TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Школа – ИШИТР
Направление подготовки – 09.03.04 Программная инженерия
Отделение школы (НОЦ) – Отделение информационных технологий

БАКАЛАВРСКАЯ РАБОТА

Тема работы
Управление зависимостями многомодульных enterprise-приложений на Java

УДК 004.654:004.434

Студент

Группа	ФИО	Подпись	Дата
8К61	Цибенко Александр Сергеевич		

Руководитель ВКР

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Кузнецов Д. Ю.	к.т.н.		

КОНСУЛЬТАНТЫ ПО РАЗДЕЛАМ:

По разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОСГН ШБИП	Спицина Л. Ю.	к.э.н.		

По разделу «Социальная ответственность»

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ООД ШБИП	Белоенко Е. В.	к.т.н.		

ДОПУСТИТЬ К ЗАЩИТЕ:

Руководитель ООП	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Чердынцев Е. С.	к.т.н.		

Томск – 2020 г.

TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Школа – ИШИТР
Направление подготовки (специальность) – 09.03.04 Программная инженерия
Отделение школы (НОЦ) – Отделение информационных технологий

УТВЕРЖДАЮ:
Руководитель ООП
_____ _____ Чердынцев Е. С.
(Подпись) (Дата) (Ф.И.О.)

ЗАДАНИЕ
на выполнение выпускной квалификационной работы

В форме:

Бакалаврской работы

(бакалаврской работы, дипломного проекта/работы, магистерской диссертации)

Студенту:

Группа	**ФИО**
8К61	Цибенко Александру Сергеевичу

Тема работы:

Управление зависимостями многомодульных enterprise-приложений на Java	
Утверждена приказом директора (дата, номер)	28.02.2020 №59-51/с

Срок сдачи студентом выполненной работы:	

ТЕХНИЧЕСКОЕ ЗАДАНИЕ:

Исходные данные к работе *(наименование объекта исследования или проектирования; производительность или нагрузка; режим работы (непрерывный, периодический, циклический и т. д.); вид сырья или материал изделия; требования к продукту, изделию или процессу; особые требования к особенностям функционирования (эксплуатации) объекта или изделия в плане безопасности эксплуатации, влияния на окружающую среду, энергозатратам; экономический анализ и т. д.).*	Работа направлена на создание унифицированной системы сборки и управления зависимостями для крупных многомодульных enterprise-приложений, написанных на языке Java
Перечень подлежащих исследованию, проектированию и разработке вопросов *(аналитический обзор по литературным источникам с целью выяснения достижений мировой науки техники в рассматриваемой области; постановка задачи исследования, проектирования, конструирования; содержание процедуры исследования, проектирования, конструирования; обсуждение результатов выполненной работы; наименование дополнительных разделов, подлежащих разработке; заключение по работе).*	1. Анализ существующих проблем в области сборки многомодульных enterprise-приложений на Java; 2. Выбор наиболее оптимальной системы сборки; 3. Перепроектирование наиболее востребованных процессов сборки. 4. Реализация спроектированных процессов; 5. Финансовый менеджмент, ресурсоэффективность и ресурсосбережение; 6. Социальная ответственность.

Перечень графического материала *(с точным указанием обязательных чертежей)*	Диаграмма «Fishbone», диаграммы, описывающие существующие и проектируемые процессы в нотациях IDEF0, IDEF3, DFD, BPMN, EPC, диаграммы последовательностей.

Консультанты по разделам выпускной квалификационной работы
(с указанием разделов)

Раздел	Консультант
Раздел 4. Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	Спицина Л. Ю.
Раздел 5. Социальная ответственность	Белоенко Е. В.

Дата выдачи задания на выполнение выпускной квалификационной работы по линейному графику	

Задание выдал руководитель:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Кузнецов Д. Ю.	к.т.н.		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8К61	Цибенко Александр Сергеевич		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ»

Студенту:

Группа	ФИО
8К61	Цибенко Александру Сергеевичу

Школа	ИШИТР	Отделение школы (НОЦ)	ОИТ
Уровень образования	Бакалавриат	Направление/специальность	Программная инженерия

Исходные данные к разделу «Финансовый менеджмент, ресурсоэффективность и ресурсосбережение»:	
1. *Стоимость ресурсов научного исследования (НИ): материально-технических, энергетических, финансовых, информационных и человеческих*	*Бюджет разработки – не более 500 тыс. руб., в т.ч. затраты на оплату труда – не более 400 тыс. руб.*
2. *Нормы и нормативы расходования ресурсов*	*Премиальный коэффициент 30%;* *Дополнительной заработной платы 12%;* *Накладные расходы 16%;* *Районный коэффициент 1,3.*
3. *Используемая система налогообложения, ставки налогов, отчислений, дисконтирования и кредитования*	*Коэффициент отчислений на уплату во внебюджетные фонды 30 %*
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. *Оценка коммерческого потенциала, перспективности и альтернатив проведения НИ с позиции ресурсоэффективности и ресурсосбережения*	*Оценка коммерческого потенциала и перспективности разработки с позиции ресурсоэффективности и ресурсосбережения:* *- Потенциальные области применения;* *- Анализ уже существующих подходов;* *- Технология QuaD;* *- SWOT-анализ.*
2. *Планирование и формирование бюджета научных исследований*	*Определение основных направлений работы.* *Планирование работ по реализации:* *- Определение структуры работ;* *- Определение трудоемкости выполнения работ;* *- Составление график разработки;* *- Определение бюджета разработки.*
3. *Определение ресурсной (ресурсосберегающей), финансовой, бюджетной, социальной и экономической эффективности исследования*	*Определение интегральных показателей:* *- финансового;* *- ресурсоэффективности;* *- эффективности разработки;* *- сравнительной эффективности.*
Перечень графического материала *(с точным указанием обязательных чертежей)***:**	
1. *Оценка конкурентоспособности технических решений* 2. *Матрица SWOT* 3. *Альтернативы разработки* 4. *График проведения и бюджет разработки* 5. *Оценка ресурсной, финансовой и экономической эффективности разработки*	

Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОСГН	Спицина Л. Ю.	к. э. н.		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8К61	Цибенко Александр Сергеевич		

ЗАДАНИЕ ДЛЯ РАЗДЕЛА
«СОЦИАЛЬНАЯ ОТВЕТСВТЕННОСТЬ»

Студенту:

Группа	ФИО
8К61	Цибенко Александру Сергеевичу

Школа	ИШИТР	Отделение (НОЦ)	ОИТ
Уровень образования	Бакалавриат	Направление/специальность	Программная инженерия

Тема ВКР:

Управление зависимостями многомодульных enterprise-приложений на Java	
Исходные данные к разделу «Социальная ответственность»:	
1. Характеристика объекта исследования (вещество, материал, прибор, алгоритм, методика, рабочая зона) и области его применения	Объект исследования: Реализация информационно-программного комплекса автоматической сборки разрабатываемого программного обеспечения. Область применения: разработка программного обеспечения.
Перечень вопросов, подлежащих исследованию, проектированию и разработке:	
1. Правовые и организационные вопросы обеспечения безопасности: – специальные (характерные при эксплуатации объекта исследования, проектируемой рабочей зоны) правовые нормы трудового законодательства; – организационные мероприятия при компоновке рабочей зоны.	Трудовой кодекс РФ, СанПиН 2.2.4.3359-16, СанПиН 2.2.2/2.4.1340-03, ГОСТ 12.2.032-78 ССБТ, ГОСТ 22269-76.
2. Производственная безопасность: 2.1. Анализ выявленных вредных и опасных факторов 2.2. Обоснование мероприятий по снижению воздействия	1. Отклонение показателей микроклимата помещений 2. Отсутствие или недостаток освещения на рабочем месте 3. Умственное перенапряжение, в том числе вызванное информационной нагрузкой 4. Монотонность труда
3. Экологическая безопасность:	Анализ воздействия объекта на литосферу (загрязнение твердыми бытовыми отходами).
4. Безопасность в чрезвычайных ситуациях:	Анализ наиболее вероятной ЧС в ходе эксплуатации разработанной системы: пожар в здании.

Дата выдачи задания для раздела по линейному графику	

Задание выдал консультант:

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ООД	Белоенко Е. В.	к.т.н.		

Задание принял к исполнению студент:

Группа	ФИО	Подпись	Дата
8К61	Цибенко Александр Сергеевич		

TOMSK POLYTECHNIC UNIVERSITY ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский Томский политехнический университет» (ТПУ)

Школа – ИШИТР
Направление подготовки (специальность) – 09.03.04 Программная инженерия
Отделение школы (НОЦ) – Отделение информационных технологий
Период выполнения – осенний / весенний семестр 2018 /2019 учебного года

Форма представления работы:

Бакалаврская работа

(бакалаврская работа, дипломный проект/работа, магистерская диссертация)

КАЛЕНДАРНЫЙ РЕЙТИНГ-ПЛАН
выполнения выпускной квалификационной работы

Срок сдачи студентом выполненной работы:	

Дата контроля	Название раздела (модуля) / вид работы (исследования)	Максимальный балл раздела (модуля)
	Раздел 1. Обзор предметной области	*20*
	Раздел 2. Проектирование ИС	*20*
	Раздел 3. Реализация ИС	*20*
	Раздел 4. Финансовый менеджмент, ресурсоэффективность и ресурсосбережение	*20*
	Раздел 5. Социальная ответственность	*20*

СОСТАВИЛ:
Руководитель ВКР

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Кузнецов Д. Ю.	к.т.н.		

СОГЛАСОВАНО:

Руководитель ООП

Должность	ФИО	Ученая степень, звание	Подпись	Дата
Доцент ОИТ ИШИТР	Чердынцев Е. С.	к.т.н.		

