Сравнительный анализ моделей жизненного цикла программного обеспечения с различными способами организации потоков работ на основе результатов имитационного моделирования[1]

М. С. Голосовский

Государственный научно-исследовательский испытательный институт военной медицины Минобороны России,
111250, Москва, 1-й Краснокурсантский проезд, 7

e-mail: *golosovskiy@yandex.ru*

Аннотация. В работе представлены результаты сравнительного анализа различных моделей жизненного цикла и методологий разработки программного обеспечения, выполненного на основе разработанной имитационной модели жизненного цикла программного обеспечения для моделей жизненного цикла программного обеспечения с различным способом организации потоков работ. Показано, что наиболее эффективной с точки зрения выполнения основной части проекта является последовательная методология разработки программного обеспечения. Однако вследствие невозможности выполнения только заранее определенной части проекта наиболее часто применяют конвейерные и итеративные модели жизненного цикла программного обеспечения, которые уступают последовательной модели по критерию срока выполнения основной части проекта, но более эффективны по критерию оперативности обработки запросов на изменение программного обеспечения.
Ключевые слова: жизненный цикл программного обеспечения, разработка программного обеспечения, модель жизненного цикла программного обеспечения.

1. Введение

С момента запуска первой ЭВМ архитектуры вычислительных машин и процессы разработки программного обеспечения претерпели значительные изменения. Во многом благодаря тому, что основные процессы разработки программного обеспечения (ПО) схожи с процессами разработки в других инженерных областях и состоят из следующих стадий: проектирование, разработка (создание образца изделия), испытания (тестирование), серийное производство, сопровождение. Процесс, начинающийся с момента принятия решения о разработке программного обеспече-

[1] Работа выполнена при финансовой поддержке РФФИ, проект № 16-06-00486 «Исследование методологических проблем инновационного развития информационных технологий организации»

ния и завершающийся его изъятием из эксплуатации, носит название жизненного цикла программного обеспечения. За время развития индустрии разработки программного обеспечения модели жизненного цикла также претерпели значительные изменения, в частности появилось большое количество «легковесных» или Agile моделей жизненного цикла, в литературе называемых «методологии», с возможностью «легкого» внесения изменений в структуру разрабатываемых программ. В дальнейшем в статье сделано допущение и слова «Методология разработки ПО» и «Модель жизненного цикла ПО» будут являться синонимами. Несмотря на различия, все модели содержат следующие общие фазы жизненного цикла:

- сбор и анализ требований к ПО;
- проектирование архитектуры;
- детальное проектирование;
- реализация;
- тестирование и интеграция;
- сопровождение;
- вывод из эксплуатации.

Хронология развития моделей жизненного цикла приведена на рис. 1. Одной из первых моделей разработки программного обеспечения является «водопадная» (waterfall) или известная под названием «каскадная» модель жизненного цикла. Он также носит название классического жизненного цикла разработки ПО отчасти в связи с тем, что эта модель повторяет последовательность действий при разработке инженерных решений в других областях науки и техники.

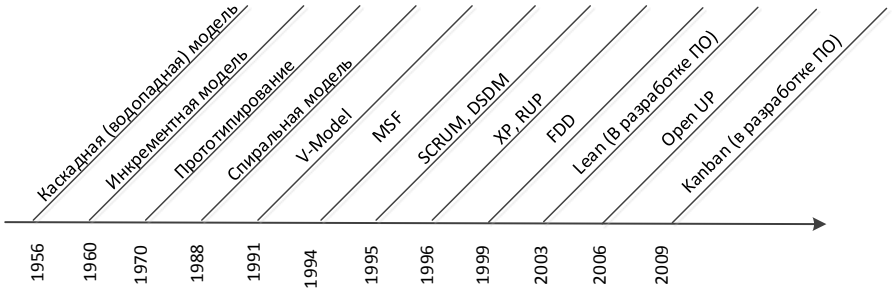

Рисунок 1. Последовательность появления моделей жизненного цикла ПО

По классификации, представленной в своде знаний по программной инженерии SWEBOK [1], третья версия которого вышла в 2013 г., модели жизненного цикла разделяются на две категории, линейные и итерационные модели. Исторически первой в системе разработки была водопадная модель, но из-за того, что в от-

сутствие полноты требований возникала необходимость в уточнении требований и внесении дополнительных изменений в проект на стадии реализации или тестирования, были предложены различные итеративные модели, которые, по сути, отличаются только количеством итераций, их размером, и действиями, выполняемыми при подготовке и завершении каждой итерации. В связи с этим был выделен класс гибких методологий разработки ПО, в которых размер итераций был сравнительно маленьким (занимал порядка недели, а в некоторых — до одного дня).

