Моделирование и анализ данных
2021. Том 11. № 4. С. 72–86
DOI: https://doi.org/10.17759/mda.2021110406
ISSN: 2219-3758 (печатный)
ISSN: 2311-9454 (online)

Modelling and Data Analysis
2021. Vol. 11, no. 4, pp. 72–86
DOI: https://doi.org/10.17759/mda.2021110406
ISSN: 2219-3758 (print)
ISSN: 2311-9454 (online)

◇◇◇◇◇◇◇◇МЕТОДИКА ОБУЧЕНИЯ◇◇◇◇◇◇◇◇

УДК 004.413

Современные реалии управления программными проектами

Бахиркин М.В.*
Московский Авиационный Институт
(Национальный исследовательский университет), АО «Почта Банк»
г. Москва, Российская Федерация
ORCID: https://orcid.org/0000-0002-9077-4426
e-mail: bakhirkin@mail.ru

Лукин В.Н.**
Московский Авиационный Институт
(Национальный исследовательский университет)
Московский государственный психолого-педагогический университет
г. Москва, Российская Федерация
ORCID: https://orcid.org/0000-0001-8906-2686
e-mail: lukinvn@list.ru

Необходимость поддерживать рабочий процесс в условиях вынужденной разобщённости повысила потребность в программных продуктах. Но оказалось, что качество многих из них ниже, чем ожидалось. Одна из причин – с ними стали работать пользователи с иными критериями качества. Другая причина – продукты объективно стали недопустимо низкого качества, во многом из-за ошибок в управлении проектом. Масштаб и значимость современного программного проекта требуют его выполнения по одной из классических моделей: она обеспечит качество. Однако из-за желания опередить конкурентов используют гибкие методологии, сокращая сроки и теряя качество. И чем крупнее проект, тем выше риск потери качества и выше стоимость потерь. Как совместить выгоды обоих подходов, не получив их недостатков?

Ключевые слова: большой программный проект, методология разработки, Waterfall, Agile, интернет-проект.

****Бахиркин Михаил Васильевич,*** к.т.н., Московский Авиационный Институт (Национальный исследовательский университет), АО «Почта Банк», г. Москва, Российская Федерация, ORCID: https://orcid.org/0000-0002-9077-4426, e-mail: bakhirkin@mail.ru

*****Лукин Владимир Николаевич,*** к.ф.-м.н., доцент, Московский Авиационный Институт (Национальный исследовательский университет), профессор, Московский государственный психолого-педагогический университет, г. Москва, Российская Федерация, ORCID: https://orcid.org/0000-0001-8906-2686, e-mail: lukinvn@list.ru

Для цитаты:
Бахиркин М.В., Лукин В.Н. Современные реалии управления программными проектами // Моделирование и анализ данных. 2021. Том 11. № 4. С. 72—86. DOI: https://doi.org/10.17759/mda.2021110406

1. ВВЕДЕНИЕ

За последние полтора «ковидных» года спрос на и информационные услуги значительно возрос. Конечный пользователь (то есть мы с вами) хочет получить услуги с качеством, не худшим, чем без так называемой «цифровой» экономики. Однако их качество не всегда на высоте, даже возникает впечатление, что оно ухудшается. В чём причина?

Посмотрим на процесс разработки программного обеспечения. Часто к основным причинам потери качества относят усложнение процессов управления из-за дефицита специалистов-управленцев и увеличения объема работ. Однако нельзя не согласиться с мнением Р. Гласса [1], который приводит три основных причины снижения качества: неверные оценки на старте проекта, нестабильность требований и ошибки в управлении. Принятие решения о старте проекта базируется на качественных и количественных оценках затрат, времени выполнения и прогнозируемом выигрыше от внедрения [2], то есть качество на стадии инициализации проекта имеет решающее значение.

Тем не менее, анализ результатов проектной деятельности в области информационных технологий показывает тенденцию к снижению качества. Причин этому много, но одна из основных – неверная оценка проекта, так как руководитель либо не знает методик оценки, либо не умеет ими пользоваться, либо их игнорирует. В любом случае проект попадает в тяжёлое положение и теряет качество.

Несмотря на то, что на протяжении многих лет исследователи и практики пытаются найти способы повышения качества проектов, согласно исследованию Standish Group более половины проектов были либо неудачны, либо провалены.

Казалось бы, пусть «неудачники» возьмут у более удачливых коллег методики разработки, но то. что подходит для одного проекта, не обязательно подойдёт для другого, и приходится ориентироваться на цели заказчика, планируемый объем работ и бюджет.

