МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Мегафакультет компьютерных технологий и управления

Факультет программной инженерии и компьютерной техники

Направление (специальность) – 09.04.04 Программная инженерия

Специализация – Веб-технологии

Научно-исследовательская работа

ТЕМА НИР: АНАЛИЗ ПРАКТИКИ НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ

ЭТАП 1. АНАЛИЗ ЗАРУБЕЖНЫХ И ОТЕЧЕСТВЕННЫХ НАУЧНЫХ ИСТОЧНИКОВ ПО ТЕМЕ ИССЛЕДОВАНИЯ

ВЫПОЛНИЛ

Студент группы Р4107 _____ А.В. Кочетыгов

№ группы подпись, дата ФИО

ПРОВЕРИЛ д.педагог.н., профессор _____ И.Б. Готская

ученая степень, должность подпись, дата ФИО

САНКТ-ПЕТЕРБУРГ

2019 г.

Оглавление

Введение 3
1. Зарождение и популяризация 6
2. Распространенность применения 8
3. Решаемая проблема 13
4. Издержки.....17
5. Преимущества.....20
6. Дополнение других практик 22
Заключение 24
Список литературы 25

Введение

В настоящее время требования к программным продуктам со стороны конечных пользователей и заказчиков неуклонно растут. Вместе с этим растет масштабность проектов и соответственно проектных команд. Кроме того, разделение обязанностей в командах становится все более ярко выраженным. Все это способствует возникновению сопутствующих проблем свойственных человеческому взаимодействию в больших группах.

Одной из таких проблем является проблема интеграции отдельных частей проекта в единое целое с последующим тестированием. Наиболее эффективным решением в данном случае может стать практика непрерывной интеграции, так как она позволяет в большом объеме автоматизировать процедуру интеграции и тестирования. Она требует ответственного подхода к внедрению, но в значительной степени окупается, так как помогает эффективно решать проблемы одного из самых подверженных издержкам процесса в ходе разработке программного обеспечения.

Анализ источников по данной тематике показывает сильную промышленную направленность в исследованиях данной практики. Однако вместе с этим академическая направленность довольно слабо развита. Так в одном из исследований, за 2016 год, утверждается, что из 69 рассмотренных работ только 5 проведены в академических, а не промышленных условиях [13]. Кроме того, стоит отметить, что основное внимание данной тематике уделяется в большинстве своем в зарубежных источниках.

Однако в научных исследованиях малое внимание уделяется области действия данной практики, тому, как она вписывается в контекст других существующих практик в разработке программного обеспечения, а также тому какие издержки и преимущества действительно проявляются при применении в реальных проектах. Все это не позволяет сформировать наиболее полного представления о практике непрерывной интеграции, что обуславливает **актуальность темы научно-исследовательской работы**.

Целью научно-исследовательской работы является формирование общего представления о концепциях, лежащих в основе практики непрерывной интеграции.

Объектом исследования выступает практика непрерывной интеграции. **Предметом исследования** являются результаты анализа отечественных и зарубежных источников по данной тематике. Областью исследования является проектирование, разработка и внедрение программного обеспечения.

Для достижения поставленной цели на 1 этапе научно-исследовательской работы необходимо было выполнить следующие задачи:

- сформировать список источников составляющих основу проводимого исследования и провести его детальный анализ;
- выполнить обзор истории зарождения и популяризации практики непрерывной интеграции;
- провести анализ распространенности применения практики непрерывной интеграции, отражающий действительность;
- описать проблему, решаемую практикой непрерывной интеграции;
- определить теоретические издержки и преимущества практики непрерывной интеграции, а также их соответствие действительности;
- рассмотреть практику непрерывной интеграции с точки зрения дополнения других практик в разработке программного обеспечения.

Теоретической основой проводимого исследования выступили работы зарубежных (*Duval P., Hilton M., Tunnell T., Sen-Tarng L., Shahin M., Ali M.* и другие) авторов, посвященные непрерывной интеграции. Поиск среди отечественных авторов по данной тематике не дал удовлетворительных результатов. Кроме того, также в целом стоит отметить малочисленность научных источников по данной тематике.

Для решения задач, поставленных в рамках исследования, применялся комплекс теоретических методов исследования, таких как сравнительный анализ, обобщение, синтез, индукция и дедукция.

Основной информационной базой проводимого исследования послужили портал *ResearchGate*, поисковая система по полным текстам научных работ *Google Scholar*, техническая исследовательская база данных *IEEE Xplore*.

Теоретический результат. Анализ работ зарубежных авторов по данной тематике позволил:

- выполнить обзор истории зарождения и популяризации практики непрерывной интеграции;
- выявить распространенность применения практики непрерывной интеграции в реальных проектах;
- определить проблему, решаемую практикой непрерывной интеграции;
- выявить как теоретические, так и действительные издержки, и преимущества практики непрерывной интеграции;
- определить возможности применения непрерывной интеграции с другими практиками в разработке программного обеспечения.