Реферат

Выпускная квалификационная работа 85 с., 16 рис., 20 табл., 19 источников, 16 прил.

Ключевые слова: сборка приложений, версионирование, Maven, Java, многомодульные приложения, enterprise-приложения, проектирование.

Объектом исследования является система управления зависимостями и сборки многомодульных enterprise-приложений на Java.

Цель работы – ускорение и упрощение разработки проекта путем создания централизованной системы управления зависимостями и сборки многомодульных enterprise-приложений на Java.

В результате исследования устаревшая система сборки проекта была переработана в наиболее неоптимизированных местах, что значительно повысило автоматизированность повседневных процессов разработчиков и повысило их продуктивность.

Степень внедрения: разработанная в выпускной квалификационной работе система управления зависимостями и сборки многомодульных enterprise-приложений на Java успешно внедрена в рабочий процесс организации.

Область применения: команды разработки крупных многомодульных enterprise-приложений, основанных на JVM.

В будущем планируется окончательно адаптировать систему сборки под новую структуру проекта, полностью устранив дублирование зависимостей и стандартизовав управление всеми зависимостями проекта.

Содержание

Введение

В сфере разработки программного обеспечения присутствует достаточно много неоптимизированных рутинных операций, от которых разработчики активно стремятся избавиться с целью повышения своей продуктивности. Так, для того чтобы превратить исходный код в рабочее приложение, это приложение для начала нужно «собрать» – скомбинировать по определенному алгоритму набор компонентов, дающих в совокупности исполняемую программу, или библиотеку кода, готового к запуску. Как раз-таки процесс сборки и является одной из таких рутинных задач, к оптимизации которого также стремятся большинство разработчиков, используя для этого системы сборки и управления зависимостями.

Но порой, частые изменения проекта во время его бурного роста и развития все же со временем приводят к снижению эффективности применения систем сборки. Так, за время развития функциональной подсистемы "Досье НФО", которая разрабатывается и активно применяется в Банке России, количество отдельных модулей, из которых она состоит, значительно возросло. В конечном счете это привело к затрудненной эксплуатации системы сборки и управления зависимостями, в результате чего возникла потребность в ее переработке под актуальные нужды проекта.

После некоторого времени разработки с измененной системой сборки, стало понятно, что она нуждается в доработке. В связи с этим была поставлена следующая цель: упростить и ускорить разработку проекта путем создания централизованной системы управления зависимостями и сборки многомодульных Enterprise-приложений на Java.

1 Обзор предметной области

1.1 Обзор предметной области

Практически любое приложение в процессе разработки проходит через множество повторяющихся рутинных этапов, таких как сборка проекта, выполнение модульных тестов, генерация артефактов, развертывание приложения в том или ином окружении и пр. Причем, чем больше становится приложение, чем больше отдельных функциональных подсистем у него появляется, чем больше разработчиков над ним трудятся, тем больше вероятность сделать ошибку на каком-либо этапе, а также тем сложнее становится процесс и больше суммарного времени и сил тратится на все эти действия.

По этим причинам большинство разработчиков используют при разработке своих приложений так называемые “сборщики проектов”. Сборщик проектов - это программное обеспечение, применяемое при разработке других программ и позволяющее автоматизировать такие задачи как компилирование, сборка, упаковка проекта, генерация артефактов и документации, выполнение автоматических тестов, развертывание собранного приложения в тестовой, продуктовой, или любой другой среде, управление прямыми и транзитивными зависимостями. Как правило, такие системы являются легко расширяемыми, для того, чтобы разработчики, использующие данные системы, могли добавлять свои уникальную логику в процессы сборки и доставки приложения.

1.2 Обзор существующих проблем

Несмотря на все плюсы современных сборщиков, их использование еще не гарантирует правильного и автоматического выполнения всех поставленных перед ними задач. В каждом проекте, независимо от системы сборки, описывается вся необходимая информация, основываясь на которой, используемая система и выполняет возложенные на нее задачи. Управление системой сборки является такой же полноценной задачей, как и написание программного кода, ведь от ее успеха зависит эффективность выполнения многих других задач разработки – разработчики могут сосредоточиться

непосредственно на создаваемом продукте, из-за чего повышается их продуктивность.

Так, за время развития функциональной подсистемы "Досье НФО", которая разрабатывается и активно применяется в Банке России, возникла проблема возросшего числа его отдельных модулей, появлявшихся в ходе разработки проекта. Одни модули были вынесены в собственные репозитории, в то время как другие создавались по мере необходимости в репозитории back-end части проекта. Со временем было принято решение централизовать хранение всех имеющихся модулей в рамках одного большого проекта. Действия по переносу отдельных модулей привели к несогласованностям в системе сборки и управления зависимостями между различными частями проекта.

Для анализа текущего состояния проекта и поиска накопившихся в нем проблем была построена диаграмма «Fishbone», представленная на рисунке 1. Данная диаграмма позволяет наглядно структурировать причинно-следственные связи возникновения проблем.

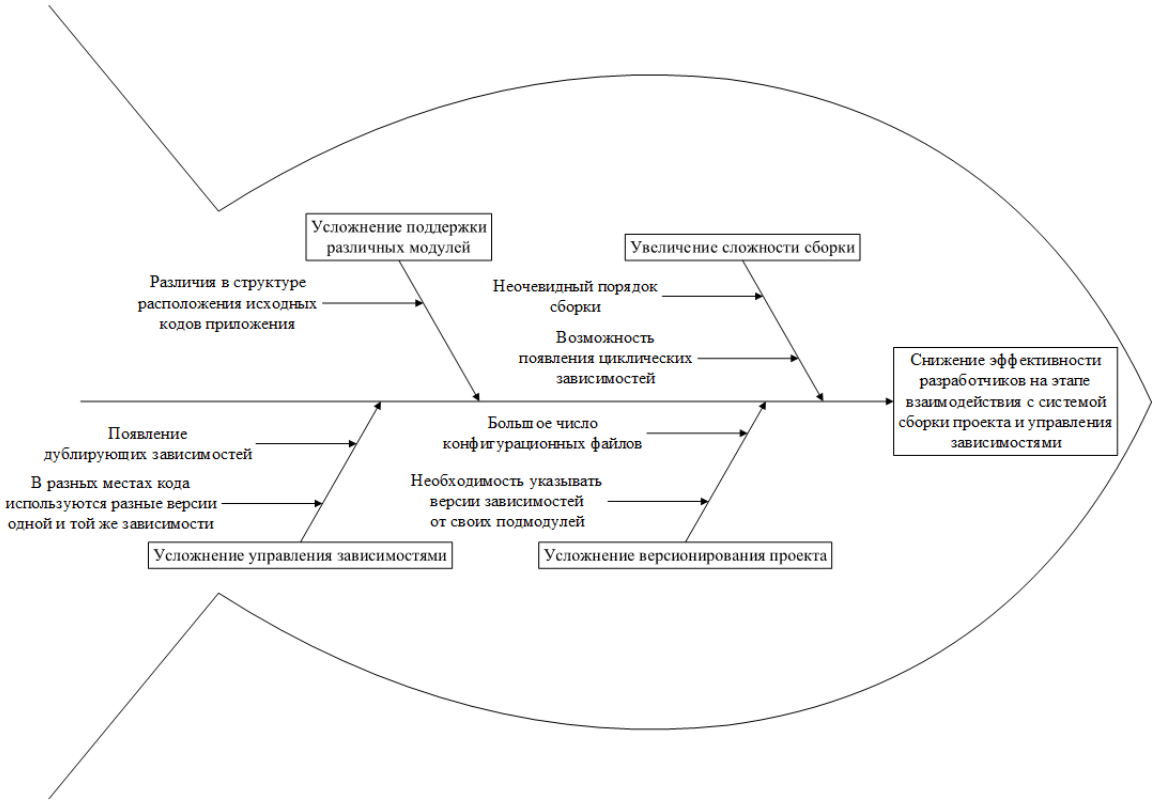


Рисунок 1 – Диаграмма «Fishbone»

Из диаграммы, представленной на рисунке 1 можно выделить четыре основные проблемы, влекущие за собой снижение эффективности разработчиков при взаимодействии с системой сборки проекта:

1. Модули, которые ранее были отдельными проектами, по-прежнему не могут проходить процесс сборки вместе с остальной частью проекта, так как были неправильно интегрированы в существующую структуру, что вынуждает разработчиков все также выполнять сборку этих модулей отдельного от всего проекта.
2. Зависимости, которые ранее описывались для обособленных модулей, ныне дублируют друг друга, причем их версии порой не совпадают.
3. Затруднена смена версии проекта при каждом новом релизе, т.к. необходимо во всех конфигурационных файлах системы сборки и управления зависимостями менять старую версию ПО на новую, как в описании версии каждого подмодуля, так и в указании зависимостей одного модуля от другого. Причем, из-за ошибок интеграции обособленных модулей, в некоторых местах сборщик не может обнаружить необходимость смены версий и оставляет их в неактуальном состоянии.
4. Каждый новый модуль создавался разными командами разработчиков, что отразилось на внутренней структуре каждого из них. Поэтому, при открытии разных модулей разработчикам приходится сначала вникать в структуру проекта, разбираться какие директории за что отвечают, а только потом переходить непосредственно к решению возникшей проблемы.

1.3 Обзор существующих процессов

1.3.1 Выявление наиболее важных функций приложения

В приложении такого большого размера сборщик проекта выполняет огромное множество функций, рассматривать все из которых не целесообразно

в рамках данной работы. Поэтому данная работа ограничивается рассмотрением лишь нескольких наиболее важных из них.

