such information available, so you won’t be as accurate as in the future. That’s okay.”*  After some time, your charts will begin to resemble the first example more and more. You will, most of the time, finish all the stories and have a sustained velocity. | **Измерение скорости разработки**  Итак, вы хотите видеть, как хорошо вы справляетесь в текущем спринте? Часто для этого используется график «выгорания»:  C:\Users\Владимир\Desktop\scrum-story-of-agile-team-BurnDown1.png  На этом графике показан пятидневный спринт, и мы предположили, что мы можем выполнить десять очков в спринте. Каждое очко представляет оставшиеся очки истории в конце каждого дня. Зеленая линия показывает идеальный путь: стабильные два очка за день. Красная линия показывает нашу действительную производительности, или действительную скорость разработки.  На картинке с доской для планирования этого не показано, но моя команда использовала лист бумаги A4, размещенный над доской для изображения графика выгорания для каждого спринта. В конце каждого дня один из членов команды должен был подсчитать очки, выполненные за день. Это просто: если программисты передвигают историю из колонки в колонку во время работы, нахождение оставшихся историй не составляет труда.  *“Нет полу-сделанных историй. Если история сделана, она сделана. Если она не завершена, она не учитывается в графике выгорания.”*  Конечно, вы ошибетесь – ЕЩЕ КАК – при подсчете! Это особенно верно для начинающих. Этого не избежать, к сожалению. Нельзя узнать, сколько очков вы сможете сделать. Ваш первый график может выглядеть так:  C:\Users\Владимир\Desktop\scrum-story-of-agile-team-BurnDown2.png  Наш первый график точно был похож на этот. Я думаю, мы даже не выполнили половины из того, на что решились. Почему? Ну, потому что оценка сложна. Неважно, что вы делаете, или как вы хороши, когда кто-то просит вас, насколько сложно то, чего вы никогда раньше не делали, вам будет сложно дать точную оценку. Не паникуйте! Делайте все, что в ваших силах. Со временем это будет удаваться легче. На некотором этапе вы можете стать способны оценивать с 70% точностью для коротких спринтов. Если спринты длиннее, ваша точность скорее всего будет ниже, потому что будет больше историй для оценки и больше переменных, которые могут измениться не так, как ожидалось.  Когда это происходит, вы приспосабливаетесь. Для следующего спринта возьмите четыре очка. Вот так:  C:\Users\Владимир\Desktop\scrum-story-of-agile-team-BurnDown3.png  Снова плохо. Вы были слишком консервативны и закончили раньше. Приспосабливаться, основываясь на ошибке предыдущей оценки, естественно. Но это тоже ошибка, только с другой стороны.  Проблема? Что вы делаете после того, как закончите что планировали? Другую историю? Как вы поместите ее на график? Вы не можете перерисовать оригинальную линию.  *“При работе с графиками выгорания рекомендуется всегда брать среднее 3-5 последних графиков при определении очков грядущего спринта. Конечно, в начале у вас нет такой информации, так что вы не будете столь же точны, сколь в будущем. Это норма.”*  После некоторого времени ваши графики начнут напоминать первый пример больше и больше. Большую часть времени вы будете заканчивать все истории с установившейся быстротой. |
| **Velocity?**  This term refers to your speed of development. It relates to how much you can do in a sprint. One of the most important concepts in Agile is to have a consistent velocity. Make the team deliver at a constant pace. With traditional project management, velocity decreases as a project ages. Complexity and rigidity forces the speed of development down.  Agile methodologies and techniques target to maintain a constant pace. Deliver fast now, and deliver faster later. How do they do it? Scrum is one of the elements of the puzzle. Other important pieces include the techniques that programmers can use to make their code and development process better. For example, XP (eXtreme Programming), Pair Programming, TDD (Test Driven Development), etc. All these, together, can make a team really great.  So we measure this velocity, but what do we actually do with it?  **Tip:** Measuring velocity is for making better predictions; not for judging a team or its members.  Use velocity to know what your team can do. Use velocity to know what to expect. Use velocity to make your team want more and be better. **Never** use velocity to judge your team or evaluate the productivity of your programmers. | **Быстротой?**  Данный термин относится к вашей скорости разработки. Он показывает, как много вы можете сделать за спринт. Одна из главных идей гибкой разработки – иметь постоянную быстроту. Сделайте так, чтобы команда давала результаты в постоянном темпе. При традиционном управлении проектом быстрота со временем снижается. Сложность и жесткость снижают скорость разработки.  