1. Зарождение и популяризация

Когда участники процесса разработки программного обеспечения знакомятся с понятием “непрерывная интеграция” (англ. *Continuous Integration*), то они скорее всего могут задаваться вопросом о том не является ли данная практика очередным новомодным трендом, который может быть не более чем временным явлением.

Впервые словосочетание “*Continuous Integration*”, упоминается в работе “*Object-oriented analysis and design with applications*” за авторством Грэди Буча, которая была опубликована в 1994 году [1].

Затем важность практики непрерывной интеграции рассматривается в статье “*Extreme programming: A humanistic discipline of software development*” за авторством Кента Бека, которая была опубликована в 1998 году [2]. Кроме того, он дал наиболее полное описание практики непрерывной интеграции в своей книге под названием “*Extreme programming explained: Embrace change*”, опубликованной в 1999 году [3].

Приведенные исторические факты также отмечаются в других исследованиях посвященных практике непрерывной интеграции [4], [5].

Однако набирать значимую популярность практика непрерывной интеграции начала с 2000 года, после того как 10 сентября Мартин Фаулер и Мэтью Феммель опубликовали статью, которая описывает понятие непрерывной интеграции и ее особенности [4], [5], [6], [7].

Так Мартин Фаулер дает следующее определение рассматриваемой концепции [7]: “Непрерывная интеграция - способ разработки программного обеспечения, при котором все участники группы осуществляют частую интеграцию результатов своей работы. Обычно каждый человек выполняет интеграцию ежедневно, что приводит к нескольким интеграциям в день. Результат каждой интеграции автоматически проверяется на предмет ошибок по возможности быстрее. Большинство групп находит, что данный подход

ведет к значительному уменьшению количества проблем интеграции и позволяет ускорить разработку связанного программного обеспечения.”

Согласно данным электронного ресурса *Google Trends*, доступным за период с 2004 года по январь 2019 года, динамика популярности запросов “*Continuous Integration*” и “Непрерывная интеграция” демонстрирует стабильный рост, отражающий развитие интереса к данной практике. Так частота англоязычной версии запроса росла до 2017 года, после чего достигла своего пика и стабилизировалась, в то время как частота русскоязычной версии, по состоянию на январь 2019 года, все еще продолжает свой рост.

Для англоязычной версии динамика популярности поискового запроса представлена на рисунке 1, а русскоязычной версии на рисунке 2.

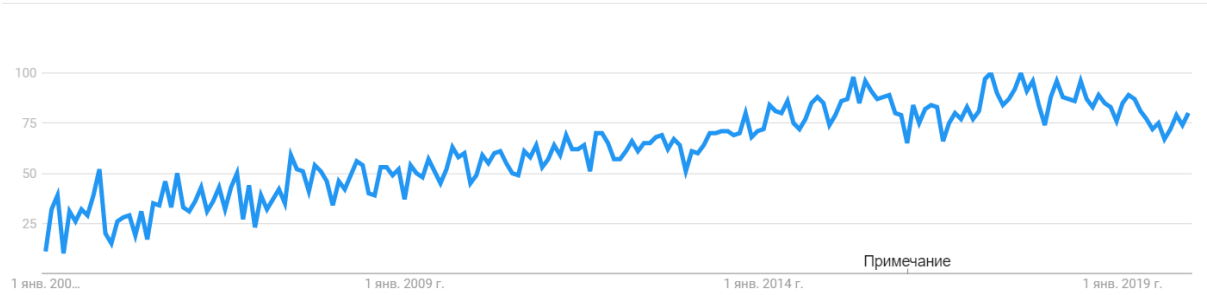

Рисунок 1 - Динамика популярности запроса “*Continuous Integration*” [18]

Рисунок 2 - Динамика популярности запроса “Непрерывная интеграция” [18]

Таким образом, можно сделать вывод о том, что непрерывная интеграция это не новомодное веяние, которое потенциально исчезнет в скором времени, а вполне зрелая практика, что явно подкрепляется тенденциями роста соответствующих поисковых запросов на протяжение длительного периода времени.

2. Распространенность применения

Популярность практики непрерывной интеграции, рассмотренная в предыдущем разделе, еще ничего не говорит о ее действительной применимости и полезности в реальных проектах. Чтобы разобраться в данном вопросе необходимо обратиться к реальной статистике о применении данной практики, чему и будет посвящен данный раздел. Стоит отметить, что приводимые далее данные базируются на выборке проектов *GitHub*, с открытым исходным кодом.

В первую очередь стоит отметить, что около 40% проектов из рассматриваемой выборки уже применяют практику непрерывной интеграции. Соответствующие данные приведены в таблице 1, а их визуализация на рисунке 3 [5].