Чтобы лучше понять, какие рутинные операции чаще всего выполняет система сборки в функциональной подсистеме «Досье НФО», была создана контекстная диаграмма системы в нотации DFD. Она представлена на рисунке 2.

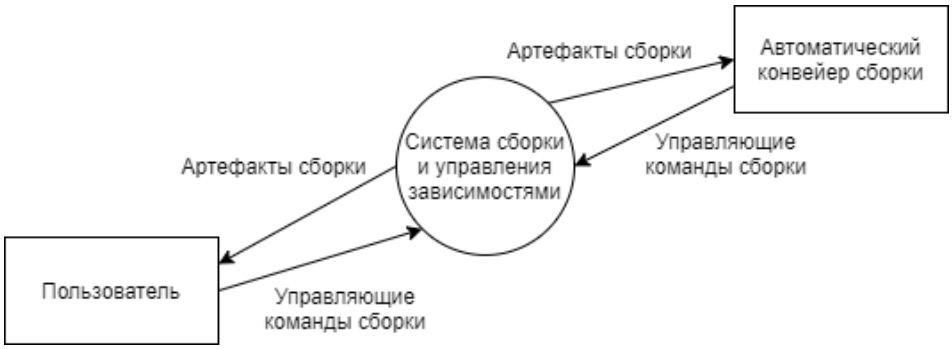


Рисунок 2 – Контекстная диаграмма в нотации DFD

Из этой диаграммы сразу же бросается в глаза наличие двух основных категорий пользователей данной системы: пользователь, то есть разработчик, и автоматический конвейер сборки, то есть программное обеспечение, работающее (чаще всего) на сервере.

Для более детального анализа системы была проведена декомпозиция контекстной диаграммы с рисунка 2. Результаты декомпозиции приведены в приложении А. Как видно из данной диаграммы, в основном система выполняет единственную задачу – сборку приложения, но при этом делает она это несколько по-разному, в зависимости от целей конкретной сборки.

В следующих пунктах будут обобщенно рассмотрены различия ручной и автоматической сборки приложения, а также процесс смены версии приложения.

1.3.2 Полная ручная сборка приложения

На текущий момент сборка проекта осуществляется не автоматически. Разработчику приходится при каждой сборке повторять одни и те же шаги раз за разом. Для лучшего понимания этого процесса, была проведена его декомпозиция. На рисунке 3 представлена контекстная диаграмма сборки

приложения в нотации IDEF0, а в приложениях Б и В ее последовательные декомпозиции вплоть до самых базовых подмодулей приложения (на примере модуля печатных форм).

Кроме линейного порядка сборки, из диаграмм, представленных в приложениях Б и В видно, что итоговый артефакт сборки приложения может различаться. Это следствие того, что приложение может запускаться на нескольких различных серверах, которые требуют разных алгоритмов сборки. Эта особенность уже была видна ранее на DFD диаграмме в приложении А. Добиться такой вариативности сборки приложения в существующей реализации системы можно с использованием профилей Maven.

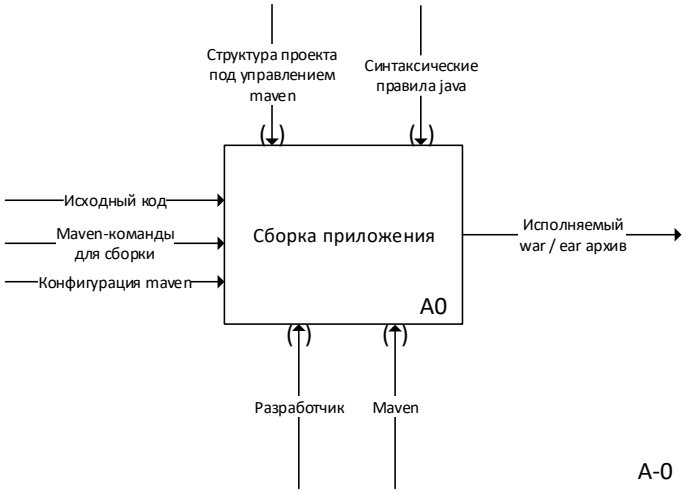


Рисунок 3 – Контекстная диаграмма в нотации IDEF0

Профили Maven представляют собой именованные и полностью подготовленные заранее конфигурации, которые расписаны внутри основной конфигурации модулей и их подмодулей. Так, для сборки под сервер Wildfly (ранее Jboss), достаточно указать в команде сборки серверной части приложения помимо всего прочего профиль «Jboss»: «mvn clean install -P Jboss [остальные параметры запуска]». Для запуска сборки для сервера WebSphere необходимо указать профиль «WebSphere» аналогичным образом. И таких профилей, применяемых в процессе сборки, во всех модулях и подмодулях приложения достаточно много.

С помощью нотации IDEF3 была проведена декомпозиция блока А4 из диаграммы в приложении Б с целью выявления влияния двух самых основных профилей на процесс сборки подмодуля серверной части приложения. Результаты декомпозиции приведены в приложении Г.

1.3.3 Полная автоматическая сборка приложения на сервере

В данном проекте есть два основных сценария автоматической сборки приложения: для сборки и развертывания приложения на стендах тестирования или разработки (применяется инструмент Jenkins), или же автоматическая сборка приложения с проведением модульного и интеграционного тестирований, а также выполнения других всевозможных проверок при оформлении разработчиком Merge Request в системе Gitlab (используется инструментарий Gitlab CI/CD).

Merge Request представляет собой запрос на внесение изменений по конкретной задаче разработчика в общий для всех разработчиков отлаживаемый набор изменений. Подробнее этот процесс можно рассмотреть с помощью диаграммы в нотации BPMN в приложении Д. Она очень удобна тем, что позволяет с практически минимальным набором визуальных элементов разобрать бизнес-процесс.

Более детально данный процесс можно описать с помощью диаграммы в нотации EPC, представленной в приложениях Е и Ж. Стоит отметить, что данная нотация позволяет четко увидеть состояние, в котором находится процесс в каждый промежуток времени, а также в деталях рассмотреть информационный обмен различных участников процесса.

1.3.4 Смена версии приложения

В существующей системе смена версий проекта весьма затруднена. Для того, чтобы сменить версию, каждые две недели отводится полноценная трудоемкая задача.

Общее количество конфигурационных файлов в проекте превышает 70 штук, и в каждом файле, как минимум в его заголовке, требуется поменять версию, но многие конфигурационные файлы указывают в качестве своих зависимостей другие подмодули приложения, что значительно затрудняет отслеживание всех мест в проекте, в которых необходимо заменить версию приложения на новую.

Более того, в некоторых конфигурационных файлах в какой-то промежуток времени не ввели правило версионирования и их версия абсолютно не соответствует версии проекта и никак не обновляется долгое время.

На рисунке 4 приведена диаграмма последовательности по смене версии приложения для наиболее оптимального расклада, когда разработчик ничего не упустил из виду и не испортил своими действиями.

Рисунок 4 – Диаграмма последовательности смены версии приложения

Как видно из данной диаграммы, процесс смены версии приложения в существующей системе крайне неэффективен и отнимает множество времени и сил.

1.4 Постановка цели и задач

Суммируя все вышесказанное, целью работы является ускорение и упрощение разработки проекта путем создания централизованной системы управления зависимостями и сборки многомодульных Enterprise-приложений на Java. Для достижения поставленной цели были выявлены следующие задачи:

1. Выбрать наиболее оптимальную систему сборки проекта;
2. Перепроектировать выявленные проблемные процессы;
3. Реализовать изменения.

1.5 Требования к системе

Программная система должна выполнять следующие функции:

1. Сборка всего проекта в приложение на базе сервера приложений “WebSphere Application Server” путем запуска единственной короткой команды в корневом каталоге проекта.
2. Сборка всего проекта в приложение на базе сервера приложений “WildFly” путем запуска единственной короткой команды в корневом каталоге проекта.
3. Выполнение модульных тестов для всех подсистем во время сборки проекта способами как из требования №1, так и из требования №2, путем добавления дополнительного параметра к команде запуска сборки.
4. Выполнение интеграционных тестов для всего собранного приложения как способами как из требования №1, так и из требования №2, путем добавления дополнительного параметра к команде запуска сборки.
5. Смена версии всего приложения единственным действием: путем запуска единственной короткой команды в корне проекта, или изменением единственного свойства в корневом конфигурационном файле.

6. Возможность сборки любого из модулей ФПС запуском команды, аналогичной команде из требований №1 или №2 (или более простой) в корневом каталоге данного модуля.
7. Генерация отчета о покрытии кода автотестами.
8. Генерация документации.

2 Проектирование информационной системы

2.1 Обзор существующих систем сборки

На текущий момент существует три наиболее популярных и современных промышленных сборщика для JVM-проектов: Ant, Maven и Gradle. Такая популярность их использования именно для JVM-проектов обусловлена тем, что эти инструменты сами написаны и работают на этой виртуальной машине, что также обеспечивает их кроссплатформенность.

2.1.1 Ant

Apache Ant (Another Neat Tool) появился первым среди рассматриваемых инструментов и предоставлял возможность сборки проектов, описывая различные этапы этого процесса с помощью XML-конфигурации, называемой сценарием. В сценарии разработчик перечисляет различные высокоуровневые цели, которых он хочет достичь в своем проекте благодаря использованию Ant, а также расписывает последовательность действий, необходимую для достижения поставленных целей.

Ant является системой с открытым исходным кодом и обычно используется для автоматизации типовых задач, регулярно выполняемых в проектах, разрабатываемых на Java. К таким задачам можно отнести преобразование исходного кода в байт-код, запуск модульных тестов, упаковка приложения в JAR-архивы, а также создание документации Javadoc.