В таблице 1 приведены данные успешности программных проектов в зависимости от масштаба.

Таблица 1

Успешность программных проектов, в зависимости от масштаба. Данные Chaos Report (в процентах)

Размер проекта	Успешные проекты	Неудачные проекты	Провалившиеся проекты
Огромный	6	51	43
Большой	11	59	30

Размер проекта	Успешные проекты	Неудачные проекты	Провалившиеся проекты
Средний	12	62	26
Небольшой	24	64	12
Маленький	61	32	7

2. МЕТОДОЛОГИИ: КРАЙНИЕ ТОЧКИ

Под методологией будем понимать систему правил, которым подчиняется разработка программного обеспечения. Она включает такие компоненты, как концепции моделирования, правила проектирования, формы представления модели предметной области, процесс проектирования и реализаций моделей.

Методология использует понятие жизненного цикла, который моделируется в виде последовательных этапов. На каждом этапе порождается определенный набор технических решений и отражающих их документов, а исходными выступают документы и решения, принятые на предыдущем этапе.

С методологией обычно связывают модель разработки: структуру, которая определяет последовательность выполнения и взаимосвязь процессов жизненного цикла, и критерии перехода от этапа к этапу. На практике термины «модель» и «методология» зачастую используются как синонимы.

Каждый класс программных продуктов для оптимизации процесса разработки требует выбора своей методологии, чаще с модификацией. Кроме того, на выбор методологии влияет характер проектной команды, параметры производства, привычка команды или руководства или другие соображения. Например, если менеджер где-то прочитал, что некто авторитетный придумал методологию, которая увеличивает производительность работ вдвое, он может попытаться применить её в качестве «серебряной пули» для спасения гибнущего проекта. Но даже если команда использует известную методологию, скажем, экстремальное программирование [5], она нередко что-то в ней меняет, например, отказывается от парного программирования. Понятно, что имеет смысл говорить только о классах методологий, не углубляясь в детали.

Оценим множество методологий управления программным проектом по условному критерию «жёсткости». Тогда можно увидеть две крайние группы: самой жёсткой, классической, соответствуют каскадные модели (Waterfall), а мягкой, более современной, – гибкие (Agile). Заметим, что вариантов каскадной модели не так много, а гибких великое множество. Однако все они обладают рядом общих свойств, так что их можно рассматривать в совокупности.

3. МЕТОДОЛОГИИ ПОЛНОГО ЦИКЛА

Классические (тяжёлые) методологии создания программного обеспечения базируются на инженерном подходе и используют модели жизненного цикла, которые включают все этапы, от анализа требований до сопровождения. В зависимости от кон-

кретной методологии характер выполнения может быть различным, процесс может содержать дополнительные этапы, но основные этапы всегда присутствуют.

Каскадная модель (водопадная, Waterfall) – исторически первая, она широко использовалась в крупных проектах. Она дала возможность перейти от примитивного ремесла к промышленному производству программных изделий. До неё оценка необходимых ресурсов происходила «на глазок», теперь она стала вычисляться по специальным моделям. Благодаря инженерным методам улучшилась управляемость производства и повысилось качество продукции. Да, увлечение «инженерным подходом» имело свои слабости, в частности, не учитывалась разница в производительности и качестве работы программистов, но это был принципиально важный шаг в сторону массового программирования.

Такой подход даёт уверенность в возможности завершения проекта в срок, в пределах бюджета и с требуемым качеством. Однако болезненная проблема подобных методологий – рост времени на устранение ошибок, сделанных на ранних этапах разработки, а выявленных на последних. Кроме того, на разработку уходят значительные ресурсы, планирование работ проводится с большим резервом. Для небольших проектов накладные расходы становятся неприемлемыми. И только если требуется высокое качество, этот вариант полезен.

Каскадная модель предполагает переход на следующий этап жизненного цикла после завершения работ предыдущего. Этап заканчивается результатом, который документируется и служит исходной позицией для следующего. Требования к системе фиксируются в начале работы над проектом и в дальнейшем не изменяются [6]. Отсюда и метафора: водопадная модель (Waterfall).

Примером разработки крупной промышленной системы может служить АСУП ППЗ – Автоматизированная система управления крупным предприятием с серийным, мелкосерийным и индивидуальным характером производства. На период разработки оно обеспечивалось двумя миллионами нормативов, в нём применялись десятки тысяч типоразмеров материалов и покупных изделий, поставляемых тысячами поставщиков. В обработке одновременно находились десятки тысяч узлов и деталей, в службах и цехах обращалось множество документов.