Таблица 1 - Количество проектов, применяющих непрерывную интеграцию

Применяется	Процент	Кол-во проектов
Да	40.27%	13910
Нет	59.73%	20634

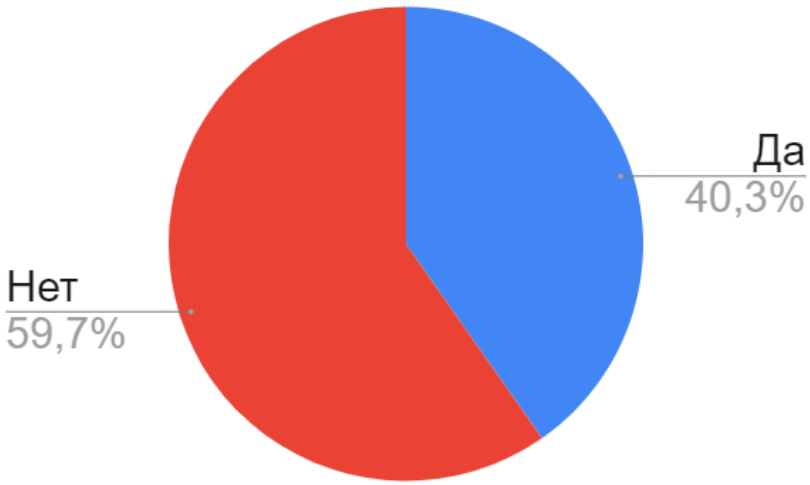

Рисунок 3 – Соотношение проектов, применяющих непрерывную интеграцию, и проектов, не применяющих ее

При этом около 90% проектов используют для этого сервис *Travis CI*, а около 14% сразу несколько сервисов. Соответствующие данные, а также данные по другим сервисам приведены в таблице 2, а их визуализация на рисунке 4 [5].

Таблица 2 - Статистика применения сервисов непрерывной интеграции

Проекты	*Travis*	*CircleCI*	*AppVeyor*	*CloudBees*	*Werker*
Доля	90.1%	19.1%	3.5%	1.6%	0.4%
Кол-во	12528	2657	484	223	59

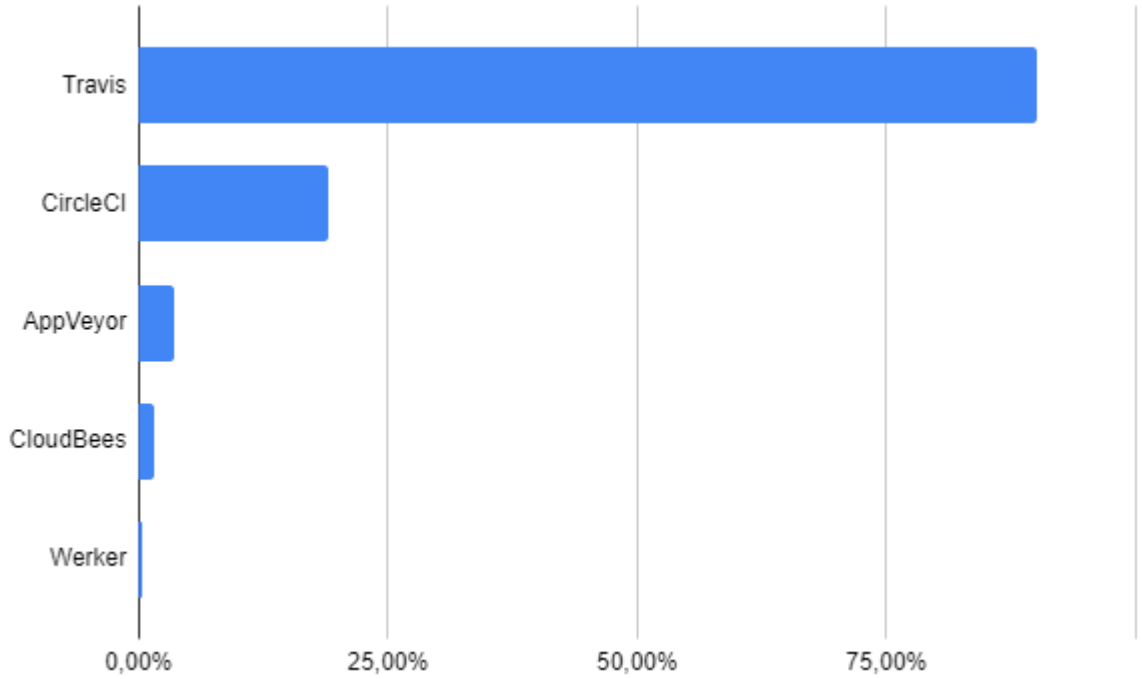

Рисунок 4 – Статистика применения сервисов непрерывной интеграции

На сервисе *GitHub* каждому зарегистрированному пользователю дается возможность поставить интересующему проекту звезду, то есть изъявить заинтересованность в проекте. Таким образом, чем больше звезд набирает проект, тем более популярным он считается. И в данном случае между популярностью проекта и применением практики непрерывной интеграции также существует корреляция. Так около 70% популярных проектов применяют данную практику, в то время как среди наименее популярных

проектов доля ее применения только 23%. Данная зависимость представлена на рисунке 5 [5].