Подобные подходы применяются в основном при разработке программного обеспечения, в условиях изменяющихся требований и высокой неопределенности предметной области применяются итеративные и относятся к семейству *Agile* [2–17], наиболее известные: экстремальное программирование (XP), Scrum, *Kanban, OpenUP, FDD*. Основной идеей, лежащей в основе этих процессов, является частый выпуск версий работающего ПО, пусть и реализующего не весь функционал, заложенный на этапе постановки задачи и анализа требований. ГОСТ Р ИСО/МЭК12207–2010 дает определение ЖЦ ПО и общее описание основных процессов жизненного цикла. Но при этом стандартом не предписывается выбор конкретной методологии разработки ПО. В связи с этим перед разработчиками ПО встает задача выбора наиболее подходящей модели жизненного цикла ПО в условиях существенно различных первоначальных требований к поставке разрабатываемого ПО и организационных процессов компании-разработчика ПО [18–22].

2. Синтез имитационной модели жизненного цикла ПО

Для исследования моделей жизненного цикла ПО введем понятие задачи как некоторого объекта, поступающего на вход процесса, обрабатываемого/изменяемого процессом и передающегося на выход процесса. В качестве задач выступают следующие артефакты ЖЦ ПО.

- Высокоуровневые требования (high level requirement) — требования, описывающие крупные функции системы в общей форме. Соответствуют бизнес-требованиям в классификации К. Вигерса [23]. Служат для высокоуровневого описания реализуемых крупных частей или выполняемых функций системы, как правило, с точки зрения бизнес-пользователей. Эти требования всегда детализируются и уточняются детальными функциональными и нефункциональными требованиями.
- Детальные требования (detailed requirement) — функциональные и нефункциональные требования, детализирующие высокоуровневые требования.
- Задачи на разработку (task) — постановки задачи разработчика, на основании которых происходит написание кода.

– Тестовые сценарии (test case) — сценарии проверки соответствия разработанной системы предъявленным требованиям.
– Дефекты (defect) — несоответствие разработанного ПО предъявленным требованиям.
– Инциденты (incident) — сбой, возникший в процессе эксплуатации ПО.

Перечисленные во введении модели жизненного цикла по последовательности выполнения процессов ЖЦ и передачи задач от одного процесса другому можно разделить на следующие три основные категории:

1. *Последовательная* — первоначальный набор задач не разбивается на части. На выполнение в следующий процесс задачи переходят только после обработки всех задач в текущем процессе. Изменения в состав проекта принимаются только после передачи программного продукта в эксплуатацию. Понятие производственного цикла отсутствует. Могут выделяться этапы работ, в рамках которых дополнительно реализуются новые высокоуровневые требования.

2. *Циклическая* (*итеративная*) — первоначальный набор задач разбивается на части, каждая часть выполняется в отдельном производственном цикле, включающем в себя основные производственные процессы жизненного цикла ПО.

3. *Конвейерная* — задачи выполняются по мере поступления. После завершения задачи на одном этапе задача передается на следующий этап в случае наличия ресурсов для выполнения задачи на следующем этапе. В случае ограниченного количества ресурсов порядок выполнения задач регулируется приоритетами. Производственные циклы отсутствуют (фактически для каждой задачи формируется свой производственный цикл, который может идти параллельно с другими).

Сравнение выделенных категорий модели жизненного цикла ПО приведено в табл. 1.

Таблица 1. Сравнение категорий модели жизненного цикла ПО

Категория ЖЦ	*Завершение всех задач проекта на одном этапе перед началом следующего*	*Политика передачи изменений на исполнение*	*Наличие производственных циклов*	*Соответствующие методологии/ Модели разработки ПО*
Последовательная	Ттребуется	После завершения основной части проекта	Могут выделяться крупные этапы	Водопадная модель
Циклическая	Не требуется	После завершения текущего производственного цикла	Вся разработка ведется производственными циклами	Scrum, OpenUP, RUP, RAD, итеративная модель, спиральная модель.
Конвейерная	Не требуется	В любое время выполнения проекта	Производственные циклы отсутствуют	XP, Kanban, Lean, FDD

Для оценки выделенных категорий моделей ЖЦ ПО с точки зрения фактора времени передачи в эксплуатацию работающего ПО можно выделить следующие критерии:

- время с момента начала проекта до передачи в эксплуатацию основной части проекта;
- время с момента поступления до передачи в эксплуатацию выполненных высокоуровневых запросов на изменение (ЗНИ). В литературе [7, 15, 24] для обозначения времени с момента поступления требования до передачи его в эксплуатацию используется термин lead-time, или time-to-market;
- время с момента поступления до передачи в эксплуатацию выполненных детальных ЗНИ;
- время выполнения проекта в целом (завершения основной части проекта и утвержденных запросов на изменение).