Уже в начале разработки обнаружились недостатки водопадной модели, особенно неприятным было позднее обнаружение проблем. Это приравнивалось к браку и приводило к большим неприятностям для автора. Тем не менее, работа была весьма успешной, чему способствовало принятие участниками разработки следующих правил:

- выделяются независимые подсистемы, чтобы распараллелить работу;
- максимально увеличивается планируемое время для предсказуемости времени окончания работ;
- проводится тщательный анализ требований и детальное проектирование;
- документация ведётся параллельно разработке;
- потребности заказчика определяются возможностями разработчиков (аналогов в мире практически нет);
- разработка ведётся самыми передовыми методами и средствами.

Итак, для уменьшения риска создания системы, которая не удовлетворяет потребностям заказчика, необходим механизм раннего обнаружения и ликвидации проблем. В рамках каскадной модели это обычно приводит к нарушению графика работ.

Модификация каскадной – поэтапная итерационная модель – предполагает наличие циклов обратной связи и корректировки между этапами, что позволяет держаться в рамках заданной функциональности. Каждый этап заканчивается сдачей результата и обсуждением его с представителями заказчика. Как только выявляется отклонение от ожидаемого результата, проект дорабатывается, на что выделяется дополнительное время. Результат обычно соответствует ожиданиям заказчика, и риск провала невелик. Но длительность разработки может стать неоправданно большой.

4. МЕТОДОЛОГИИ СОКРАЩЁННОЙ РАЗРАБОТКИ

В конце 20-го века, когда персональные компьютеры стали доступными, появилась высокая потребность в программном обеспечении. Методологии полного цикла не могли её удовлетворить, и были предложены методологии быстрой разработки: «лёгкие», «гибкие» (Agile) [7]. Они предполагают отказ от некоторых технологических этапов или их редуцирование ради получения быстрого результата. Принципы гибкой разработки сформулированы в опубликованном в начале 2000-х годов Манифесте (Agile Manifesto):

- индивидуумы и взаимодействия важнее процессов и инструментов;
- работающие программы важнее документации;
- сотрудничество с заказчиками важнее формальных договоров;
- реагирование на изменения важнее плана.

Следование базовым идеям не означает отказа от альтернативных ценностей. Взаимодействие ускоряет взаимопонимание заказчиков и разработчиков, но увеличивают риск ошибочного представления, а следование формальным процессам его уменьшает. Программы без документации трудны и дороги в сопровождении, но внедряются быстро. Сотрудничество с заказчиками – хорошая практика, но без формального договора есть риск конфликта. Реагирование на изменения важно, но без плана можно затянуть разработку до бесконечности [8, 9]. Вообще, следование лёгким методологиям обычно более рискованное дело, чем следование тяжёлым, но может дать быстрый выигрыш.

Небольшие динамичные компании, работающие в условиях сжатых сроков и быстро меняющихся требований, стараются использовать лёгкие методологии. Их внедрение обычно не требует ни серьезных инвестиций, ни перестройки структуры фирмы. Область их применения – небольшие и средние проекты.

Оценим плюсы и минусы этих двух подходов (табл. 2).

Таблица 2

Достоинства и недостатки жёсткого и гибкого подходов

	Waterfal	***Agile***
Плюсы	прозрачная структура процессов разработки	возможность динамического изменения требований при разработке
	подробная и качественная документация системы	короткие тестируемые итерации; документация в конце проекта
	контролируемые процессы: всегда можно получить информацию о затраченных ресурсах, рисках и т.д.	гибкий процесс внесения изменений в требования: малый риск «неправильного» продукта
	качественный продукт на выходе: качество важнее затраченного времени	быстрый выход на рынок с минимально пригодным продуктом
Минусы	начальные требования к продукту не корректируются по ходу разработки	трудно определить стоимость в начале проекта из-за изменений требований
	нет обратной связи между этапами	проектная команда должна использовать одну методологию
	нет обратной связи с заказчиком: он не участвует в процессе разработки	команда проекта должна быть квалифицированной и опытной
	разработка ведётся согласно начальным требованиям: есть шанс получить устаревший продукт	новые требования могут противоречить архитектуре решений и реализованному функционалу
	итоговое тестирование происходит в конце: высокая вероятность ошибки	есть риск, что при гибкой реакции на требования проект станет бесконечным