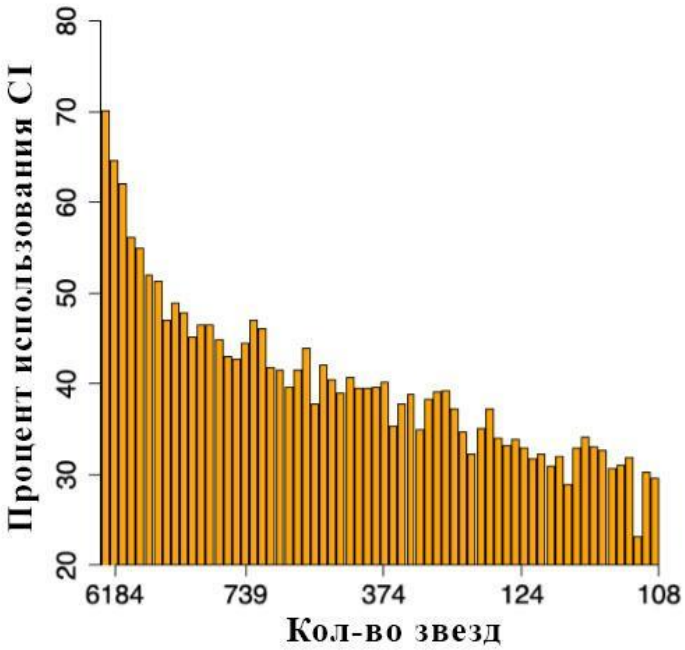

Рисунок 5 - Зависимость между популярностью проекта и применением непрерывной интеграции [5]

В то же время применение непрерывной интеграции никак не зависит от популярности используемого языка программирования. Однако стоит отметить, что высокие показатели применения непрерывной интеграции наблюдаются на проектах, которые используют языки программирования с динамическое типизацией, такие как *Ruby*, *PHP*, *CoffeeScript*, *Clojure*, *Python* и *JavaScript*. Соответствующие данные представлены в таблице 3 [5].

Таблица 3 - Статистика применения непрерывной интеграции с языками программирования

Язык	Всего проектов	С непрерывной интеграцией, шт.	С непрерывной интеграцией, %
Scala	329	221	67.17
Ruby	2721	1758	64.61
Go	1159	702	60.57
PHP	1806	982	54.37
CoffeeScript	343	176	51.31
Clojure	323	152	47.06
Python	3113	1438	46.19
Emacs Lisp	150	67	44.67
JavaScript	8495	3692	43.46
Other	1710	714	41.75
C++	1233	483	39.17
Swift	723	273	37.76
Java	3371	1188	35.24
C	1321	440	33.31
C#	652	188	28.83
Perl	140	38	27.14
Shell	709	185	26.09
HTML	948	241	25.42
CSS	937	194	20.70
Objective-C	2745	561	20.44
VimL	314	59	18.79

И наконец около 94% опрошенных в данном исследовании разработчиков утверждают, что они определенно будут применять систему непрерывной интеграции в своем следующем проекте. При том, что даже 53% разработчиков, делающих данное утверждение на момент опроса, еще не применяли данную практику. Соответствующие данные представлены на рисунке 6 [5].