Сравнение моделей жизненного цикла на основе реальных проектов по указанным критериям затруднительно в связи с тем, что для получения валидных результатов требуется провести сравнение моделей ЖЦ на одинаковых проектах по составу работ и по составу исполнителей (время, затраченное исполнителем на выполнение каждой отдельной работы должно быть одинаковым для разных моделей ЖЦ). В связи с этим сравнение произведено с использованием имитационной модели. Схема имитационной модели жизненного цикла приведена на рис. 2 и талб. 2.Каждый блок модели может быть одного из перечисленных ниже типов:

- процессор: реализует обработку объектов по заданному алгоритму;
- генератор: используется для генерации объектов в заданный момент времени;
- очередь: используется для хранения объектов до момента освобождения ресурсов в блоке типа процессор для обработки объектов очереди.

Один час времени модели соответствует одному циклу — обходу всех блоков модели в соответствии с их номерами и увеличении текущего времени блока на один час. С использованием этого механизма можно имитировать выполнение различных задач.

Особенностью предложенного подхода к имитационному моделированию является то, что объекты, имитирующие задачи, обрабатываемые в модели, генерируются заранее и перед запуском модели сгенерированный набор задач передается на обработку в модель. Оценки времени выполнения задачи и фактические временные затраты на выполнение одной задачи, связи между задачами также генерируются до выполнения модели.