Гибкие методологии и техники нацелены на поддержку постоянного темпа. Делайте быстро сейчас, и быстрее позже. Как этого добиться? Скрам – один из элементов головоломки. Другие важные детали включают техники, которые программисты используют, чтобы сделать их код и процесс разработки быстрее. Например, XP (экстремальное программирование), парное программирование, TDD (разработка через тестирование), и т.д. Все это вместе может сделать команду по-настоящему хорошей.  Итак, мы измеряем эту быстроту, но что нам с ней делать?  **Совет:** Измерение быстроты позволяет делать более точные предположения, оно не для того, чтобы судить команду или ее членов.  Используйте быстроту, чтобы узнать, на что способна ваша команда. Используйте быстроту, чтобы узнать, чего ожидать. Используйте быстроту, чтобы ваша команда хотела большего и была лучше. **Никогда** не используйте быстроту для осуждения вашей команды или оценки продуктивности ваших программистов. |
| **Always Look Back and Improve**  Following the first few sprints, our scrum master gathered the whole team. He began asking us about the good and bad things from the past week. This might be uncomfortable at the beginning, but it’s still incredibly important. Describing what you felt went wrong in the past week will create awareness. And, of course, it is also helpful to highlight what went well!  These meetings are typically referred to as *Retrospectives*. They offer the scope to highlight what was good and what went wrong. Here are some examples from my own retrospectives.  *Bad Things*   * Team members were fighting too much * Team member X or Y was not collaborative when pair programming * We lost too much time with small things, like X or Y * We did not pair program all the time * We did not write unit tests for module M   When discussing problems, try to put aside your personal feelings; speak about what’s bothering you. This is the only way for the team to resolve the issue. The most important thing is to immediately propose a solution for each problem. Don’t simply let the list be written and forgotten; stay for a few minutes, and have the whole team think about what can be done to avoid them next week.  *Good Things*   * We finished on time * We were able to talk without fighting * Some of us became more receptive to suggestions and ideas * We wrote all the code with TDD   Again, highlight as many good things as possible – especially at the beginning or with junior programmers. For example, having all the code written with TDD may be a big achievement for a junior team. Make them feel really good about it, make them want to do it more and more. The same is true for a senior team; they simply have other things to highlight (TDD is done by reflex). | **Всегда оглядывайтесь назад и улучшайте**  После первых нескольких спринтов наш скрам-мастер собрал всю команду. Он начал спрашивать нас о хорошем и плохом на прошлой неделе. Это может быть неудобно в начале, но это все же неописуемо важно. Описание того, что вам показалось неправильным на прошлой неделе создает осведомленность. И, конечно, полезно отметить то, что прошло хорошо!  Эти встречи обычно называются ретроспективы. Они предлагают возможность оценить, что было хорошо и что пошло плохо. Вот некоторые примеры из моих собственных ретроспектив.  *Плохое*   * Члены команды слишком много ссорились * Член команды А или Б не сотрудничал во время парного программирования * Мы потеряли слишком много время на мелочи вроде А или Б * Мы не программировали в парах все время * Мы не написали юнит-тесты для модуля М   При обсуждении проблем попробуйте отстраниться от своих личных чувств; говорите о том, что вас беспокоит. Это единственный способ для команды решить проблему. Главное – немедленно предлагать решение для каждой проблемы. Не позволяйте написанному забыться; пусть команда несколько минут подумает над тем, что можно сделать для того, чтобы избежать этого на следующей неделе.  *Хорошее*   * Мы закончили вовремя * Мы говорили, не ссорясь * Некоторые из нас стали более восприимчивы к предложениям и идеям * Мы написали весь код с применением TDD   Опять же, отмечайте так много хорошего, как можете – особенно в начале или с младшими программистами. Например, написание всего кода с использованием TDD может быть большим достижением для начинающей команды. Позвольте им хорошо чувствовать себя по этому поводу, сделайте так, чтобы они хотели делать больше. То же работает и для опытной команды; им просто нужно отмечать другие вещи (TDD применяется автоматически). |