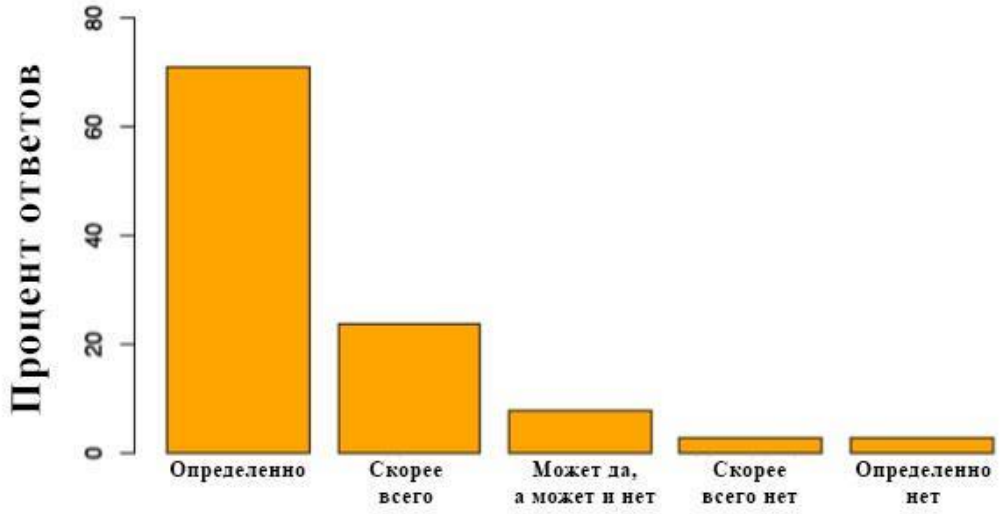

Рисунок 6 - Статистика ответов на вопрос “Будете ли вы применять непрерывную интеграцию в следующем проекте?” [5]

Таким образом, на основе всей приведенной, в данном разделе, статистики можно более уверенно утверждать, что растущая популярность данной практики является вполне оправданной и это подтверждается как минимум тем, что уже около 40% проектов рассмотренной выборки успешно ее применяют. Однако необходимо выявить причины применения разработчиками данной практики, а также определить, какие издержки возникают при ее применении и какие она дает преимущества, чему и будут посвящены следующие разделы.

3. Решаемая проблема

Иногда участники процесса разработки программного обеспечения склонны слишком много прибегать к предположениям в своей проектной деятельности, что является одной из важнейших проблем разработки, так как они по своей природе случайны и рано или поздно могут привести к трудно разрешимым проблемам проекта [7].

Наиболее часто последствия предположений вскрываются при интеграции и тестировании проекта, когда его уже необходимо сдать заказчику [7]. В результате сроки сдачи смещаются и возникает необходимость доработки проекта, с целью устранения проблем интеграции, что порождает около 40% всех издержек процесса разработки. Соотношение издержек между различными видами проектной деятельности представлены в таблице 4 [8].

Таблица 4 - Соотношение издержек между различными видами проектной деятельности

Деятельность	Издержки
Управление	5%
Определение требований	5%
Определение дизайна	10%
Написание кода и юнит-тесты	30%
Интеграция и тестирование	40%
Развертывание	5%
Подготовка сред	5%

Использование предположений, в первую очередь, подразумевает то, что по факту проекты большую часть времени проводят в заведомо нерабочем состоянии, так как в течении всего периода разработки никто даже не пытается развернуть проект в полном объеме и попытаться использовать его в среде максимально близкой к рабочей. Все это следствие мыслей разработчиков о

том, что если часть проекта, которую они разрабатывают, работоспособна, то она в конечном итоге не приведет к поломке уже целостного проекта. То есть все сводится к тому, что никто не заинтересован в комплексной проверке еще незаконченного проекта [10].

Тоже самое относится и к проектам, в которых есть промежуточные интеграции, но между ними существуют большие временные интервалы, что также повышает вероятность возникновения ошибок, так как в больших объемах изменений гораздо труднее определить проблемный участок кода [9].

Кроме того, разработчики также тратят около 20% времени на настройку и исправление их тестировочных сред [14].

Именно эти масштабные проблемы, приводящие к столь большим издержкам, и призвана устранить непрерывная интеграция. Вся ее суть предполагает частое внесение небольших порций изменений за которыми следует автоматическая интеграция и тестирование целостного проекта, что позволяет команде постоянно поддерживать проект в рабочем состоянии, так как система будет оповещать ее об ошибках. Соответственно исправление возникающих ошибок всегда будет первостепенной задачей, ведь без этого попросту нельзя будет продолжить дальнейшую работу [9, 10].

Так в одном из исследований, посвященных внедрению интеграции в крупном проекте приводится информация о том, что данная практика позволила сократить количество неустраненных дефектов на 90% и практически свести на нет необходимость устранения найденных пользователями дефектов, при том что ранее около 30% рабочей силы тратилось как раз на устранение дефектов [14].

При этом для обеспечения эффективности данной практики, необходимо соблюдать следующие обязательные требования [10]:

- регулярно регистрировать изменения, что является наиболее важной процедурой. Выполнять необходимо минимум два раза в день. При

этом вероятность повредить сборку будет минимальной и всегда можно вернуться к более хорошей версии кода;

- иметь полный набор автоматических тестов, что будет гарантировать работоспособность приложения. Должны присутствовать три вида тестов [12]:
  - модульные, предназначенные для проверки поведения небольших частей приложения, изолированных друг от друга;
  - компонентные, предназначенные для тестирования поведения одновременно нескольких частей приложения. могут проверять базу данных, файловую систему или другие системы;
  - приемочные, предназначенные для проверки удовлетворения критериев, поставленных заказчиком. могут относится как к функциональности, так и к качественным показателям: производительности, доступности, безопасности и т.д.;
- обеспечить быстроту процессов сборки и тестирования. В данные процессы должны занимать не более нескольких минут. При этом десять минут верхний предел, а пять минут - неплохо время;
- предоставить хорошую изолированную среду разработки. У разработчиков всегда должна быть возможность выполнять сборку, тестирование и развертывание в среде, находящейся под их контролем. Конфигурации, а также сценарии для баз данных, сборок и развертывания должны храниться в системе контроля версий.