Недостатки гибких методологий связаны с тем, что приоритеты разработки сдвинуты в сторону подготовки и сдачи продукта, приемлемого по качеству на данный момент. Не уделяется внимание длительной эксплуатации, в результате и без того дорогостоящее сопровождение становится просто обузой. Кроме того, есть риск, что получившееся изделие будет обладать рядом дефектов:

- основное внимание уделяется экранным и выходным формам, структура данных и функциональность остаются за кадром;
- демонстрация прототипа вызывает у пользователя иллюзию готовности, и это провоцирует разработчика на недостаточную функциональность;
- без строгой дисциплины порождается разнобой стилей реализации;
- функциональность развивается «по факту», структура базы данных изменяется почти бесконтрольно;
- на документацию времени не хватает, и если нет интуитивной ясности, работа с системой требует усилий;
- без постоянного интеграционного тестирования система может стать слабо связанным набором фрагментов;
- трудно поддерживать высокий уровень качества.

Если для заказчика указанные факторы риска будут критическими, то от лёгких методологий лучше отказаться.

Одной из первых моделей жизненного цикла для лёгких методологий, которая применялась ещё в RAD в конце 1980-х годов, была спиральная модель (RAD – rapid application development, концепция разработки качественных программных продуктов при сильном ограничении сроков и бюджета и нечётко определённых требованиях). Появлением своим она обязана «нетерпеливому заказчику»: ему нужно начать работу с системой, пусть даже с ограниченными возможностями. Мощность системы наращивается параллельно с эксплуатацией первой версии. Это эволюционная модель, в которой система представляется совокупностью функций с определённым приоритетом ввода в действие. Вначале тщательно проектируется и реализуется очень простая, но полезная версия системы, затем начинается работа над второй версией и так далее. Каждая версия полностью требованиям заказчика не удовлетворяет, но последовательность версий итеративно приближается к ним.

5. БОЛЬШИЕ ПРОЕКТЫ, ИНТЕРНЕТ-ПРОЕКТЫ

Применение строгих методологий наиболее эффективно для больших программных проектов. Они позволили довольно точно планировать сроки и ресурсы, необходимые для разработки, и стали доминирующими при производстве сложных программных систем.

Первая попытка создания промышленных прикладных программ – это автоматизированные системы управления (АСУ), предложенные академиком В.М. Глушковым. Для их реализации требуется изменить технологию управления, чтобы уменьшить длину информационных и управляющих цепочек, что позднее стали называть реинжинирингом бизнес-процессов (BPR). Основная роль АСУ состоит в обеспечении корректного функционирования этих цепочек, а также в накоплении информации и анализе её для поддержки принятия решений. Для успешной работы АСУ требуются большие вычислительные мощности, сложное программное обеспечение и грамотные специалисты. Создание подобных систем привел к значительному росту производства вычислительной техники, расширению её спектра, существенному улучшению её качества.

Но задача производства систем такого класса, как АСУ, для уровня развития технологии того времени оказалась трудно разрешимой, в результате чего только немногие проекты были успешно завершены. Большинство либо закончились частичной автоматизацией некоторых процессов или производств, либо были просто провалены.

Одна из причин неудач – необходимость преобразовать управление производством. Однако бюрократическая структура как раньше всячески препятствовала этому, так и до сих пор препятствует: с одной стороны, клерки не понимают её, с другой – боятся лишиться места. В результате «цифровизация» превращается в абсурд: вместо максимального сокращения оборота документов, он растёт, да ещё в бумажной форме, благо у принтера от заполнения разных бланков руки не болят! Заметим, что для больших прикладных интернет-проектов проблема становится ещё более острой. Мы вынуждены идти либо на неприемлемый риск потери качества при использова-

нии гибких методологий, либо на увеличение срока разработки, что вообще приведёт к катастрофе: за это время изменятся технологии.

И тут мы приходим к методологиям совсем другого типа, суть которых – построение крупного прикладного продукта как набора взаимосвязанных и взаимодействующих компонентов, каждый из которых обладает большой степенью автономности. Это эволюционные методологии, основанные на инкрементном подходе.