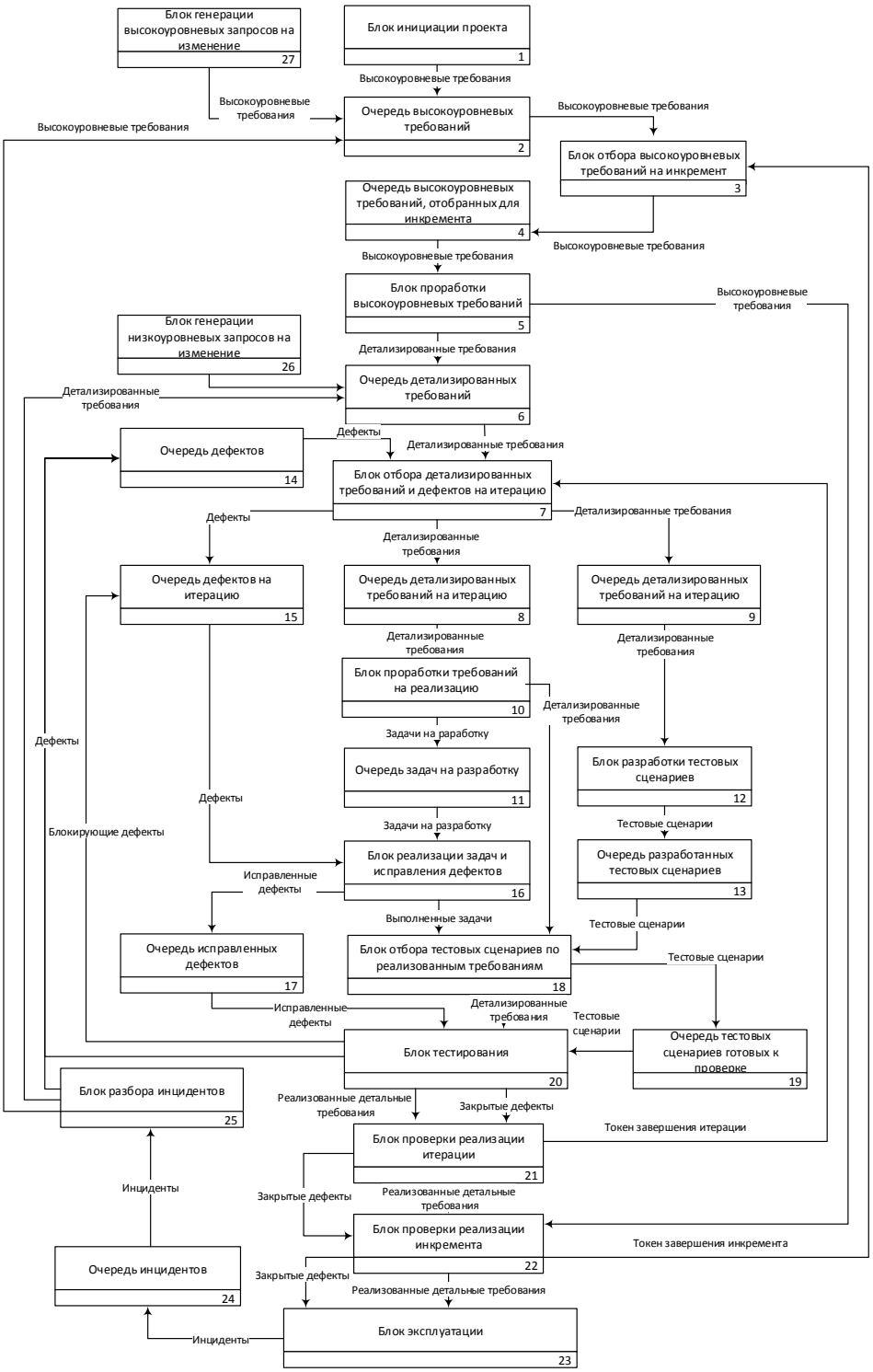

Рисунок 1. Имитационная модель жизненного цикла ПО

Таблица 2. Описание блоков модели типа процессор

№ п/п	*Номер на рис. 2*	*Название блока*	*Краткое описание*
1	3	Блок отбора высокоуровневых требований на инкремент	Производит отбор высокоуровневых требований на инкремент по трудоемкости из очереди. Отбор на следующий инкремент производится после того, как реализованы все высокоуровневые требования предыдущего инкремента
2	5	Блок проработки высокоуровневых требований	Имитирует процессы сбора и анализа требований к ПО и проработки архитектуры. Для каждого полученного на вход высокоуровневого требования создает заданное на этапе генерации состава работ число детализированных требований. Число параллельно обрабатываемых высокоуровневых требований равно числу бизнес-аналитиков
3	7	Блок отбора детализированных требований и дефектов на итерацию	Производит отбор высокоуровневых требований на инкремент. Отбор производится на основе сгенерированной оценки трудоемкости требований и дефектов
4	10	Блок проработки требований на разработку	Имитирует процесс детального проектирования. Создает для каждого полученного на вход детализированного требования заданное на этапе генерации состава работ число задач на разработку. Число параллельно обрабатываемых детализированных требований равно числу системных аналитиков
5	12	Блок разработки тестовых сценариев	Имитирует процесс разработки тестовой модели и тестовых сценариев. Создает для каждого полученного на вход детализированного требования заданное на этапе генерации состава работ число тестовых сценариев
6	16	Блок реализации задач и исправления дефектов	Имитирует процесс реализации ПО. Число параллельно обрабатываемых задач и дефектов равно числу разработчиков ПО
7	18	Блок отбора тестовых сценариев по реализованным требованиям	Для детализированных требований, у которых реализованы все задачи отбирает тестовые сценарии и передает их в очередь на тестирование
8	20	Блок тестирования	Имитирует процесс тестирования ПО. Для каждого полученного на вход тестового сценария создает заданное на этапе генерации состава работ число детализированных требований. Для каждого полученного на вход дефекта производит проверку, будет он переоткрыт или нет. Число параллельно обрабатываемых тестовых сценариев и дефектов равно числу специалистов по тестированию
9	21	Блок проверки реализации итерации	Производит проверку, что все отобранные на итерацию дефекты и требования закрыты. В этом случае отправляет токен на набор новых объектов на итерацию в блоке 7
10	22	Блок проверки реализации инкремента	Проводит проверку, что все высокоуровневые требования, отобранные на инкремент, закрыты. Проводит регрессионное тестирование перед передачей реализованных требований в эксплуатацию. Отправляет токен на набор новых объектов на инкремент в блоке 3
11	24	Блок эксплуатации	Генерирует поток инцидентов по переданным в эксплуатацию требованиям
12	26	Блок разбора инцидентов	Производит проверку, является переданный инцидент дефектом или нет. Число параллельно обрабатываемых инцидентов равно числу специалистов по эксплуатации

Указанный подход позволяет провести корректное сравнение выделенных категорий моделей жизненного цикла ПО с точки зрения эффективности моделей по выделенным критериям.

Предложенная имитационная модель является универсальной и позволяет моделировать все три категории моделей жизненного цикла. Но для последовательной и конвейерной моделей в имитационную модель вводятся следующие ограничения.

- Для последовательной модели жизненного цикла в первый инкремент и итерацию помещается весь состав работ основной части проекта. Внесение изменений не допускается. Очередь выходных объектов блокирует выдачу объектов до тех пор, пока процессор не обработает все объекты из входной очереди. Таким образом имитируется последовательное прохождение этапов работ проекта.
- Для конвейерной модели блоки отбора на итерацию и инкремент используются только в режиме дальнейшей передачи объектов. Завершение инкремента и итерации не отслеживается. В блоке 22 производится только регрессионное тестирование перед передачей реализованных требований в эксплуатацию.