Однако стоит отменить, что руководящее положение "частых изменений” в действительности соблюдается лишь в некоторой степени, причем между проектами существуют значительные различия в плане соблюдения этого руководства. Кроме того, задержка подтверждения запроса на принятие изменений возрастает несмотря на то, что самих изменений становится меньше [15].

Также обеспечение быстроты процессов сборки и тестирования является одним из наиболее проблемных аспектов непрерывной интеграции. Так исследование 104442 сборок, в рамках практики непрерывной интеграции, среди 67 проектов на *GitHub*, которые используют сервис *Travis CI*, показывает, что продолжительность сборки, как правило, составляет от 8 до 90 минут (средняя продолжительность около 20 минут). При этом 84% сборок выполняется свыше приемлемого предела в 10 минут, 40% сборок выполняются более 30 минут. Распространенными причинами таких задержек являются: запуски сборок в будни дни или дневное время; множественный перезапуск команд (дающий лишь 3% шанс того, что сборка все же будет выполнена успешно); отсутствие кэширования редко меняющегося содержимого [16].

Таким образом, практика непрерывной интеграции позволяет решить большинство проблем интеграции проекта еще на самой ранней стадии, путем сдвига парадигмы программирования в сторону того, что приложение на протяжении всего периода разработки должно всегда оставаться работоспособным, а устранение проблем интеграции должно иметь наивысший приоритет для всей команды еще на стадии разработки. При этом для наибольшей эффективности необходимо соблюдать пять основных требований, среди которых наиболее важными являются: наличие автоматизированных тестов и частая регистрация изменений.

4. Издержки

Существуют мнения о том, что применению практики непрерывной интеграции могут помешать [7]:

- дополнительные затраты на поддержку данной системы. Однако это не так, поскольку необходимость процесса интеграции существует всегда, вне зависимости от применения данной практики, и поддерживать ее проще, чем контролируемые вручную процессы;
- внедрение повлечет слишком много изменений уже существующих процессов. Однако это можно нивелировать путем инкрементного перехода;
- дополнительные издержки на аппаратный и программные средства;
- разработчики все равно будут должны выполнять эти действия. Однако это всего лишь одно из распространенных заблуждений руководства.

Опрос проектов, с открытым исходным кодом, которые используют сервис *Travis CI*, показывает реальную статистику причин, по которым проекты не используют данную практику. Данная статистика отражена в таблице 5, а их визуализация на рисунке 7 [5].

Таблица 5 - Причины отказа разработчиков от практики непрерывной интеграции

Причина	%
Недостаток знаний о практике	47.00
В проекте нет автоматизированных тестов	44.12
Изменения в проект вносятся не так часто	35.29
Она не используется сейчас, но в будущем будет	26.47
Система непрерывной интеграции имеет высокие издержки обслуживания	20.59

Таблица 5. Продолжение

Система непрерывной интеграции требует слишком много времени на установку	17.65
Проекту и так хватает тестов	5.88

Рисунок 7 - Причины отказа разработчиков от практики непрерывной интеграции

В данных проектах смена конфигурации системы непрерывной интеграции в среднем проводится около 12 раз за время существования проекта. Также стоит отметить, что 25% проектов меняли конфигурацию 5 раз или меньше. При этом существенная настройка производится только при первоначальной установке, а в дальнейшем вносятся лишь незначительные изменения, большая часть которых касается зависимостей проекта [5].

В одном из других рассматриваемых исследований, 50% разработчиков также утверждают, что инструменты и сервисы непрерывной интеграции сложны в конфигурировании и имеют слабую интеграцию с другими инструментами [17].

Таким образом, можно сделать вывод о том, что многие утверждения о недостатках практики непрерывной интеграции делаются с позиции отсутствия достаточных знаний о ней, так как 47.70% проектов, которые ее не используют, попросту не обладают достаточными знаниями о ней. В то время как всего 20.59% проектов отказываются от непрерывной интеграции по причине высоких издержек и 17.65% по причине того, что установка системы потребует слишком много времени.

5. Преимущества

Непрерывная интеграция дает ряд существенных преимуществ, которые особенно важны в больших проектах, где в разработке участвует большое количество людей [7].

Частая интеграция проекта позволяет снизить риски за счет [7]:

- раннего обнаружения и устранения дефектов;
- постоянного контроля за состоянием проекта;
- уменьшения количества совершаемых предположений.