6. ИНКРЕМЕНТНАЯ МОДЕЛЬ

При проектировании больших систем в рамках «водопадной» методологии планируемый срок выполнения каждого этапа велик, и в этот период с большой вероятностью появляются дополнительные требования. На них, чтобы не потерять качество, нужно реагировать. Дополнительное время может стать сравнимым с временем реализации этапа, и возникает риск «вечной разработки» или создания неактуальной системы. При значительной неопределённости или высокой изменчивости предметной обласи результат станет весьма печальным. В этой ситуации разумно использовать эволюционную инкрементную модель. Её стадии «состоят из расширяющихся приращений оперативного программного обеспечения, с направлением эволюции, определяемым опытом работы» (Б. Боэм). По мере готовности каждое приращение передаётся в эксплуатацию, включается в общую систему и эволюционирует относительно независимо.

В отличие от водопадных моделей, инкрементная даёт возможность получать пользу даже от не полностью готового продукта, а разработчику позволяет улучшить качество следующих подсистем.

При разработке по инкрементной модели вначале определяются системные цели, формулируются требования, определяются ресурсы. Затем на основе требований разрабатывается архитектура системы, в которой учитывается разбиение её на относительно независимые замкнутые подсистемы, определяются приоритеты, в соответствии с которыми планируется разработка. После этого разрабатывается каждая подсистема. Таким образом, разработка ведётся по-прежнему в рамках принятой модели, но не для всего проекта, а для каждой отдельной подсистемы. Благодаря раннему вводу в эксплуатацию, инкрементная модель даёт возможность быстрее и лучше осознать потребности пользователя и внести корректировки в требования к ещё не разработанным подсистемам.

Поскольку каждая подсистема относительно простая, планировать ресурсы заметно проще. Хуже с достижением единого стиля, ввиду изолированности разработки. Общую идеологическую направленность проекта и интерфейсы между подсистемами удаётся реализовать успешно, но унифицировать характер взаимодействия с пользователем довольно сложно.

Своеобразный современный пример инкрементного подхода при разработке больших прикладных систем – это системы управления предприятиями (ERP II), родственные АСУП. Они объединяют системы ERP, CRM, SCM и EAM. Здесь

- ERP (Enterprise Resource Planning) – управление ресурсами предприятий: набор приложений, которые поддерживают все основные аспекты управления производством.
- CRM (Customer Relationship Management) – управление взаимоотношениями с заказчиком: набор приложений для сбора, хранения и обработки информации о клиенте.
- SCM (Supply Chain Management) – управление цепью поставок: координация, планирование и управление процессами снабжения, производства, складирования и доставки товаров или услуг потребителям.
- EAM (Enterprise Asset Management) – управление имуществом предприятий.

Возвращаясь к истории, заметим, что АСУП, о которой шла речь ранее, планировалось разрабатывать по каскадной модели, но срок разработки не устраивал заказчика. И модель модифицировали: после проектирования архитектуры провели декомпозицию на подсистемы, которая обеспечила их минимальную взаимную зависимость. Дальнейшая разработка велась отдельными подсистемами:

- техническая подготовка производства;
- оперативно-календарное планирование;
- технико-экономическое планирование;
- материально-техническое обеспечение;
- бухгалтерский учёт;
- управление организацией труда и заработной платой;
- система метрологического обеспечения;
- система управления качеством;
- управление стандартизацией;
- управление кадрами;
- управление оборудованием;
- управление технологическими процессами.

По сути, была изобретена инкрементная модель, о которой, как о таковой, тогда никто не знал. Она позволила разработать систему, которая полноценно работала задолго до ERP II, стандарт которой был предложен компанией Gartner в 1999 г.

Второй пример – автоматизация работы крупной многопрофильной больницы. Кака было принято, разработчики решили создать АСУ. После анализа была определёна архитектура системы и проведена её декомпозиция. Предполагалась общая техническая, информационная и технологическая база, был продуман регламент работ, определены общие для всех подсистем процедуры. Однако с разработкой дело было хуже: предметная область была незнакомой, и приоритет работ определить было трудно. Начали с подсистемы для одного из лечебных корпусов – не очень получилось. А вот успешная реализация более понятных, «периферийных», подсистем вселила в разработчиков уверенность, и они, после успешной работы с приёмным отделением, подступились к лечебным корпусам. В конце концов, система практически полностью стала покрывать информационные потребности больницы. Мы видим типичный инкрементный подход, который участники разработки изобрели в отчаян-

ной ситуации. Приоритет вначале был один: давайте сделаем, что можно. И только на основании опыта стало возможным делать, что нужно.