| **I Want to See What You Did This Sprint**  *“The demo is for showing stake holders (and the product owner) the progress of the project.”*  This heading comes from the words of my scrum master. At that point, he also was the product owner. Before the end of a sprint, he’d ask us to present him with what we accomplished. We prepared a Demo, or a working example in a controlled environment.  Scrum proposes these demos at the end of each sprint. These should be done before the retrospective meeting that we discussed above. The team should prepare a special environment, and ensure that the product is capable of showcasing the features done in this sprint. The demo is for showing stake holders (and the product owner) the progress of the project.  You may ask yourself why I mentioned a controlled environment, when our product should be production-ready at the end of each sprint. Yes, the product should be as close to production-ready as possible, but that does not mean that the feature, itself, is ready. Often, there will be features which are too big to fit within a single sprint. The product will remain stable, but the feature will not quite be ready. When stakeholder see the demo, they want to review the feature and what it can do. In many cases, to showcase some functionalities for unfinished features, special environments must first be prepared.  Additionally, based on these demos, the product owner may determine that a bigger feature is good enough, and a new version of the product should be published and sent to the users. Most of the time, even if a feature is not quite complete, a release will help the project gain valuable user feedback, and concentrate the completion of the feature in a way that will satisfy as many users as possible.  Well, this sounds quite simple. You’re an agile team, you keep your tests always on green, and your product is in a stable state. How difficult can it be to prepare a quick demo? It’s more difficult than you might think!  Our team needed, if I remember correctly, more than five attempts before we managed to correctly prepare the demo. Luckily for us, the stake holders were not included in these first failed demos! | **Я хочу видеть, что вы сделали в этот спринт**  *“Демо нужно для того, чтобы показать акционерам (и владельцу продукта) прогресс проекта.”*  Этот заголовок основан на словах моего скрам-мастера. В то время он был также и владельцем продукта. Перед концом спринта он просил нас показать, чего мы достигли. Мы подготавливали демо, или работающий пример в контролируемой среде.  Скрам предлагает делать такие демо в конце каждого спринта. Они должны быть сделаны перед ретроспективой, которую мы обсуждали ранее. Команде следует приготовить специальную среду и убедиться, что продукт способен показать возможности, реализованные в этом спринте. Демо нужно для того, чтобы показать акционерам (и владельцу продукта) прогресс проекта.  Вы можете задаться вопросом, почему я упомянул контролируемую среду, если продукт должен быть готов к производству в конце каждого спринта. Да, продукт должен быть максимально близок к состоянию производства, но это не значит, что сама возможность готова. Часто есть возможности, которые не умещаются в один спринт. Продукт останется стабильным, но возможность не будет готова в полной мере. Когда акционер видит демо, он хочет увидеть возможности в действии. Во многих случаях для демонстрации функционала незаконченных возможностей необходимо сначала подготовить специальную среду.  В дополнение, на основании этих демо владелец продукта может определить, что бо́льшая возможность достаточно хороша и новую версию продукта следует издать и отослать пользователям. Большую часть времени, даже если возможность еще не готова, релиз поможет продукту получить ценные отзывы пользователей и сконцентрировать завершение возможности так, чтобы удовлетворить как можно большее количество пользователей.  Что ж, звучит довольно просто. Вы гибкая команда, вы проходите тесты, ваш продукт в стабильном состоянии. Как сложна может быть подготовка демо? Сложнее, чем вы можете думать!  Нашей команде потребовалось, если я правильно помню, более пяти попыток для правильной подготовки демо. К счастью для нас, акционеры не участвовали в этих первых неудачных презентациях! |