3. Верификация моделей жизненного цикла ПО на основе реализованных проектов

Верификация моделей проведена посредством сравнения результатов моделирования с реальными проектами. Для чего из системы управления проектами были выгружены данные по 8 завершенным проектам, выполнявшимся по следующим моделям жизненного цикла: Водопадная (относящаяся к последовательным моделям), Scrum (относящаяся к циклическим моделям), Canban (относящаяся к конвейерным моделям). Длительность проектов, отобранных для сравнения, была выбрана не менее одного года и не более полутора лет, за исключением проектов, разработанных по водопадной модели. В связи с тем, что водопадная модель на практике применяется редко, для сравнения были отобраны только два проекта длительностью 4 и 6 месяцев. За длительность проекта берется промежуток времени, начинающийся с момента инициации проекта до перевода его на этап поддержки после реализации всех высокоуровневых и детальных требований и исправления критических дефектов, когда основные работы по разработке завершаются. Характеристики отобранных проектов приведены в табл. 3.

Для каждого отобранного проекта был сформирован на основе полученной статистической информации состав работ и участников для имитационной модели. Для каждой работы брались фактические трудозатраты. После чего было проведено имитационное моделирование с использованием жизненного цикла, соответствую-

щего моделируемым проектам. Результаты сравнения характеристик реальных проектов (РП) с полученными в результате имитационного моделирования (ИМ) приведены в табл. 4.

Таблица 3. Характеристики отобранных для сравнения проектов

№ проекта	*Тип проекта*	*Модель ЖЦ*	*Фактическая трудоемкость чел.-дни*	*Число высокоуровневых изменений*	*Число детализированных изменений*
1	Разработка мобильного приложения без серверной части	Последовательная	611	0	5
2	Разработка серверной с web-интерфейсом	Последовательная	982	1	8
3	Разработка системы с web-интерфейсом	scrum	3628	2	12
4	Разработка мобильного приложения с серверной частью	scrum	3465	1	14
5	Разработка системы с web-интерфейсом	scrum	2834	3	11
6	Разработка системы с web-интерфейсом	kanban	3254	2	10
7	Разработка системы с web-интерфейсом	kanban	4132	2	18
8	Разработка клиент-серверной системы с мобильным приложением и web-интерфейсом	kanban	2968	3	9

Максимальная ошибка моделирования на основании исторических данных по сравнению с результатами реального проекта для параметра выполнения основной части проекта не превышает 39%, для параметра выполнение всего проекта не превышает 50%, для параметра Lead time высокоуровневых ЗНИ не превышает 90%, для параметра Lead time детальных ЗНИ не превышает 40%. Разработанная модель не учитывает многих дополнительных факторов, влияющих на процесс разработки программного обеспечения, но в общем случае позволяет дать сравнительную оценку различных моделей ЖЦ ПО.

Таблица 4. Результаты сравнения характеристик реальных проектов (РП) с полученными в результате имитационного моделирования (ИМ)

№ Проекта	*Выполнение основной части проекта (дни)*		*Выполнение всего проекта (дни)*		*Lead time высокоуровневых ЗНИ (дни)*		*Lead time детальных ЗНИ (дни)*	
	РП	*ИМ*	*РП*	*ИМ*	*РП*	*ИМ*	*РП*	*ИМ*
1	56	68	82	123	0	0	36	64
2	91	121	133	179	62	112	62	79
3	247	198	274	401	163	310	89	78
4	234	270	312	353	207	331	88	95
5	219	259	257	290	140	157	56	71
6	268	211	289	342	126	238	63	45
7	305	186	316	264	194	253	34	41
8	214	263	285	340	128	216	68	52

4. Результаты сравнения моделей жизненного цикла ПО

Сравнение моделей жизненного цикла проведено по следующей схеме.

- Сгенерирован состав работ на основе генератора случайных чисел.
- Проведено имитационное моделирование с размером команды от 20 до 40 человек. Выбиралось равное количество людей на каждый тип деятельности (бизнес-аналитика, системная аналитика, разработка, тестирование, разбор инцидентов).

Сроки выполнения задач для критериев: выполнение основной части проекта, выполнение всего проекта, lead time высокоуровневых ЗНИ, lead time детальных ЗНИ приведены на рис. 3. Согласно полученным результатам наибольшие затраты по времени у итеративной модели жизненного цикла по всем показателям за исключением lead time детальных ЗНИ. В этом показателе наибольший результат у модели ЖЦ последовательного типа.

У итеративной модели жизненного цикла в отличие от других моделей жизненного цикла существует зависимость срока выполнения по показателям в зависимости от размера итерации.

Зависимости времени выполнения по выделенным критериям от размеров итерации и инкремента приведены на рис. 4. Общая зависимость: время выполнения основной части проекта и выполнение всего проекта с увеличением инкремента увеличивается.