7. ИНТЕРНЕТ-ПРОЕКТЫ

Появление интернета стало стимулом к развитию программных изделий, использующих возможности глобальной сети. Большинство интернет-проектов (ИП) в конце 1990-х годов были статичными web-страницами, но развивались они удивительно быстро. Уже в начале 2000-х годов они занимали весьма значительную долю рынка программных изделий. Их принципиальное отличие – работа на широкую (неопределённую) аудиторию, тогда как у обычных проектов были конкретные заказчики, которые платили за функциональность, надёжность и удобство работы с продуктом. Разработчики фактически не имели конкурентов и работали спокойно, добиваясь заданных параметров качества.

Доступность интернет-приложений для множества пользователей привела к конкуренции на рынке программных продуктов, к стремлению опередить конкурента во что бы то ни стало. Приоритетом разработчиков стало не качество, а срок выпуска. Если большие системы разрабатывались годами, то теперь такая роскошь стала недоступной, и семилетние проекты превратились в семимесячные [10].

Как только «менеджеры» почувствовали выгоду коротких сроков разработки, они забыли о методах оценки времени проекта [2] и стали ужимать время: якобы, за деньги можно сделать всё. Однако сокращённые сроки провоцируют отказ от документации, планирования, анализа и конструктивной оценки проделанной работы. Да и юное поколение программистов – выпускников вузов, нередко лишь с начальными знаниями в голове, энергично подхватывает новые веяния.

Пока проекты были относительно небольшими, такой подход ещё проходил. Да, ИП были достаточно дрянными, но что-то нужное они пользователю давали, стоили недорого и даже в случае ошибок не наносили существенного вреда. И для них можно использовать лёгкие методологии.

Но очень скоро пользователи и руководители проектов становятся более требовательными: ИП рассматриваются как жизненно важные. Масштаб их стал увеличиваться, а требования к качеству расти.

Раз так, нельзя ли воспользоваться известной и надёжной водопадной моделью? Ответ очевиден – нет, время не позволяет. Но что делать, если для сложных проектов традиционные этапы планирования, анализа, разработки просто необходимы? Тогда приходится ориентироваться на версионные модели с дополнительными функциональностями и искать баланс между строгими методами и анархией agile.

Следует сказать несколько слов об оценке времени разработки ИП. Большинство их начиналось без чётких требований. Отсюда (кроме, конечно, гонки) и слабые методы оценки проектов. На самом деле, оценка ИП не отличаются от обычных оценок, только ИП более рискованные.

Источник многих неприятностей – изменчивые требования. В ИП ситуация хуже: пользователь порой не знает, что хочет, нередко происходит смещение требований. Кроме того, источник риска в ИП связан с доступом из любой точки. Нужно особо учитывать производительность, надёжность, защиту и конфиденциальность. Поэтому вместо обещания сделать что-то крупное и неизвестное с большими факторами риска, лучше договориться о серии небольших вещей с меньшим фактором риска. Следовательно, нужно разбивать проект на относительно независимые части. И вот мы на новом уровне пришли к известной инкрементной модели!

Итак, что выбрать Waterfall или Agile? Большие серьёзные проекты (например, в оборонке, медицине, банковской области) требуют высокой надёжности, и, значит, высокой производственной дисциплины, в отличие от небольших, но широко распространённых интернет-проектов. В первом случае используем Waterfall, во втором Agile. Но для больших ИП оба этих метода в чистом виде не годятся.

8. ПРИГОДНОСТЬ КРАЙНИХ МЕТОДОВ

Реально оба крайних варианта методологий выдержали проверку временем и могут успешно применяться при создании программных систем, однако на практике существует противостояние Agile и Waterfall.

Рассмотрим статистику от Standish Group [3] (табл. 3). Напрашивается вывод: гибкие методологии опережают «классические» каскадные, от которых, видимо, нужно отказаться. Однако здесь совершенно не учитывается качество проекта и цена его последующего сопровождения, что для многих заказчиков имеет принципиальное значение! Опять же, кто может назвать ИП успешным или неудачным? Кто, кроме владельца, основная заинтересованная сторона? Каковы критерии успеха?