| **Still, We Need More Guidance**  *“In these meetings, each team member must answer three questions.”*  It was the moment when our scrum master proposed to make a meeting each day. Yes! Every day, every morning, at an exact hour!  I find this to be a very good thing for new teams – for people who are not yet comfortable with one another, or with the project. These daily meetings, called Daily Scrum, are kept with the team, at a specified time every day. This should be kept before any work is done for the respective day. In my team, we set the time to 10AM each morning, which was difficult to do. Nonetheless, it was the correct decision.  The daily scrum is a short and simple meeting (not more than fifteen minutes). The scope of it is to help team members see who is doing what, and determine where the problems and bottlenecks in the development project are.  **Tip:** Because we want to ensure that these meetings remain short, we stand up. People usually get tired after 15 minutes of standing, which is perfect! If you find coworkers searching for places to sit and rest, your meeting has likely gone on too long.  In these meetings, each team member must answer three questions:   * What did you do yesterday? – A short answer is expected – max 2-3 sentences. * What are you planning to do today? – Same type of short answer, things like “I will work on this story today.” * Are there any problems with your process? If yes, what? Can they be quickly solved? This should be an answer highlighting the problems and solutions, if known. No detailed discussions should be taken, while this meeting goes on. The scrum master should take note of the problem, and work toward solving it together with the team, after the meeting is adjourned.   Solving the problems and impediments in the way of developers should be high priority for the team, so that they can continue with their development as soon as possible. Often, the person who had the problem is capable of solving it in a timely manner. Other times, he or she requires the help of a teammate. And other times, the problem can be so serious that the team will have to stop development and concentrate exclusively on solving the one thing that prevents them from continuing their work.  I remember my team encountering these huge road-blocks on several different occasions. There were tasks and stories, which seemed to be quite obvious at first site, but, after a pair or a single programmer had the chance to dig into the problem, the obvious became confusing and wrong. We discovered several times that a third party library could not provide us with the necessary functionality, and ended up concentrating all of our efforts into finding another, more capable library – or even implementing a solution ourselves.  The majority of our project is written in PHP. At some point, we had to interface our project with VMWare. We reviewed the official libraries for VMWare API, and found out there are Java and Perl versions. There’s also an unofficial Ruby option. We were sure that we could use one of them, and simply do some exec() calls from PHP to capture the output as a string. As we thought, parsing from there should be piece of cake.  It turned out that this was next to impossible. Neither API library worked quite as we expected it to. Some were abandoned or incomplete, and they had nearly impossible to parse outputs. Ultimately, we were forced to do something that nobody had ever done before: implement a VMWare API library in PHP. Unfortunately, there was no other reasonably acceptable way to do it.  This problem was massive; it set back the initial plans by weeks! Of course, our product owner was immediately notified, and, together with him, we planned a new schedule, and developed new stories, which included the creation of this API library.  More often than not, your problems will be much smaller. People might get stuck with some more sophisticated logic, but, many times, by the following morning, they already have ideas and solutions. Other times, they will simply be going on the wrong road, and a teammate will need to help get them back on track. These represent your typical issues. | **И все же, нам нужна поддержка**  *“На этих встречах каждый член команды должен ответить на три вопроса.”*  Однажды наш скрам-мастер предложил устраивать встречи каждый день. Да! Каждый день, каждое утро, в одно и то же время!  Я считаю, что это очень полезно для новых команд – для людей, которым еще не так комфортно чувствуют себя друг с другом или с проектом. Эти ежедневные встречи, названные ежедневный скрам, устраиваются ежедневно в одно и то же время. Их нужно проводить перед выполнением любой работы в этот день. В моей команде мы устанавливали время в 10 утра каждое утро, и это было сложно. Тем не менее, это было правильное решение.  Ежедневный скрам – короткая и простая встреча (не больше 15 минут). Ее задача – помочь членам команды понять, кто что делает и определить проблемы и узкие места в разработке проекта.  **Совет:** Так как мы хотим убедиться, что эти встречи будут короткими, мы стоим. Люди обычно устают после 15 минут стояния, что отлично! Если вы увидите, что ваши коллеги ищут места, чтобы присесть и отдохнуть, наверное, встреча затянулась.  На этих встречах каждый член команды должен ответить на три вопроса:   * Что вы сделали вчера? – Ожидается короткий ответ, максимум 2-3 предложения. * Что вы планируете сделать сегодня? – такой же коротки ответ, например: «Сегодня я буду работать над этой историей». * Есть ли у вас какие-то проблемы? Если да, то какие? Можно ли их быстро решить? Ответ должен давать представление о проблеме и решении, если оно известно. Не нужно устраивать подробное обсуждение во время встречи. Скрам-мастер должен отметить проблему и работать над ее решением вместе с командой после завершения встречи.   Решение проблем разработчиков должно быть первостепенной задачей команды, чтобы они могли продолжить разработку как можно скорее. Часто человек может решить проблему через некоторое время. Иногда он или она нуждается в помощи члена команды. А если проблема серьезна, команде придется остановиться и сконцентрироваться исключительно на устранении того, что мешает им продолжать работу.  Я помню, что моя команда несколько раз сталкивалась с такими препятствиями. Были задачи и истории, которые казались довольно очевидными на первый взгляд, но после того, как пара или один программист углублялись в нее, очевидное становилось сомнительным и неправильным. Мы несколько раз обнаруживали, что сторонняя библиотека не дает нам необходимого функционала и в результате концентрировались на поиске другой, более способной, или даже искали решение самостоятельно.  Большая часть наших проектов написана на PHP. В некоторый момент нам нужно было связать наш проект с VMWare. Мы изучили официальные библиотеки для VMWare API и обнаружили версии для Java и Perl. Также существовала неофициальная версия для Ruby. Мы были уверены, что сможем использовать одну из них и просто выполнять запросы exec() из PHP для захвата вывода как строки. Как мы думали, ее разбор будет проще простого.  Оказалось, что это было почти невозможно. Ни одна библиотека API не работала так, как мы от нее ожидали. Некоторые были заброшены или не завершены, их вывод было почти невозможно разобрать. Наконец, нам пришлось делать то, чего никто не делал раньше: реализовывать библиотеку VMWare API на PHP. К сожалению, другого подходящего способа не было.  Проблема была серьезной; она отодвинула начальные планы на недели! Конечно, наш владелец продукта был немедленно осведомлен, вместе с ним мы спланировали новое расписание и создали новые истории, которые включали создание этой библиотеки API.  Чаще же ваши проблемы не будут такими серьезными. Люди могут запутаться в сложной логике, но на следующее утро у них уже будут идеи и решения. В другое время они могут просто пойти не по тому пути, и нужна будет помощь члена команды, чтобы они это поняли. Вот какими будут ваши основные проблемы. |
| **Conclusion**  Here we are at the conclusion. At least, this is how my team got started with Scrum. Some rules were very useful; others less so. Further, some rules were only useful for a short time, while others are still respected religiously by our team.  I can only recommend that you embrace the agile process, try out Scrum, and form your own conclusions. I’m sure that you’ll find bits and pieces to adopt for your team. Be agile, adapt it for your style of work, for you projects and your personalities, and don’t be afraid of adding your own custom rules. After all, Agile refers to adaptation, not blindly following a set of pre-determined rules!  *“For more top-shelf eBooks, courses, and tutorials, like this one, be sure to consider signing up for* [*Tuts+ Premium!*](http://tutsplus.com)*”* | **Заключение**  Вот мы и пришли к заключению. По крайней мере, так моя команда начинала применять скрам. Некоторые правила были очень полезны, другие – менее. Также некоторые правила были полезны некоторое время, в то время как другие все еще уважаются как заповеди в нашей команде.  Я могу лишь посоветовать вам обратиться к гибкому программированию, попробовать скрам и сделать собственные выводы. Я уверен, что вы найдете то, что сможете применить в вашей команде. Будьте гибкими, адаптируйте его под ваш стиль работы, проекты и индивидуальность, не бойтесь добавлять ваши собственные правила. В конце концов, гибкость означает адаптацию, а не слепое следование предопределенным правилам!  *“Чтобы получить больше замечательных электронных книг, курсов и руководств, вроде этого, подумайте о регистрации в* [*Tuts+ Premium!*](http://tutsplus.com)*”* |