Автоматизированность процесса свойственная практике непрерывной интеграции позволяет уменьшить количество рутинных действий, таких как: компиляция, интеграция базы данных, тестирование и развертывание. Что достигается за счет того, что [7]:

- процесс каждый раз выполняется одинаково;
- поддерживается упорядоченность операций;
- процесс осуществляется при каждом изменении в хранилище с контролем версий.

Все это позволяет снизить трудозатраты рутинные процессы (компиляция, тестирование, развертывание) и освободитель людей для более важных работ, а также повысить качество контроля над проектом за счет [7]:

- постоянного наличия информации о текущем состоянии и качественных показателях последней сборки проекта;
- возможности отслеживать тенденции, общего качества и другой информации о проекте.

Повышенное доверие к программному продукту как со стороны участников проекта, так и со стороны его потребителей, так как участники проекта всегда знают как их изменения повлияли на общий код, а потребители имеют возможность наблюдать за развитием проекта [7].

Опрос проектов, с открытым исходным кодом, которые используют инструмент *Travis CI*, показывает, что 87.71% проектов меньше беспокоятся о

нарушениях в процессе сборки. Также 79.61% проектов раньше обнаруживают дефекты. Это две наиболее распространенные причины использования практики непрерывной интеграции среди проектов с открытым исходным кодом. В то же время лишь 33.66% проектам непрерывная интеграция позволяет проводить меньше времени за отладкой [5].

Также, согласно статистике, проекты, использующие практику непрерывной интеграции, по сравнению с проектами, которые ее не используют, публикуют новые версии в 2 раза чаще и отвечают по запросам на изменение исходного кода в 1.6 раза быстрее. При этом 72.03% сборок основной ветки, в хранилище с контролем версий, выполняются успешно [5].

В одном из других рассматриваемых исследований, среди 523 опрошенных разработчиков со всего мира, 78% утверждают, что чувствуют себя более продуктивными с практикой непрерывной интеграции, а 85% утверждают, что непрерывная интеграция заставляет их большее значение придавать автоматизированным тестам [17].

Таким образом, можно сделать вывод о том, что преимущества использования практики непрерывной интеграции подтверждаются реальными фактами. В частности, данная практика действительно позволяет меньше беспокоится о нарушениях в процессе сборки за счет автоматизации процесса и об этом свидетельствуют 87.71% проектов. Также она действительно позволяет раньше обнаруживать дефекты и об этом свидетельствую 79.61% проектов. Увеличенная частота публикации новых версий и скорость принятия запросов косвенно подтверждает факт повышенного контроля над качеством проекта.

6. Дополнение других практик

Практика непрерывной интеграции является частью и дополнением некоторых других практик разработки программного обеспечения.

Так практика непрерывной поставки (англ. *Continuous Delivery*) комбинирует в себе непрерывную интеграцию и автоматизацию развертывания для обеспечения автоматической доставки программного обеспечения в рабочую среду после прохождения автоматизированных тестов и проверок качества [13].

Непрерывная поставка в свою очередь является частью практики непрерывного развертывания (англ. *Continuous Deployment*), которая заходит еще дальше и в дополнении к уже автоматизированным процессам интеграции, тестирования и доставки также выполняет автоматическое развертывание проектов в рабочих средах [13].

Основное отличие непрерывного развертывания от непрерывной поставки заключается в том, что она исключает участие человека в процессе разворачивания приложения в рабочей среде.

Также непрерывной интеграции отлично дополняет практики [7]:

- соблюдения стандартов программирования, так как в процессе сценария сборки могут запускаться инструменты статического анализа исходного кода и оповещать об отклонениях от принятых в разработке стандартов;
- рефакторинга, так как в процессе сценария сборки могут запускаться инструменты инспекции, которые способны обнаружить потенциально проблемные области кода;
- малых выпусков, ведь при использовании данной практики проект практические всегда находиться в полностью рабочем состоянии и можно совершать выпуски так часто, как это нужно;

- коллективной собственности, так как она предотвращает ситуацию, при которой только одному человеку известно, что именно содержит и как работает определенная часть системы.

Таким образом, можно сделать вывод о том, что практика непрерывной интеграции является фундаментальной основой и дополнением других успешных практик разработки программного обеспечения.

Заключение

В результате проведенного исследования была описана практика непрерывной интеграции, история ее зарождения и популяризации, рассмотрена проблема, на решение которой, в большей степени, направлена данная практика, а также приносимые ей издержки и преимущества в ходе разработки программного обеспечения. Кроме того, было рассмотрено то, как непрерывная интеграция может дополнять другие практики в разработке программного обеспечения.

Использование практики непрерывной интеграции позволяет существенно улучшить производительность команды, в особенности большой, в процессе интеграции и тестирования проекта, где возникает около 40% всех проектных издержек.