Таблица 3

Успешность программных проектов в зависимости от выбранной методологии. Данные Chaos Report (в процентах)

Размер проекта	Методология	Успешные проекты	Неудачные проекты	Провалившиеся проекты
Все проекты	Agile	39	52	9
	Waterfall	11	60	29
Большой	Agile	18	59	23
	Waterfall	3	55	42
Средний	Agile	27	62	11
	Waterfall	7	68	25
Маленький	Agile	58	38	4
	Waterfall	44	45	11

В итоге компания должна решить и выбрать, по какой методологии она работает в той или иной нише. Лёгкие методологии предпочтительнее для небольших компаний, работающих в условиях сжатых сроков, быстро меняющихся требований

и необходимости обеспечения достаточного качества. Область их применения – небольшие и, в крайнем случае, средние проекты. Если необходимы стабильность и качество продукта, а временные и финансовые затраты вторичны, стоит использовать «классические» каскадные методологии.

9. МОДИФИКАЦИЯ ИЛИ СМЕСЬ?

Из-за того, что ни одно «крыло» не может претендовать на универсальное решение, компания зачастую изобретает гибрид, пытаясь работать по Agilefall (Agile + Waterfall). Последний вариант, когда часть компании, например, разработчики внешних систем, работают по Agile, а внутренние процессы разрабатываются по Waterfall, самый худший. Зачастую это происходит из-за веяния моды, компании стремятся привлечь востребованных специалистов на модные тренды, не проведя внутреннюю перестройку процессов. В итоге страдают и выгорают первыми именно высококвалифицированные сотрудники, которые, подгоняемые менеджментом, стремятся работать по гибким методологиям, а их ожидания разбиваются о внутренние каскадные процессы [11].

В литературе описан подобный случай многофакторного выбора, и результат все знают.

Агафья Тихоновна: «*Если бы губы Никанора Ивановича да приставить к носу Ивана Кузьмича, да взять сколько-нибудь развязности, какая у Балтазара Балтазаровича, да, пожалуй, прибавить к этому ещё дородности Ивана Павловича – я бы тогда тотчас бы решилась. А теперь поди подумай!*» (*Н.В.Гоголь, «Женитьба»*).

Какой выход из данной ситуации? С точки зрения авторов, не стоит изобретать Agilefall, в рамках которого не особо комфортно всем.

Предлагается следующий подход. На этапе проектирования и создания сложной информационной системы с длительностью проекта более 6 месяцев использовать классическую методологию и разработать её первую версию с надлежащим качеством, внятной документацией, проработанной архитектурой решения и инфраструктурой проекта. В таком случае все службы компании работают по единой водопадной методологии, у участников проекта не возникает диссонанса, так как все находятся в одном информационном поле [12], разработчики, аналитики, тестировщики следуют четкому плану проекта. Да, в таком случае будет потеря времени, но она несравнимо мала с ценой возможной ошибки. Когда «костяк» проекта и команды создан, следует постепенно перейти на гибкие методологии. При таком подходе выигрывают все: и команда проекта, и заказчик, который получает систему с необходимым качеством. С переходом на гибкие методологии команды разделяются и расширяются, происходит распараллеливание задач. Выделяются команды, которые будут выполнять полный цикл разработки частей системы и её дальнейшие развитие.

Примером такого подхода может служить создание системы дистанционного банковского обслуживания (ДБО). В этой разработке участвовали команды, отвечающие за развитие следующих подсистем:

- интернет банка;
- мобильное приложение;
- АРМ сотрудника банка в отделении;
- платежный функционал;
- виртуальная продуктовая линейка банка;
- команда чатов и пушей;
- продуктовые команды (например, вклады, кредитные карты);
- прочие подсистемы.

Для небольших проектов, где цена ошибки невысока, предполагается сразу использовать гибкие методологии с учётом того, что все процессы в компании «живут» по гибким методологиям.

10. ВЫВОДЫ

Для сложных программных проектов разумно использовать комбинацию водопада для первого цикла разработки системы, её «костяка», с дальнейшим переходом на проектную работу в среде гибких методологий. Но крайне не рекомендуется с самого начала, особенно если в компании основные процессы живут по водопадной методологии, использовать гибрид Agilefall. От него страдают как разработчики, так и заказчики.