Сам по себе Ant не имеет собственной подсистемы управления зависимостями и при необходимости такой функциональности приходится использовать сторонние инструменты, такие, как Ivy (Ivy – отдельный подпроект Apache для Ant с открытым исходным кодом, берущий на себя управление прямыми и транзитивными зависимостями в проекте).

Также при создании нового проекта с использованием Ant разработчику приходится каждый раз заново описывать типовые цели и шаги по их достижению, что увеличивает дублирование кода и снижает продуктивность

разработчика. Однако, это дает разработчику полную свободу в организации проекта, гибкость и расширяемость конфигураций.

2.1.2 Maven

Apache Maven, в отличии от Ant, является уже не просто инструментом для сборки, а инструментом для управления проектами. Разница между двумя этими понятиями заключается в том, что все действия по достижению тех или иных целей уже написаны заранее. Разработчик в конфигурации, которая находится в файле «pom.xml» в корне проекта, лишь описывает свой проект: его название, кто разработчик, во что должен собираться проект, как должны запускаться модульные тесты, какие зависимости использует проект и много чего еще, что разработчик посчитает нужным (но в рамках стандартов, определенных Maven).

Проект, описываемый с помощью Maven должен соответствовать определенной стандартизированной структуре, с которой может работать Maven. То же самое касается и конфигурационного файла – в нем нельзя просто так описать что угодно. Можно использовать лишь заранее определенные теги, использование которых также должно соответствовать заранее определенной стандартизированной структуре файла. Вообще вся философия Maven направлена на уменьшение дублирования кода: все что нужно для описания проекта уже заранее написано и стандартизовано, разработчику остается только выбрать необходимые компоненты для применения в своем проекте.

Помимо всех тех возможностей, что дает Ant, Maven также имеет встроенные инструменты для управления зависимостями проекта, позволяет описывать метаинформацию о проекте (название продукта, его версию, лицензию распространения, контакты разработчиков и тд), указывать дочерние подмодули проекта и много чего еще.

Отдельно стоит отметить систему профилей и поддержку гибкой системы плагинов. Профили позволяют частично или полностью переопределить все описание проекта: от конкретных настроек проекта, до совершенно другого

набора используемых подмодулей и плагинов. Плагины же в свою очередь представляют собой отдельные программы, которые помогают Maven выполнить определенные, специфичные конкретно для этого проекта, действия на нужном этапе жизненного цикла приложения. С помощью плагинов можно, например, по расположенной в исходном коде проекта метаинформации генерировать другой недостающий исходный код проекта, или, например, также по метаинформации генерировать документацию к проекту.

2.1.3 Gradle

При создании Gradle разработчики попытались учесть все недостатки Ant и Maven и сделать продукт, который стал бы чем-то средним между этими инструментами, вобрав в себя все самое лучшее от обоих. Конфигурация Gradle размещается в файле «build.gradle», если для его написания используется язык Groovy, или же в файле «build.gradle.kts», если для его написания используется язык Kotlin. Таким образом, Gradle уходит от конфигураций на базе XML и использует непосредственно языки программирования для своих конфигураций, что позволяет реализовывать более сложную логику, которая может понадобиться во время сборки проекта, а также максимально уменьшает избыточность, присущую XML тегам.

Примечательно, что Gradle имеет инструменты для переноса всех старых конфигураций проекта, будь они написаны хоть для Maven, хоть для Ant. Более того, Gradle можно интегрировать в уже существующую инфраструктуру системы управления зависимостями (например, используемую Ivy, или Maven).

По умолчанию Gradle использует стандартную для Maven структуру проекта, что побуждает разработчиков действовать более стандартизованно, вследствие чего повышается читаемость проекта и удобство работы с ним для разных разработчиков, однако, при желании можно задать и свою уникальную структуру проекта.

Gradle, также как и Maven, позволяет быстро в декларативном стиле описывать типовой проект, используя уже заранее написанные плагины,

прописывать типовую метаинформацию о проекте, указывать подмодули проекта и внешние зависимости, гибко настраивать это все. Но при этом Gradle также, как и Ant, позволяет вручную писать задачи, необходимые для сборки проекта, что опять-таки дает разработчикам ту свободу действий, какую давал им Ant, но при этом уже на полноценном языке программирования Groovy, а не с помощью описания в XML ограниченного набора типовых функций.

Еще одно отличие от Maven заключается в том, что в Gradle нет наследования конфигураций в их классическом понимании: конфигурация, которая хочет унаследовать свойства другой конфигурации, не может этого сделать. Наследование возможно лишь из специальных блоков в родительских конфигурационных файлах, а порядок наследования определяется в специальных файлах настройки [7].

2.2 Выбор системы сборки

Для выбора наиболее оптимального сборщика среди рассмотренных был использован метод взвешенной суммы критериев, где список критериев был составлен на основе экономических соображений по переносу решения на новую систему, предъявляемых функциональных требований к системе, а также на основе возможности реализации существующих в проекте решений в рамках новой системы. Оценка каждому критерию давалась по пятибалльной шкале, соответственно, максимальный бал, который могла получить система сборки при данной оценке также равен пяти. Результаты анализа приведены в таблице 1. Буквы A, M и G в таблице означают Ant, Maven и Gradle соответственно.

Таблица 1 – Результаты оценки конкурентоспособности систем сборки

Критерии оценки	**Вес критерия**	**Сборщики проектов**		
		Ant	**Maven**	**Gradle**
Легкость переноса проекта на данный сборщик	0,15	1	5	3
Время переноса проекта на данный сборщик	0,1	1	5	3
Наличие плагинов для сборки под WebSphere	0,1	3	5	5
Наличие плагинов для сборки под WildFly	0,1	3	5	5
Выполнение модульных тестов с генерацией отчета	0,02	3	5	5
Выполнение интеграционных тестов с генерацией отчета	0,02	3	5	5
Возможность управления метаинформацией проекта	0,03	1	5	5
Работа с многомодульными проектами	0,05	2	4	4
Наличие плагинов под проверку стилистики кода	0,01	3	5	5
Возможность генерации документации	0,01	5	5	5
Возможность переключения между различными сценариями сборки	0,05	1	5	4
Наличие механизмов управления зависимостями	0,1	3	5	5
Удобство работы с большими проектами	0,1	2	4	4
Квалификация персонала в работе с данным инструментом	0,16	1	4	3
Итого	**1**	**1,89**	**4,69**	**3,98**

Как видно из таблицы 1, в рамках переработки имеющейся системы наиболее оптимальным вариантом остается все-таки Maven. В первую очередь это связано с достаточно большими затратами на полный перенос системы, в процессе которого непременно будут возникать неочевидные проблемы и т.д. Тем не менее, Gradle практически по всем параметрам ничем не уступает Maven, но, к сожалению, перенос системы на него не стоит того.

2.3 Перепроектирование процесса сборки приложения

Основной проблемой, повлекшей за собой понимание необходимости переработки системы, стало привлечение разработчика к ручной сборке практически каждого подмодуля каждый раз, когда он хочет пересобрать проект, что абсолютно не эффективно и тратит его драгоценное рабочее время и внимание.

В связи с этим, диаграмма процесса полной сборки приложения в нотации IDEF0 с рисунка 3 и ее декомпозиция из приложения Б были переработаны под желаемый результат, который требовал бы участия разработчика только один раз – для запуска самого процесса. Переработанная контекстная диаграмма представлена на рисунке 5, а ее декомпозиция в приложениях З.

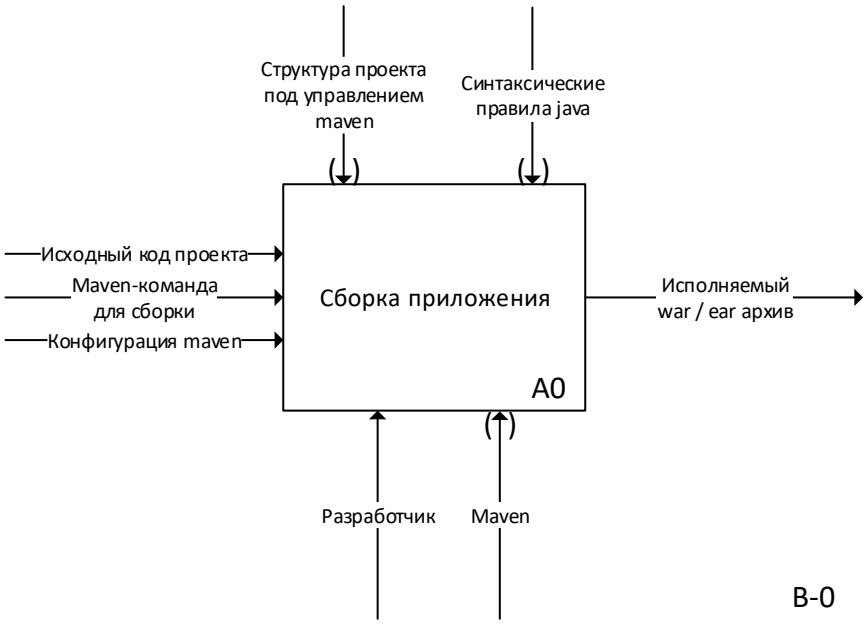


Рисунок 5 – Контекстная диаграмма переработанного процесса в нотации IDEF0

Стоит отметить, что на уровне отдельных модулей и их подмодулей процессы в принципе уже и так отлажены и работают достаточно хорошо, ведь ранее они разрабатывались в своих отдельных репозиториях и у них была своя отработанная система сборки. По этой причине проводить более детальную декомпозицию, как в приложении В, не имеет смысла, ведь на этом уровне процессы останутся без существенных изменений.

