УДК 004.413

УПРАВЛЕНИЕ ПРАКТИКАМИ В ГИБКОМ ПРОЦЕССЕ

Кетов А.В., Моисеев В.В., Лебедкин П.В., Пачуфаров Д.О.
Научный руководитель профессор Булакина Е.Н.

Сибирский федеральный университет

В настоящее время существует множество программных продуктов для команд разработчиков, предназначенных для организации эффективного процесса разработки программного обеспечения. Тем не менее, процесс разработки программного обеспечения в большой степени является творческим. Его невозможно полностью строго формализовать и решать возникающие проблемы, полагаясь только на технические средства, не учитывая особенностей социального взаимодействия в команде и индивидуальных особенностей разработчиков. Всегда существует множество различных решений одной и той же задачи, но с различными эффективностью, сложностью интеграции и стоимостью сопровождения данного решения. Довольно часто при выборе решения эти неучтенные особенности играют ключевую роль, в результате решение получается не оптимальным, особенно в долгосрочной перспективе.

Отдельно стоит рассмотреть ошибки интеграции, так как они являются одними из самых дорогих и сложных в прогнозировании ошибок. Данные ошибки возникают, когда два или более корректных компонента системы, полностью работоспособных и протестированных, начинают взаимодействовать, и поведение одних компонентов не соответствует ожидаемому другими компонентами. Ошибка интеграции, помимо собственного негативного действия, может спровоцировать еще два вида ошибок: системные, модульные. Большинство модульных ошибок легко находятся модульным тестированием и тестированием интеграции.

Системные ошибки проходят эти тесты так как являются отклонением в поведении всей системы в целом, но нормой для отдельных модулей.

Основными причинами возникновения ошибок интеграции и спровоцированных ими ошибок других типов являются:

- недостаточное планирование интеграции на стадии проектирования;
- недостаточное общение между группами разработчиков, реализующими различные компоненты;
- различия в применяемых шаблонах проектирования среди разработчиков;
- отсутствие видения полной картины проекта.

Такие практики гибких методологий как, например, Daily Scrum или ретроспектива иногда позволяют вскрыть причины возникновения таких ошибок, но ввиду своей ориентированности на текущие проблемы в рамках одной итерации, подобные практики редко позволяют выявить ошибки связанные с интеграциями, которые проходят по окончанию нескольких итераций. Так же усугубляет проблему отсутствие или ограниченное применение архитектурного планирования во многих гибких методологиях. Особенно остро проблема встает в проектах, в которых

невозможно организовать четкий итеративный процесс ввиду постоянно меняющихся задач, требований и приоритетов.

В рамках исследования данной проблемы, был произведен анализ работы двух команд работающих в схожих условиях формирования требований. Процесс первой команды не имеет выраженной методологии и близок к тому, что разработчики называют code & fix. Процесс второй команды основан на принципах Lean (Бережливое производство).

Команды находились в схожих условиях: непрогнозируемое появление задач, внезапное изменение приоритетов, жесткие ограничение по времени, постоянные интеграции, несколько релизов за день и другие негативные факторы, не позволяющие организовать четкий и отлаженный итеративный процесс. При этом использующая Lean команда редко не укладывалась в сроки и намного меньше испытывала проблем с ошибками интеграции.

Особо остро проблемы интеграции проявились при расширении штата: каждый новый сотрудник повторял одни и те же ошибки. Перенос опыта в виде семинаров, закрепления новичков за опытными сотрудниками не дал хороших результатов. Кроме того, увеличение количества сотрудников привело к увеличению количества компонентов интегрируемых за один раз в релиз продукта, и опытные разработчики так же начали повторять ошибки, приводящие к проблемам связанных с интеграцией.

Исходя из этого, был проведен анализ процессов в этих двух командах, и предпринята попытка внедрить практики и принципы Lean. В результате чего в очередной раз подтвердилось мнение, что полноценное внедрение гибких методологий возможно только при принятии командой ценностей методологии и понимании основных принципов. В существующих условиях стресса, и сильного дефицита времени, вызванного ожиданием руководством линейного роста производительности пропорционального увеличению количества разработчиков в команде, обучение команды новой методологии не представлялось возможным. Таким образом, было принято решение создать несколько практик, необходимых для успешной интеграции, не требующих много времени для внедрения, и простой процесс для обучения и эффективной передачи практик между разработчиками.

Исходя из наблюдений за процессом и интеграциями, в частности, был создан список плохих и хороших практик обязательных к ознакомлению. Данное решение дало положительный результат, но заметно ниже ожидаемого. Данный список довольно быстро вышел из регулярного употребления, многие разработчики вспомнили о нем только тогда, когда кто-либо совершал грубую ошибку, описанную в нем.

Причина непопулярности данного решения была в том, что списка практик никогда не было под рукой. Также, в большинстве случаев нет личной мотивации для того, чтобы перечитать его или зайти на корпоративный сетевой ресурс, чтобы получить к нему доступ.

Следует обратить внимание на то, что задача состоит не в том, чтобы разработчики знали плохие и хорошие практики, а применяли это знание в работе. Для того чтобы уйти от отношения разработчиков «да я знаю это!» к реальным действиям была внедрена практика «мягкого» ознакомления: помимо списка в общедоступном месте, лидер команды систематически, но ненавязчиво для разработчиков, в виде советов и рекомендаций напоминал о необходимых действиях. Посредством запланированных целенаправленных, но ненавязчивых советов при выдаче задания были сформированы новые хорошие практики. Далее с помощью того же способа эти практики превратились в привычку. Систематическое формирование подобных привычек, через совет и последующее действие, требует от лидера команды и старших разработчиков дополнительных затрат времени и некоторых коммуникационных способностей, но позволяют значительно сократить количество ошибок, особенно новыми сотрудниками. В дальнейшем, когда такая практика закрепляется в команде, разработчики сами начинают передавать опыт схожим образом.