Литература

1. *Гласс Р.* Программирование и конфликты 2.0. Теория и практика программной инженерии. – СПб: Символ–Плюс, 2010. – 240 с.
2. *Лукин В.Н., Бахиркин М. В.* Динамическая оценка времени разработки программных систем. Монография. – М.: Изд-во МАИ, 2019. – 160 с.
3. Standish Group, Homepage, https://www.standishgroup.com/, last accessed 2021/08/01.
4. Стандарты: http://www.gost.ru/, last accessed 2021/07/01.
5. *Бек К.* Экстремальное программирование: разработка через тестирование. – СПб.: Питер, 2017. – 224 с.
6. Ryland Leyton: The Agile Business Analyst: Moving from Waterfall to Agile 1st Edition (2015).
7. Project Management Institute.: Agile Practice Guide 1st Edition (2017).
8. Tom DeMarco, Timothy Lister: Waltzing with Bears: Managing Risk on Software Projects 1st Edition, Kindle Edition (2013).
9. *Демарко Т.* Deadline. Роман об управлении проектами. – М.: Вершина, 2008. – 288 с.
10. *Йордон Э.* Управление Интернет-проектами. М.: «Лори», 2021. – 344 с.
11. *Брукс Ф.* Мифический человеко-месяц или как создаются программные системы. – Пер. с англ. – СПб.: Символ-Плюс, 2012. – 304 с.
12. *Tom DeMarco, Timothy Lister.* Peopleware: Productive Projects and Teams 3rd Edition, Kindle Edition (2013).

Modern Realities of Software Project Management

Michail V. Bakhirkin*
Moscow Aviation Institute (National Research University)
Joint-stock company «Post Bank», Moscow, Russia
ORCID: https://orcid.org/0000-0002-9077-4426
e-mail: bakhirkin@mail.ru

Vladimir N. Lukin**
Moscow Aviation Institute (National Research University)
State University of Psychology & Education, Moscow, Russia
ORCID: https://orcid.org/0000-0001-8906-2686
e-mail: lukinvn@list.ru

The need to maintain the workflow in conditions of forced disconnection has increased the need for software products. But it turned out that the quality of many of them is lower than expected. One of the reasons is that users with different quality criteria began to work with them. Another reason is that the products have objectively become unacceptably low quality, largely due to errors in the management of the project. The scale and significance of a modern software project require its implementation according to one of the classical models: it will ensure quality. However, due to the desire to get ahead of competitors, flexible methodologies are used, reducing deadlines and losing quality. And the larger the project, the higher the risk of quality loss and the higher the cost of losses. How to combine the benefits of both approaches without getting their disadvantages?

Keywords: large software project, development methodology, Waterfall, Agile, Internet project.

For citation:
Bakhirkin M.V., Lukin V.N. Modern Realities of Software Project Management. *Modelirovanie i analiz dannykh = Modelling and Data Analysis,* 2021. Vol. 11, no. 4, pp. 72–86. DOI: https://doi.org/10.17759/mda.2021110406 (In Russ.,abstr. in Engl.).

References

1. Glass R. Programming and conflicts 2.0. Theory and practice of software engineering. – St. Petersburg: Symbol-Plus, 2010. – 240 p.
2. Lukin V.N., Bakhirkin M.V. Dynamic estimation of software systems development time. Monograph. – M.: Publishing House of MAI, 2019. – 160 p.
3. Standish Group, Home page, URL: https://www.standishgroup.com (last accessed 2021/08/01).

***Michail V. Bakhirkin,** Candidate of Technical Sciences, Moscow Aviation Institute (National Research University), Joint-stock company «Post Bank», Moscow, Russia, ORCID: https://orcid.org/0000-0002-9077-4426, e-mail: bakhirkin@mail.ru

****Vladimir N. Lukin,** Candidate of Physical and Mathematical Sciences, docent, Moscow Aviation Institute (National Research University), Professor, State University of Psychology & Education, Moscow, Russia, ORCID: https://orcid.org/0000-0001-8906-2686, e-mail: lukinvn@list.ru

4. Standards: URL: http://www.gost.ru (last accessed on 2021/07/01).
5. Beck K. Extreme programming: development through testing. – Sp.: Pi-ter, 2017. – 224 p.
6. Ryland Leyton: agile business analyst: moving from waterfall to an agile 1st edition (2015).
7. The project management Institute.: Guide to flexible practice 1st edition (2017).
8. Tom DeMarco, Timothy Lister: Waltzing with bears: Managing risk in software projects 1st edition, Kindle edition (2013).
9. DeMarco T. deadline. A novel about project management. – Moscow: Vershina, 2008. – 288 p.
10. Yordon E. Internet project management. M.: “Lori”, 2021. – 344 p.
11. Brooks F. The Mythical man-month or how software systems are created. – Translated from English – St. Petersburg: Symbol-Plus, 2012. – 304 p.
12. Tom DeMarco, Timothy Lister. Software for People: Productive Projects and Teams 3rd Edition, Kindle Edition (2013).