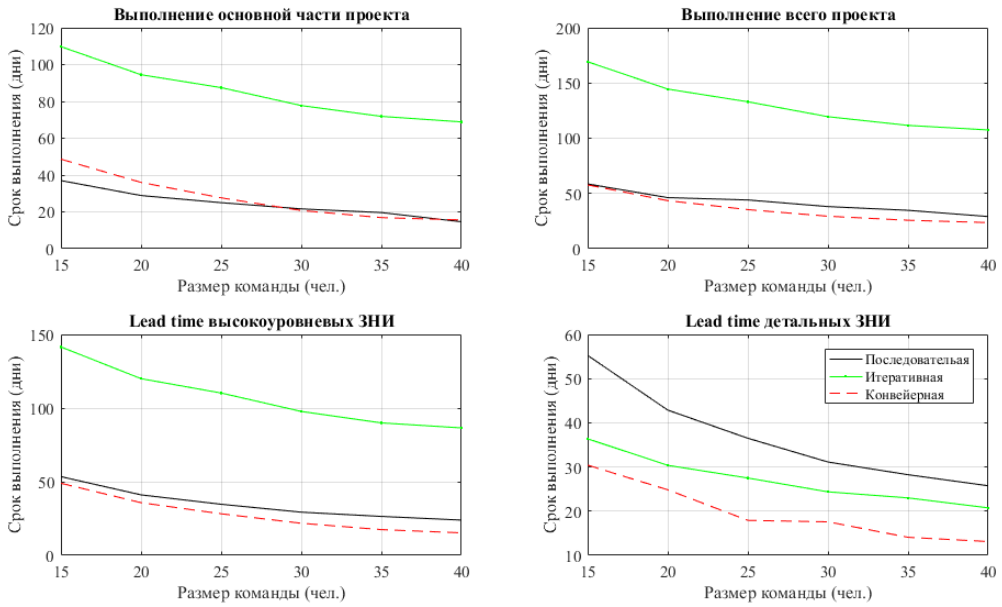

Рисунок 3. Зависимость сроков выполнения от размера команды для различных моделей жизненного цикла ПО

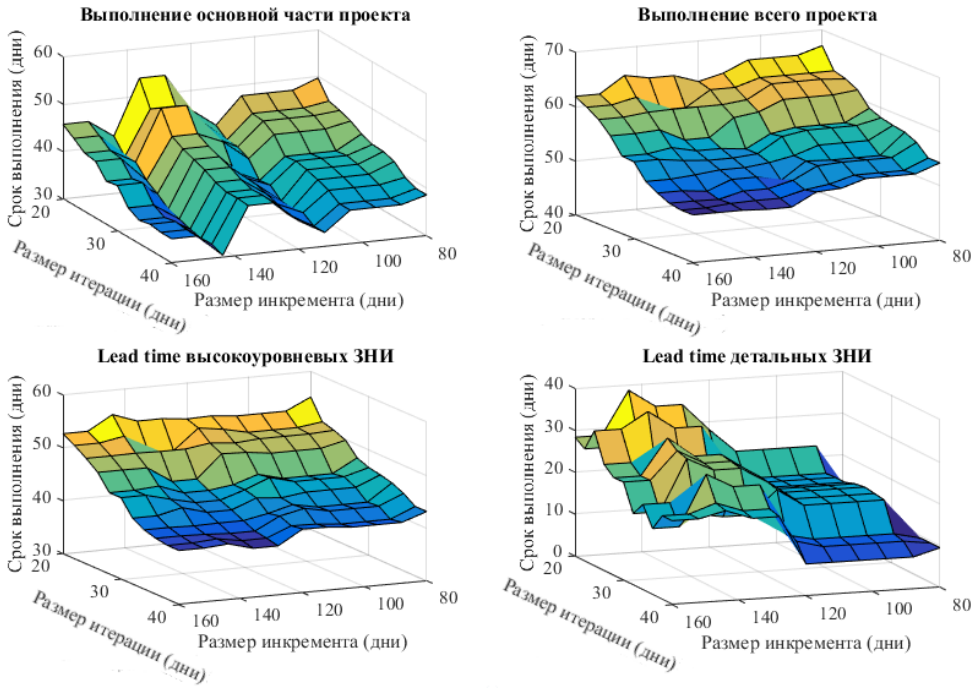

Рисунок 4. Зависимость срока выполнения от размера инкремента и размера итерации

Заключение

Проведенное имитационное моделирование показывает, что наиболее эффективной с точки зрения выполнения основной части проекта является последовательная методология. На практике выполнение только заранее определенной части проекта является сложным и зачастую невозможным [17, 19, 22, 24]. В связи с этим наибольшее развитие получают конвейерные и итеративные модели ЖЦ, которые хотя и менее эффективны по сравнению с последовательной моделью по критерию срока выполнения основной части проекта, но более эффективны в части обработки запросов на изменение.