2.4 Перепроектирование процесса смены версии приложения

Также, как и выполнение сборки всего приложения, смена версии приложения также должна выполняться единственной командой (диаграмма данного варианта приведена на рисунке 7), или, что является еще более желаемым результатом – изменять версию всего приложения единственным файлом (диаграмма данного варианта приведена на рисунке 6).

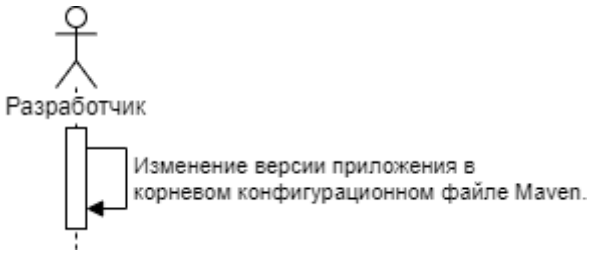


Рисунок 6 – Диаграмма варианта использования смены версии приложения ее заменой в корневом конфигурационном файле приложения

Такой вариант является наиболее предпочтительным по причине того, что после должной отладки и всех проверок, при смене версии приложения в списке изменений будет всего один единственный файл – корневая конфигурация, что является максимально наглядным и простым методом смены версии приложения.

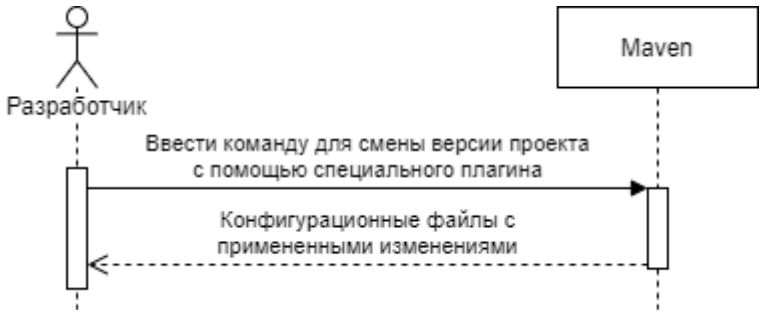


Рисунок 7 – Диаграмма варианта использования смены версии приложения вызовом соответсвующей Maven команды

Данный вариант исполнения также является приемлемым, однако у него есть небольшой недостаток по сравнению с предыдущим: при смене версии изменения все равно затронут все конфигурационные файлы проекта (которых более 70 штук) и в таком потоке изменений достаточно просто пропустить какое-либо случайное изменение, временно внесенное разработчиком локально в проект, которое не должно попасть в основную ветку разработки.

3 Реализация информационной системы

3.1 Реализация нового процесса полной ручной сборки приложения

3.1.1 Описание необходимых для реализации инструментов

Maven предоставляет три основных инструмента связывания различных модулей приложения:

- Использование другого модуля в качестве зависимости;
- Наследование конфигурации;
- Агрегация подмодулей.

Для целей автоматизации сборки наибольшую важность несет именно агрегация подмодулей. Она позволяет из родительского конфигурационного файла указывать его подмодули, что говорит Maven о том, что введенную команду также необходимо применять и для подмодулей.

Более того, во время анализа зависимостей Maven строит граф зависимостей и убеждается, что он ацикличный. После этого в этом графе он находит модули, которые необходимо собирать в рамках запуска текущей команды и строит в графе зависимостей такой путь, чтобы на каждом следующем шаге сборки отсутствовали не собранные зависимости собираемого модуля. Другими словами, Maven сам определяет необходимый порядок сборки, если все зависимости указаны корректно.

Использование профилей поможет ограничить круг затрагиваемых при сборке модулей, а также позволит определить уникальные для того или иного сценария свойства и применяемые плагины.

3.1.2 Реализация

Прежде всего, модули, которые учувствуют в сборке, были разделены по частоте внесения изменений. При детальном анализе выяснилось, что чаще всего изменяется серверная часть приложения. Отдельно от серверной части приложения может изменяться front-end часть приложения, если задача подразумевает разработку именно в ней. Гораздо реже меняется ядро

приложения, модуль печатных форм и модуль работы с внешними источниками. По сути их приходится пересобирать только при смене версии приложения.

Приложение является очень большим, и его сборка осуществляется десятками минут. Поэтому было принято решение разграничить сборку с помощью профилей, выделив ряд комбинаций модулей по частоте пересборки:

1. Application – полная сборка приложения, применяется после смены версии приложения, когда в локальном репозитории требуются все модули актуальной версии;
2. IntegrationTests – полная сборка приложения со сборкой и выполнением интеграционных тестов (в основном применяется на сервере из-за сложности корректной локальной настройки);
3. FrontEnd – сборка исключительно front-end части приложения, применяется при работе front-end разработчиков;
4. Backend – сборка исключительно back-end части приложения, регулярно применяется для отладки приложения во время разработки.

Далее все выявленные профили были добавлены в корневой конфигурационный файл:

```
<profile>
    <id>IntegrationTests</id>
    <modules>
        <module>cr-dss-be</module>
        <module>cr-dss-db</module>
        <module>cr-dss-etl</module>
        <module>cr-dss-pf</module>
        <module>cr-dss-ear</module>
        <module>cr-dss-qa</module>
        <module>cr-dss-it</module>
    </modules>
</profile>
```

Рисунок 8 – Профиль сборки с интеграционным тестированием

После добавления профилей необходимо согласовать их названия с уже существующими профилями в модулях: если в модуле присутствуют профили, которые использовались в том же контексте, что и новые профили, то названия таких профилей следует согласовать с новым профилем. Так, например, на

рисунке 8 изображен новый профиль для сборки с интеграционным тестированием в корневом конфигурационном, а на рисунке 9 приведенные ему в соответствие профиль в модуле серверной части приложения.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <artifactId>cr-dss-be</artifactId>
    <packaging>pom</packaging>

    …

    <profiles>

        …

        <profile>
            <id>IntegrationTests</id>
            <modules>
                <module>cr-dss-tests</module>
            </modules>
        </profile>

        …

    </profiles>

    …
```

Рисунок 9 – Приведенный в соответствие профиль внутри модуля

После разработки все модули тщательно отлаживались и тестировались в разных комбинациях. В результате типичный цикл разработки стал выглядеть следующим образом:

1. В начале работы над новой версией, разработчик через систему контроля версий получает все последние обновления приложения;
2. Разработчик активирует полную сборку новой версии приложения путем применения команды «mvn clean install -P Application -P Jboss», причем время сборки приложения сократилось с приблизительно 30 минут с учетом постоянного ручного ввода команд до 15;
3. Разработчик активно работает над текущей задачей, и при необходимости проверки внесенных изменений непосредственно в работе приложения собирает серверную часть путем применения команды «mvn clean install -

P Backend -P Jboss», причем время сборки приложения сократилось с приблизительно 6 минут с учетом постоянного ручного ввода команд до 3;

3.2 Реализация нового процесса смены версии приложения

3.2.1 Реализация на основе наследования версии

Наиболее оптимальным вариантом смены версии приложения является смена версии в корневом конфигурационном файле с автоматическим применением изменений на все модули и их подмодули.

Для реализации такого способа идеально подходит инструментарий, появившаяся в Maven только начиная с версии 3.5.0. Этим инструментарием является специальная зарезервированная переменная «revision» [8] особенность данной переменной заключается в том, что ее помимо всех основных мест в конфигурации можно применять также и в теге «parent».

Другими словами, при объявлении в корневом конфигурационном файле «revision», ее можно использовать и внутри тега «version» это же файла, и внутри тега «parent», который находится в конфигурационном файле дочернего модуля. Тэг «parent», отвечает за механизм наследования конфигурации, соответственно, если свойство «revision» не объявлено в конфигурационном файле подмодуля, то оно берется как раз-таки из корневого конфигурационного файла.

Однако, чтобы воспользоваться данным инструментом, абсолютно все конфигурационные файлы проекта должны находиться в иерархии наследования. Для реализации подобной иерархии необходимо быть уверенным, что многочисленные наследования не станут причиной конфликтов зависимостей и многочисленных настроек.

Для анализа осуществимости подобных изменений была проанализирована абсолютно вся структура взаимосвязей конфигурационных файлов проекта. В целях защиты внутрибанковской информации полные результаты анализа в данной работе не приведены. Однако, можно сказать, что у некоторых модулей не было четкой иерархии конфигурации, а у другой части (в

основном у наиболее важных модулей) иерархия была в некотором виде искаженной иерархией из второй части статьи «Идеальный Мавен» [9]. Это стало огромным преимуществом, т.к. во всех модулях присутствовали корневые (для каждого отдельного модуля) конфигурационные файлы, в которых не было объявления зависимостей и практически не было никаких настроек, а в основном метаинформация о конкретном модуле. Это позволило собрать черновой вариант реализации иерархии, часть структуры которой приведена на рисунке 10.

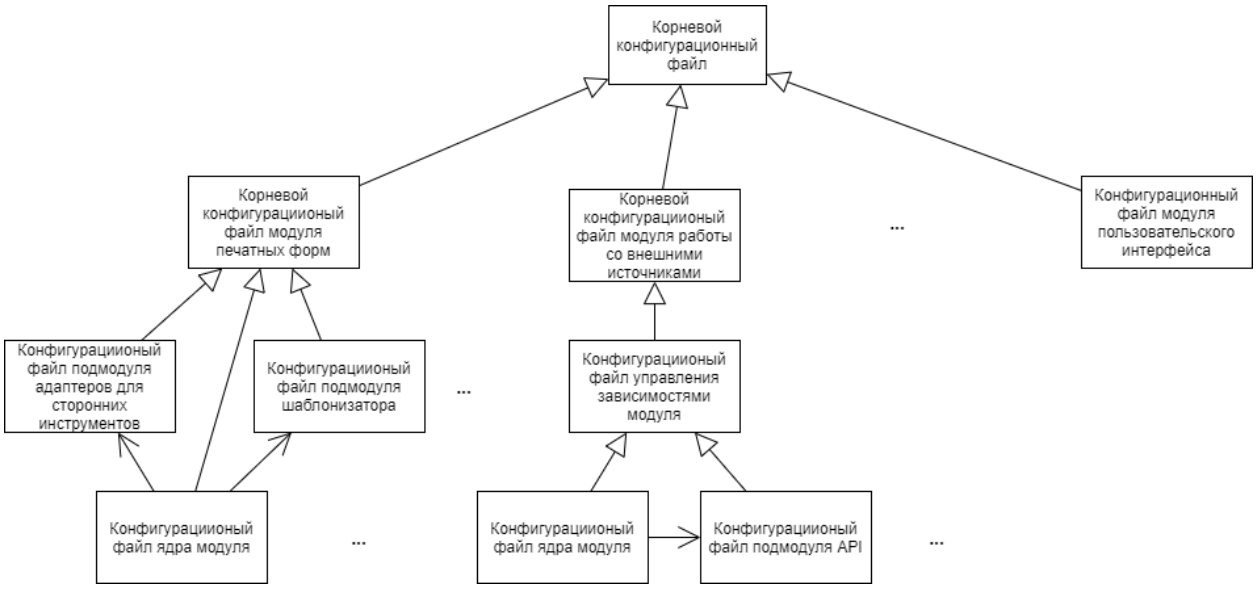


Рисунок 10 – Черновая макетная иерархия конфигурационных файлов

После черновой реализации и отладки иерархии в корневом конфигурационном файле была объявлена переменная «revision» (рисунок 11), а во всех ссылках на подмодули данного проекта (в том числе и внутри тегов «parent») конкретные версии подмодулей были заменены указанием данной переменной (рисунок 12).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <artifactId>cr-dss</artifactId>
    <version>${revision}</version>
    <packaging>pom</packaging>

    …

    <properties>
        <revision>3.0.0</revision>
        …
```