Около 40% проектов рассмотренной выборки уже успешно применяют данную практику и около 87% из них утверждают, что данная практика позволяет им меньше беспокоится о проблемах интеграции и тестирования, а также около 79% проектов утверждают, что практика позволяет им намного раньше обнаруживать дефекты. При этом в 47% случаев отказов от применения данной практики основной причиной является недостаток знаний о ней.

Стоит отметить, что на рынке существует ряд сервисов, которые позволяют применять практику непрерывной интеграции в проектах, например *Travis CI*, *CircleCI*, *AppVeyor*, *CloudBees*, *Werker* и другие. При этом наиболее распространенным инструментом среди рассмотренной выборки является *Travis CI*, доля которого составляет около 90%.

В дальнейшем предполагается детальное исследования применения данной практики в контексте веб-разработки, а также связи с гибкими методологиями разработки программного обеспечения.

Список литературы

1. Grady, B. Object-oriented analysis and design with applications / B. Grady, R. A. Maksimchuk, M. W. Engel, B. J. Young, J. Conallen, K. A. Houston // Addison-Wesley. — Boston, 1994.
2. Beck, K. Extreme Programming: A Humanistic Discipline of Software Development in Fundamental Approaches to Software Engineering. Springer / K. Beck // Addison-Wesley. — Santa Clara, 1998.
3. Beck, K. Extreme Programming Explained: Embrace Change / K. Beck, C. Anders // Addison-Wesley. — Boston, 1999.
4. Lai, S. Applying Continuous Integration for Increasing The Maintenance Quality And Efficiency of Web App / S. Lai // International Journal of Software Engineering & Applications (IJSEA). — 2019. — Vol. 10. — № 1. — P. 37–50.
5. Hilton, M. Usage, Costs, and Benefits of Continuous Integration in Open-Source Projects / M. Hilton, T. Tunnell, K. Huang, D. Marinov, D. Dig // Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. — IEEE, 2016. — P. 426–437.
6. Mårtensson, T., Continuous Integration is Not About Build Systems / T. Mårtensson, P. Hammarström, J. Bosch // Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA). — IEEE, 2017. — P. 1-9.
7. Дюваль, П. М. Непрерывная интеграция: улучшение качества программного обеспечения и снижение риска / П. М. Дюваль, С. М. Матиас III, Э. Гловер // Вильямс. — Москва, 2008. — 240 с.
8. Ройс, У. Управление проектами по созданию программного обеспечения / У. Ройс // Лори. — Москва, 1998. — 431 с.
9. Дэвис, Д. Философия DevOps. Искусство управления IT / Д. Дэвис, К. Дэниелс // Питер. — Санкт-Петербург, 2017. — 416 с.

10. Хамбл, Д. Непрерывное развертывание ПО: автоматизация процессов сборки, тестирования и внедрения новых версий программ / Д. Хамбл, Д. Фарли // Вильямс. — Москва, 2011. — 432 с.
11. Эберхард, В. Continuous delivery. Практика непрерывных апдейтов / В. Эберхард // Питер. — Санкт-Петербург, 2018. — 320 с.
12. Yuksel, H.M. Using Continuous Integration and Automated Test Techniques for A Robust C4ISR System / H.M. Yuksel, E. Tuzun, E. Gelirli, E. Biyikli, B. Baykal // Proceedings of the 24th International Symposium on Computer and Information Sciences. — IEEE, 2009. — P. 734–748.
13. Shahin, M. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices / M. Shahin, M. Ali, L. Zhu // Institute of Electrical and Electronics Engineers (IEEE). — 2017. — Vol. 5. — P. 3909–3943.
14. Chen, L. Continuous Delivery: Huge Benefits, but Challenges Too / L. Chen // Institute of Electrical and Electronics Engineers (IEEE). — 2015. — Vol. 32. — № 2. — P. 50–54.
15. Zhao, Y. The Impact of Continuous Integration on Other Software Development Practices: A Large-Scale Empirical Study / Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, B. Vasilescu // Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering. — IEEE, 2017. — P. 60–71.
16. Ghaleb, T. A. An Empirical Study of the Long Duration of Continuous Integration Builds / T. A. Ghaleb, D. A. da Costa, Y. Zou // Empirical Software Engineering. — 2019. — Vol. 24. — № 4. — P. 2102–2139.
17. Hilton, M. Continuous Integration (CI) Needs and Wishes for Developers of Proprietary Code / M. Hilton, N. Nelson, D. Dig, T. Tunnell, D. Marinov // Corvallis, OR: Oregon State University, Dept. of Computer Science. — 2016.

18. Google Trends [Электронный ресурс]. Режим доступа: https://trends.google.ru/ (Дата обращения: 20.11.2019).