Литература

[1] Guide to the Software Engineering: body of knowledge, version 3.0 [Электронный ресурс]. URL: http://www4.ncsu.edu/~tjmenzie/cs510/pdf/SWEBOKv3.pdf

[2] *Кон М.* Scrum: гибкая разработка программного обеспечения. — М. : Вильямс, 2011.

[3] *Ларман К., Базили В.* Итеративная и инкрементальная разработка: краткая история // *Открытые системы*. 2003. № 9. С. 43–53.

[4] Agile-манифест разработки программного обеспечения [Электронный ресурс] URL: http://agilemanifesto.org/iso/ru/.

[5] *Beck K.* Embracing Change with Extreme Programming // *Computer*. 1999. Vol. 32. No. 10. P. 70–77.

[6] *Beck K.* Test Driven Development by Example. Addison-Wesley Professional, 2002.

[7] *Brooks F.* No Silver Bullet — Essence and Accidents of Software Engineering // *Computer*. 1987. Vol. 20. No. 4. P. 10–19.

[8] *Cockburn L.* Using Both Incremental and Iterative Development. — // STSC CrossTalk (USAF Software Technology Support Center)/ 2008. Vol. 21. No. 5. P. 27–30.

[9] *Larman C.* Agile and Iterative Development: a Manager's Guide. — Addison-Wesley Professional, 2004.

[10] *Сафронов В. В., Федорец О. Н.* Метод построения эффективных моделей разработки программного обеспечения // *Информационные технологии*, 2010. № 1. С. 34–40

[11] *Скопин И. Н.* Модели жизненного цикла программного обеспечения // Новосибирская школа программирования. Перекличка времен. — Новосибирск : Изд. Института систем информатики им. А. П. Ершова СО РАН, 2004. С. 120–173.

[12] *Бахтизин В. В., Глухова Л. А.* Выбор модели жизненного цикла разработки программных средств и систем на основе сводной таблицы критериев классификации проекта // *Доклады Белорусского государственного университета информатики и радиоэлектроники*. 2005. № 1. С. 110–113.

[13] *Скрябин А. М., Кардаш Д. И.* Жизненный цикл композиционно-адаптируемого программного обеспечения // *Аспирант и соискатель*. 2008. № 2. С. 171–174.

[14] *Чумакова Т. Я., Цыганенко С. М.* Международные стандарты и жизненные циклы программного обеспечения // *Математические машины и системы*. 2009. Т. 1. № 3. С. 144–150.

[15] *Голосовский М. С.* Модель жизненного цикла разработки программного обеспечения в рамках научно-исследовательских работ // *Автоматизация. Современные технологии.* 2014. № 1. С. 43–46.

[16] *Богомолов А. В., Майстров А. И.* Технология анализа системных причинно-следственных связей на основе диаграмм Исикавы // Системный анализ в медицине (САМ 2014) Материалы VIII международной научной конференции, 2014. С. 13–16.

[17] *Голосовский М. С.* Информационно-логическая модель процесса разработки программного обеспечения // Программные системы и вычислительные методы. 2015. № 1. С. 59–68.

[18] *Шпудейко С. А., Богомолов А. В.* Методологические основы организации немонотонных процессов обучения сложным видам деятельности на основе теории трансформационного обучения // *Информационные технологии*. 2006. № 3. С. 74–79.

[19] *Голосовский М. С.* Модель оценивания погрешностей прогнозирования сроков разработки программного обеспечения // *Программные системы и вычислительные методы*. 2015. № 3. С. 311–322.

[20] *Ларкин Е. В., Богомолов А. В., Привалов А. Н.* Методика оценивания временных интервалов между транзакциями в алгоритмах сжатия речевых сообщений // *Научно-техническая информация. Серия 2: Информационные процессы и системы*. 2017. № 9. С. 23–28.

[21] *Голосовский М. С., Есев А. А., Богомолов А. В.* Комплекс автоматизированной экспертизы технического уровня сложной системы // Свидетельство о государственной регистрации программы для ЭВМ № 2014612330. Опубл. 20.03.2014. 1 с.

[22] *Голосовский М. С.* Модель расчета оценок трудоемкости и срока разработки информационных систем на начальном этапе жизненного цикла проекта // *Программная инженерия*. 2016. Т. 7. № 10. С. 446–455.

[23] *Вигерс К., Битти Д.* Разработка требований к программному обеспечению. — Русская Редакция, 2016.

[24] *Брукс Ф.* Мифический человеко-месяц, или как создаются программные системы. — Символ-Плюс, 2010.