Рисунок 11 – Указание и применение переменной «revision»

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <artifactId>cr-dss-etl-was</artifactId>
    <packaging>war</packaging>
    <parent>
        <artifactId>cbr-dossier-etl-base</artifactId>
        <version>${revision}</version>
        …
    </parent>


    <dependencies>
        <dependency>
            <artifactId>cr-dss-etl-web</artifactId>
            <type>jar</type>
            <version>${revision}</version>
            …
        </dependency>
        …
```

Рисунок 12 – Использование переменной «revision» в подмодулях

После разработки началась стадия тщательного тестирования, в ходе которого выяснилось, что в некоторых случаях система производила сборку корректно (рисунок 13), а в некоторых случаях сборка оканчивалась самыми разными неинформативными ошибками дающих лишь понять, что определенный артефакт с некорректной версией не может быть найдет.

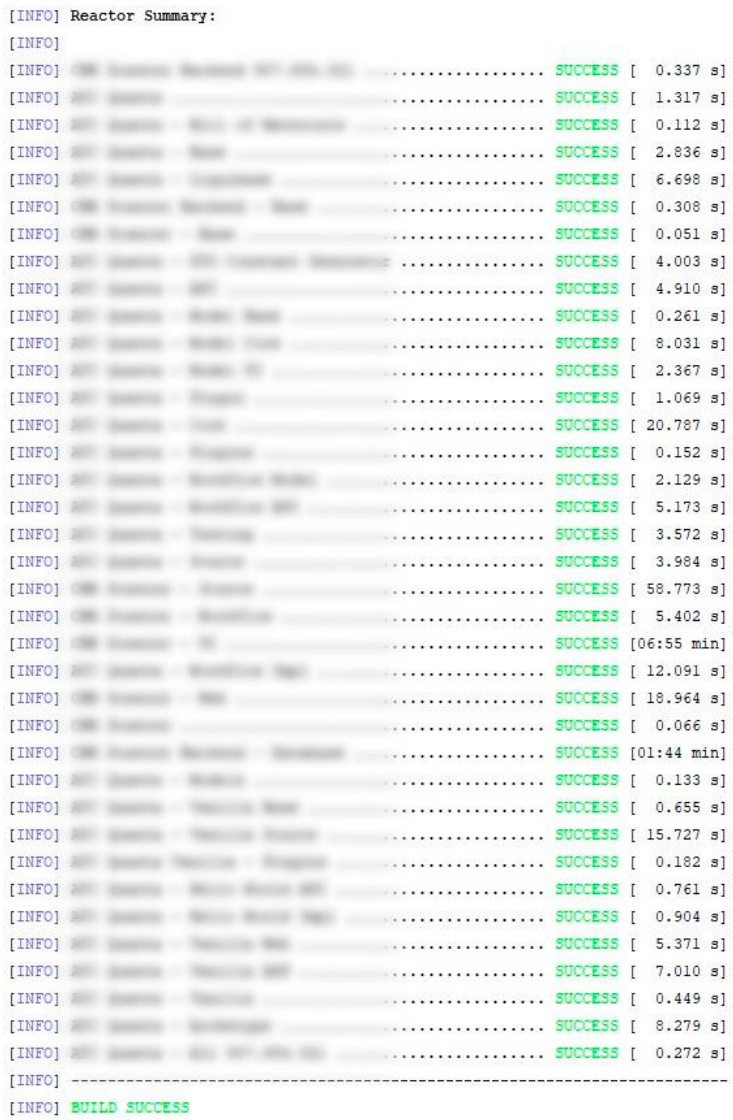


Рисунок 13 – Пример успешной сборки проекта (названия модулей скрыты)

Скорее всего, Maven в ряде случаев не может корректно обработать такую сложную иерархию модулей, в которой порой насчитывается до 5 уровней наследования и больше 70 конфигурационных файлов. Такая нестабильность поведения неприемлема, из-за чего пришлось отказаться от данной реализации версионирования приложения.

3.2.2 Реализация на основе плагина

Вторым вариантом реализации версионирования являлось применение плагина Versions Maven Plugin [10]. Данный плагин проходит по иерархии агрегации модулей и находит места для замены старой версии на новую. Однако, его применение сопряжено с рядом проблем:

1. Иерархия агрегации в текущей системе профилей охватывает далеко не все подмодули проекта;
2. Алгоритмы плагина по определению мест для замены версий не совершенны и в некоторых случаях могут пропускать версии подмодулей в качестве зависимостей;
3. При оформлении запроса на слияние, точек замены все еще очень много – легко пропустить случайное лишнее изменение.

Последнюю проблему, к сожалению, устранить полностью не удастся, однако количество изменений при смене версии было сокращено в несколько раз благодаря замене прямого указания версии подмодулей в секции описания зависимостей других подмодулей на переменную версии проекта «project.version» (рисунок 14). Данное решение также исправило вторую проблему. Переменная «project.version» не смогла бы стать заменой переменной «revision» из предыдущего способа реализации, так как она недоступна внутри заголовочных тегов конфигурации.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <artifactId>cr-dss-etl-was</artifactId>
    <packaging>war</packaging>
    <parent>
        <artifactId>cbr-dossier-etl-base</artifactId>
        <version>3.0.0</version>
        …
    </parent>


    <dependencies>
        <dependency>
            <artifactId>cr-dss-etl-web</artifactId>
            <type>jar</type>
            <version>${project.version}</version>
            …
        </dependency>
             …
```

Рисунок 14 – Замена прямого указания версии зависимости на переменную

Проблему неполного охвата подмодулей было решено обойти с помощью создания отдельного профиля, предназначенного исключительно для смены версии приложения, в котором перечислены абсолютно все модули и подмодули проекта. Также данный профиль был настроен на активацию по переменной «newVersion», которая указывается при запуске плагина смены версии, что позволило активировать созданный профиль только при использовании данного плагина. Для упрощения команды смены версии проекта необходимые настройки плагина были также вынесены в созданный профиль. Итоговая реализация профиля для смены версии приложения представлена на рисунке 15.

Далее в документацию для разработчиков приложения было внесена инструкция по смене версии приложения. Снимок инструкции представлен на рисунке 16.

```
<profile>
    <id>VersionChanging</id>

    <activation>
        <property>
            <name>newVersion</name>
        </property>
    </activation>

    <properties>
        <generateBackupPoms>false</generateBackupPoms>
        <processAllModules>true</processAllModules>
    </properties>

    <modules>
        <module>cr-dss-be </module>
        <module>cr-dss-be/cr-dss-base</module>
        <module>cr-dss-be/cr-dss-source</module>
        …
    </modules>
</profile>
```

Рисунок 15 – Итоговая реализация профиля для смены версии приложения

Рисунок 16 – Инструкция по смене версии приложения в документации

Команду смены версии приложения, представленную в документации, удалось сократить до минимума благодаря использованию активации профиля по переменной, используемой в плагине, а также вынесению настроек плагина в этот же профиль.

4 Финансовый менеджмент, ресурсоэффективность и ресурсосбережение

4.1 Предпроектный анализ

В работе рассматривается программное обеспечение, позволяющее централизовать управление зависимостями, а также автоматизировать сборку крупных корпоративных приложений, разрабатываемых и применяемых в Центральном Банке Российской Федерации.

Актуальность разработки обусловлена большими масштабами разрабатываемых приложений, процесс сборки которых усложняется при росте как самих приложений, так и количестве их зависимостей. Более того, во время разработки сборку необходимо выполнять постоянно, как вручную, так и автоматически при внесении изменений в исходный код для проверки работоспособности измененной системы.

Использовать данную систему будут как другие разработчики Банка России во время выполнения своих повседневных задач, так и автоматические системы сборки на сервере.

4.2 Оценка коммерческого потенциала и перспективности разработки с позиции ресурсоэффективности и ресурсосбережения

4.2.1 Потенциальные области применения

В проекте, над которым в данный момент работает наша команда, для сборки приложения и управления его зависимостями используется Apache Maven. Для большинства типов проектов существуют проверенные временем и хорошо работающие подходы к структурированному и понятному конфигурированию Maven, соответственно, выгоднее бы было использовать уже готовые подходы к построению комплексной системы сборки большого проекта.

Для этого было решено провести анализ существующих на рынке подходов к организации систем разных типов. В ходе анализа рынка было выявлено, что ключевые отличия при конфигурировании проектов возникают из-за различий в их архитектуре и размере. По этим параметрам было проведено сегментирование рынка с выявлением наиболее распространенных подходов в

каждом из сегментов. На основе проведенного анализа была построена карта сегментирования подходов к конфигурированию, представленная в таблице 2.

Таблица 2 – Карта сегментирования рынка подходов к конфигурированию приложений на основе Apache Maven

Наиболее распространенные подходы к конфигурированию maven-приложений		Архитектура приложения		
		Монолитное приложение	Многомодульное приложение	Микросервисное приложение
Размер проекта	Малый			
	Средний			
	Большой			
	Стандартное конфигурирование Maven			
	Стандартное конфигурирование Maven для каждого микросервиса			
	Иерархическое применение агрегации более мелких модулей			
	Иерархическое применение агрегации с промежуточным выделением BOM спецификаций			

Проект, над которым ведется работа, является достаточно большим многомодульным приложением, и, как видно из таблицы 2, для данного сегмента рынка по сути отсутствуют какие-либо хорошо зарекомендовавшие себя

подходы, которые способны дать четкую наглядную структуру конфигурирования.

В связи с этим и было решено разработать единую унифицированную структуру многомодульных проектов на основе Apache Maven, ориентированную на средние и большие приложения (в нашем проекте количество конфигурационных файлов, соответствующих модулям и их подмодулям, составляет 70-80 штук).

4.2.2 Анализ уже существующих подходов

Для того, чтобы создать наиболее эффективную модель конфигурирования, нужно понять сильные и слабые стороны других подходов. Для этого была построена оценочная карта конкурентных подходов, представленная в приложении И, на которой сравниваются следующие подходы:

1. Стандартное конфигурирование Maven в одном файле;
2. Иерархическое применение агрегации более мелких модулей;
3. Иерархическое наследование подмодулей от основных модулей;
4. Комбинированные решения, в том числе включающие в себя перечисленные выше подходы.

$Б_i$ и $К_i$ в строке j таблицы в приложении И показывают баллы, полученные подходом i в критерии j, и конкурентоспособность, рассчитанную на основе проставленных баллов, соответственно. В последней строке рассчитана сумма всех полученных выше показателей для каждого отдельного столбца.

Огромное преимущество разработки программного обеспечения заключается в том, что очень многие решения распространяются совершенно бесплатно: либо под лицензиями, которые позволяют использовать их в коммерческих целях, либо же и вовсе в качестве информационных статей от других разработчиков, которые не имеют никакой лицензии и подразумевают под собой свободное использование кем угодно.

Как видно из расчетов в таблице из приложения И, наиболее конкурентоспособным оказалось комбинированное решение: 3,54 из 5. В связи с этим было принято решение строить систему сборки именно на комбинированном подходе, выбирая наиболее оптимальные решения для каждой отдельной подзадачи.

Основным преимуществом комбинированного подхода является то, что он отлично подходит для многомодульных проектов: агрегация подмодулей позволяет производить все необходимые операции для всех подмодулей сразу, наследование позволяет избавляться от дублирования одинаковых частей конфигурации в разных модулях, а стандартная структура конфигурационных файлов в любом случае будет встречаться в каждом из них, т.к. это по сути правила, по которым они написаны.

4.2.3 Технология QuaD

Перед началом разработки необходимо задаться вопросом – будет ли вообще данная разработка полезна целевому потребителю, или же средства, которые планируется потратить на разработку, не принесут желаемого результата и качество останется на недостаточно высоком уровне.

Технология QuaD (Quality Advisor) позволяет оценить качество разрабатываемого продукта с точки зрения наиболее важных для целевой аудитории параметров. Основным выходным критерием данной методики является итоговое средневзвешенное значение, которое представляет собой предполагаемый процент реализации качественных характеристик продукта от идеального продукта, который бы полностью удовлетворял потребителя и заказчика.

Как видно из приложения К, средневзвешенное значение продукта оказалось равным 73,7 из 100. Такое значение попадает под категорию «качество выше среднего», что является достаточно хорошим результатом и свидетельствует о том, что разработка будет являться не пустой тратой времени.

4.2.4 SWOT-анализ

SWOT-анализ позволяет наглядно оценить сильные (S – Strengths) и слабые (W – Weaknesses) стороны продукта, открытые перед ним возможности (O – Opportunities), а также возможные угрозы (T – Threats), которые стоят перед проектом. Данная методика применяется для комплексного анализа внутренней и внешней среды проекта и состоит из трех основных этапов.

Первый этап.

На первом этапе оценки проекта по методике SWOT, необходимо выделить наиболее важные для проекта характеристики внутренней и внешней среды, которые и зашифрованы в названии методики. Для разрабатываемой системы сборки и управления зависимостями многомодульных enterprise-приложений на Java были выявлены такие характеристики. Они разбиты по категориям и приведены в таблице 3.

Таблица 3 – Определение характерных особенностей проекта по методике SWOT-анализа

	Положительное влияние	Отрицательное влияние
Внутренняя среда	Сильные стороны проекта: С1. Имеется очевидная необходимость в разработке; С2. Значительное упрощение версионирования проекта; С3. Значительное упрощение выполнения рутинных операций сборок для разработчиков.	Слабые стороны проекта: Сл1. Невозможность изменения структуры проекта из-за необходимости сохранять историю изменений; Сл2. Необходимость поддержки совместимости с уже существующей инфраструктурой проекта; Сл3. Необходимость сотрудника при внесении изменений делать это согласно разработанной схеме.
Внешняя среда	Возможности: В1. Внутренняя разработка - возможность проектировать под любую удобную версию Apache Maven; В2. Потенциальное устранение сложных в обнаружении ошибок, связанных с использованием одних и тех же зависимостей разных версий; В3. Обобщение и документирование подхода для использования в других новых проектах компании.	Угрозы: У1. Возможная скорая переработка целевого проекта – изменение / смена структуры; У2. Инструменты, выбранные для решения отдельных подзадач, могут не корректно работать в таком большом проекте; У3. Появление новых ошибок в результате переработки системы.

Второй этап.

На втором этапе проводится попарное сопоставление категорий из верхней и нижней строк таблицы, полученной на первом этапе. Для каждой полученной пары строится отдельная интерактивная матрица, позволяющая определить комбинации взаимосвязей отдельных характеристик между категориями.

Результаты сопоставления приведены в таблицах 4-7. Знаком «+» отмечена корреляция соответствующих характеристик, знаком «-» – ее отсутствие, а цифрой «0» - затруднение в ее определении.

Помимо простого сопоставления, на данном этапе также сразу определяется связь сопоставленных элементов и вырабатывается решение о том, как эту ситуацию использовать с пользой для проекта.

Таблица 4 – Интерактивная матрица сильных сторон проекта и его возможностей

		Сильные стороны проекта		
		С1	С2	С3
Возможности проекта	В1	0	+	0
	В2	0	-	-
	В3	+	-	-

Таблица 5 – Интерактивная матрица слабых сторон проекта и его возможностей

		Слабые стороны проекта		
		Сл1	Сл2	Сл3
Возможности проекта	В1	-	+	-
	В2	-	-	0
	В3	-	-	0

Таблица 6 – Интерактивная матрица сильных сторон проекта и угроз, существующих для него

		Сильные стороны проекта		
		С1	С2	С3
Угрозы проекту	У1	+	-	-
	У2	-	+	-
	У3	-	-	-

Таблица 7 – Интерактивная матрица слабых сторон проекта и угроз, существующих для него

		Слабые стороны проекта		
		Сл1	Сл2	Сл3
Угрозы проекту	У1	-	0	-
	У2	-	-	-
	У3	-	+	-

Третий этап.

На третьем этапе заполняется итоговая таблица с результатами SWOT-анализа. Результаты анализа для разрабатываемой системы сборки и управления зависимостями многомодульных enterprise-приложений на Java представлены в приложении Л.

Приведенная таблица в наглядном виде позволяет лучше понять, как наиболее выгодно применять факторы внешней среды в проекте и на какие из них следует обратить особо пристальное внимание во избежание негативного влияния на проект, или по крайней мере его уменьшения.

Таким образом, результаты SWOT-анализа показали, что все же стоит попробовать применить в разрабатываемом решении возможности новых версий Maven, так как они могут дать существенную выгоду при переработке версионирования приложения, к тому же инфраструктурный вопрос в данном случае решаем. Однако вносить абсолютно любые изменения в систему сборки следует с особой осторожностью, ведь данная часть проекта является достаточно важной и способна нанести большой ущерб команде разработки из-за невозможности в случае ошибки пользоваться жизненно необходимой в повседневной работе системы.