***Автор*:**

Михаил Сергеевич Голосовский — младший научный сотрудник 24 отдела, Государственный научно-исследовательский испытательный институт военной медицины Минобороны России

Comparative analysis of software life cycle models with different ways of organizing workflows based on simulation results

M. S. Golosovsky

State Research Institute of Military Medicine of the Ministry of Defense of Russia, 1-st Krasnokursantsky proezd, 7, Moscow, Russia, 111250

e-mail: *golosovskiy@yandex.ru*

Abstract. The paper presents the results of a comparative analysis of different life cycle models and software development methodologies based on the developed simulation model of the software life cycle for software life cycle models with different ways of organizing workflows. It is shown that the most effective from the point of view of implementation of the main part of the project is a consistent methodology of software development. However, due to the impossibility of performing only a predetermined part of the project, the pipeline and iterative models of the software life cycle are most often used, which are inferior to the sequential model by the criterion of the deadline for the execution of the main part of the project, but are more efficient by the speed of processing requests for software changes.

Key words: software life cycle, software development, software life cycle model, sequential software life cycle model, software life cycle pipeline model, iterative software life cycle model.

References

[1] Guide to the Software Engineering: body of knowledge, version 3.0. [Jelektronnyj resurs]. URL: http://www4.ncsu.edu/~tjmenzie/cs510/pdf/SWEBOKv3.pdf

[2] *Kon M.* (2011) Scrum: gibkaya razrabotka programmnogo obespecheniya. Moscow, Vil'jams. 576 p. [In Rus]

[3] *Larman K.*, *Bazili V.* (2003) *Otkrytye sistemy.* 9:43–53. [In Rus]

[4] *http://agilemanifesto.org/iso/ru/*

[5] *Beck K.* (1999) Computer **32**(10):70–77.

[6] *Beck K.* (2002) Addison-Wesley Professional. Addison-Wesley Professional. 252 p.

[7] *Brooks F.* (1987) *Computer*, **20**(4):10–19.

[8] *Cockburn L.* (2008) Using Both Incremental and Iterative Development. *Cross Talk.* May. P. 27–30.

[9] *Larman S.* (2004) Agile and Iterative Development: a Manager's Guide. Addison-Wesley Professional.

[10] *Safronov V. V*, *Fedorec O. N.* (2010) *Informacionnye tehnologii*, 1:34–40 [In Rus]

[11] *Skopin I. N.* (2004) Modeli zhiznennogo tsikla programmnogo obespecheniya. Novosibirsk, Izd. Instituta sistem informatiki im. A. P. Ershova SO RAN. P. 120–173. [In Rus]

[12] *Bahtizin V. V, Gluhova L. A.* (2005) *Doklady Belorusskogo gosudarstvennogo universiteta informatiki i radiojelektroniki*. 1:171–174. [In Rus]
[13] *Skryabin A. M. Kardaş D. İ.* (2008) *Aspirant i soiskatel'*. 2:171–174. [In Rus]
[14] *Chumakova T. Ja., Cyganenko S. M.* (2009) *Matematicheskie mashiny i sistemy*. **1**(3):144–150. [In Rus]
[15] *Golosovskij M. S.* (2014) *Avtomatizacija. Sovremennye tehnologii.* 1:43–46. [In Rus]
[16] *Bogomolov A. V., Majstrov A. I.* (2014) Tekhnologiya analiza sistemnykh prichinno-sledstvennykh svyazey na osnove diagramm Isikavy. *In* Sistemnyj analiz v medicine (SAM 2014) Materialy VIII mezhdunarodnoj nauchnoj konferencii. P. 13–16. [In Rus]
[17] *Golosovskij M. S.* (2015) *Programmnye sistemy i vychislitel'nye metody*. 1:59–68. [In Rus]
[18] *Shpudejko S. A., Bogomolov A. V.* (2006) *Informacionnye tehnologii.* 3:74–79. [In Rus]
[19] *Golosovskij M. S.* (2015) *Programmnye sistemy i vychislitel'nye metody.* 3:311–322. [In Rus]
[20] *Larkin E. V., Bogomolov A. V., Privalov A. N.* (2017) *Nauchno-tehnicheskaja informacija. Serija 2: Informacionnye processy i sistemy.* 9:23–28. [In Rus]
[21] *Golosovskij M. S., Esev A. A., Bogomolov A. V.* (2014) Kompleks avtomatizirovannoy ekspertizy tekhnicheskogo urovnya slozhnoy sistemy. Svidetel'stvo o gosudarstvennoj registracii programmy dlja JeVM no. 2014612330. Opubl. 20.03.2014. 1 p. [In Rus]
[22] *Golosovskij M. S.* (2016) *Programmnaja inzhenerija*. **7**(10):446–455. [In Rus]
[23] *Vigers K., Bitti D.* (2016) Razrabotka trebovaniy k programmnomu obespecheniyu. Russian Edition. [In Rus]
[24] *Bruks F.* (2010) Mificheskiy cheloveko-mesyats, ili kak sozdayutsya programmnyye sistemy. Simvol-Pljus. [In Rus]