4.3 Определение основных направлений разработки

В ходе анализа системы для оценки ее коммерческого потенциала и перспективности было выявлено, что перспективность разработки выше среднего. В связи с этим было принято решение о продолжении разработки.

При дальнейшем анализе системы с помощью технологий QuaD и SWOT, было выделено три приоритетных направления разработки, которые дадут наибольший выхлоп от инвестированных в проект ресурсов:

1. Максимальная автоматизация сборки приложения;
2. Смена системы версионирования целевого приложения;
3. Актуализация зависимостей и стандартизация их объявлений.

Помимо этого, SWOT-анализ позволил обратить внимание на необходимость тщательнейшего тестирования решения в различных сценариях использования перед непосредственным введением в эксплуатацию.

4.4 Планирование работ по реализации

4.4.1 Структура работ

После принятия решения о необходимости разработки и определения ключевых направлений деятельности следующим этапом является формирование комплекса необходимых работ, установка очередности их выполнения, а также определения списка лиц, задействованных в каждой задаче.

В приложении М приведен список основных этапов по разработке системы сборки и управления зависимостями многомодульных enterprise-приложений на Java, работы, необходимые на каждом из этих этапов, а также перечислены должности исполнителей каждой отдельной работы.

4.4.2 Определение трудоемкости выполнения работ

Каждую из перечисленных на предыдущем этапе работ необходимо оценить на продолжительность выполнения для дальнейшего детального планирования. Не стоит забывать о таком когнитивном искажении, присущем любому человеку, как ошибка планирования. Для ее минимизации каждой работе дается две оценки: минимальная и максимальная, но и это не гарантирует реализацию проекта в срок.

Расчёт трудоемкости приведен в приложениях Н и О. Условные обозначения, применяемые в них:

- $t_{min i}$ – минимальная трудоемкость задачи в человеко-днях;
- $t_{max i}$ – максимальная трудоемкость задачи в человеко-днях;
- $t_{ож i}$ – ожидаемая трудоемкость задачи, некоторое усредненное значение максимальной и минимальной трудоемкостей;
- $T_{р i}$ – продолжительность задачи (учитывается количество задействованных лиц) в рабочих днях;
- $T_{к i}$ – продолжительность задачи в календарных днях;

- Н. 1, 2 и т.д. – приоритетное направление разработки № 1, 2 и т.д. соответственно (см. пункт 4.2).

Как видно из последней строки «Итого» приложения О, итоговая оценка трудоемкости проекта составила от 74 до 109 календарных дней в зависимости от приоритетного направления разработки. Стоит отметить, что в целях минимизации ошибки планирования как минимальные, так и максимальные оценки (приложение Н) были даны с небольшим запасом.

4.4.3 Составление графика разработки

Для более тщательного планирования предстоящих работ очень полезно иметь их наглядное представление в календарном плане. Это способствует лучшему восприятию масштаба отдельных задач и дает возможность оценить всю разработку целиком.

При расчете трудоемкости выполнения работ было выявлено, что третье направление (актуализация зависимостей и стандартизация их объявлений) обладает максимальной трудоемкостью из всех трех ключевых направлений разработки. На основе данного наименее благоприятного по времени выполнения решения была построена диаграмма Ганта, представленная в приложении П.

Из полученной диаграммы видно, что из-за всего одного разработчика достичь параллельности работ практически нельзя, в связи с чем весь пласт работ является по большей части линейным, а состав работ изначально спланирован для водопадной модели разработки.

4.4.4 Бюджет разработки

Наиболее важной частью предварительного этапа работ является оценка бюджета разработки, т.к. при недостаточном финансировании не будет никакого смысла начинать разработку. Также важно заранее выбрать наиболее оптимальный вариант для текущего бюджета и желаемых результатов. При оценке бюджета часто принято разделять затраты по следующим категориям:

- Материальные затраты;

- Затраты на специальное оборудование;
- Основная заработная плата исполнителей;
- Отчисления во внебюджетные фонды;
- Затраты на командировки;
- Контрагентные расходы;
- Накладные расходы.

Это достаточно общий список и далеко не все из этих категорий присутствуют в той или иной разработке, особенно это касается разработки программного обеспечения, где взаимодействие с физическим миром минимально. Далее расписаны каждая из перечисленных выше категорий расходов при разработке системы сборки и управления зависимостями многомодульных enterprise-приложений на Java.

Материальные затраты, затраты на специальное оборудование, затраты на командировки, контрагентные расходы.

Данные категории расходов отсутствуют при разработке исследуемой системы, поэтому в дальнейшей оценке бюджета разработки учитываться не будут.

Основная заработная плата исполнителей.

Сразу стоит отметить, что непосредственная заработная плата каждой категории сотрудников очень условная и основана лишь на личных предположениях и усредненных рыночных зарплатах в иных организациях. Реальная заработная плата каждого отдельно взятого сотрудника индивидуальна и зависит от множества факторов. Эта категория информации относится к персональным данным сотрудника.

Учитывая все вышесказанное, были произведены ориентировочные расчеты основной заработной платы всех исполнителей, вовлеченных в процесс разработки. Результаты расчетов приведены в таблице 8.

Таблица 8 – Основная заработная плата исполнителей

Исполнители	**$З_{тс}$, руб.**	**$З_{м}$, руб.**	**$З_{дн}$, руб.**	**$Т_{р}$, раб. дн.**			**$З_{осн}$, руб.**		
				Н. 1	**Н. 2**	**Н. 3**	**Н. 1**	**Н. 2**	**Н. 3**
Начальник отдела	120000	202800	10920	1	1	1	14504	14504	14504
Лидер команды разработки проекта	100000	169000	9100	4	4	4	37604	37604	39394
Программный архитектор	100000	169000	9100	1	1	1	8058	8058	9849
Другие члены команды	80000	135200	7280	6	6	12	45124	45124	88100
Разработчик	20000	33800	1820	88	71	104	160889	129195	189719
Итого							**266179**	**234485**	**341566**

Условные обозначения, используемые в таблице:

- $З_{тс}$ – заработная плата по тарифной ставке;
- $З_{м}$ – месячный должностной оклад работника;
- $З_{дн}$ – среднедневная заработная плата работника;
- $Т_{р}$ – продолжительность работ, приходящаяся на всю категорию исполнителей;
- $З_{осн}$ – основная заработная плата;
- Н. 1, 2 и т.д. – приоритетное направление разработки № 1, 2 и т.д. соответственно (см. пункт 4.3).

Дополнительная заработная плата исполнителей.

Дополнительная заработная плата рассчитывается на основе основной заработной платы и предназначена для оплаты отпусков, переработок и пр. Результаты расчётов дополнительной заработной платы приведены в таблице 9.

Таблица 9 – Дополнительная заработная плата исполнителей

Исполнители	**Здоп, руб.**		
	Н. 1	**Н. 2**	**Н. 3**
Начальник отдела	1741	1741	1741
Лидер команды разработки проекта	4512	4512	4727
Программный архитектор	967	967	1182
Другие члены команды	5415	5415	10572
Разработчик	19307	15503	22766
Итого	**31942**	**28138**	**40988**

Отчисления во внебюджетные фонды.

К данной статье расходов относят отчисления на обязательное медицинское страхование, в пенсионный фонд, в фонд социального страхования и т.д. Результаты расчётов приведены в таблице 10.

Таблица 10 – Отчисления во внебюджетные фонды

Исполнители	**Зосн, руб**			**Здоп, руб**		
	Н. 1	**Н. 2**	**Н. 3**	**Н. 1**	**Н. 2**	**Н. 3**
Начальник отдела	14504	14504	14504	1741	1741	1741
Лидер команды разработки проекта	37604	37604	39394	4512	4512	4727
Программный архитектор	8058	8058	9849	967	967	1182
Другие члены команды	45124	45124	88100	5415	5415	10572
Разработчик	160889	129195	189719	19307	15503	22766
Коэффициент отчислений во внебюджетные фонды	0,3					
Итого						
Направление 1						**89436**
Направление 2						**78787**
Направление 3						**114766**

Накладные расходы.

К накладным расходам относятся все остальные расходы, которые не были учтены ранее: электроэнергия, водоснабжение, водоотвод, мелкие канцелярские расходы и пр. Накладные расходы считаются как некоторый процент от всех остальных расходов (в данном случае берется 16%). Накладные расходы для разработки системы сборки и управления зависимостями многомодульных enterprise-приложений на Java приведены в таблице 11.

Таблица 11 – Накладные расходы

Наименование статьи	**Величина расходов, руб.**		
	Напр. 1	**Напр. 2**	**Напр. 3**
Основная ЗП	266179	234485	341566
Дополнительная ЗП	31942	28138	40988
Отчисления во внебюджетные фонды	89436	78787	114766
Накладные расходы	**62009**	**54626**	**79571**

Формирование бюджета затрат разработки.

Ниже приведена сводная таблица по всем рассчитанным ранее статьям.

Таблица 12 – Бюджет затрат разработки

Наименование статьи	**Сумма, руб.**		
	Напр. 1	**Напр. 2**	**Напр. 3**
Основная ЗП	266179	234485	341566
Дополнительная ЗП	31942	28138	40988
Отчисления во внебюджетные фонды	89436	78787	114766
Накладные расходы	62009	54626	79571
Бюджет разработки	**449566**	**396036**	**576891**

Из анализа таблицы 12 видно, что приоритезация актуализации зависимостей и стандартизация их объявлений чревата превышением бюджета разработки практически на сто тысяч рублей, а смена системы версионирования целевого приложения является наиболее экономически целесообразной для проекта.

4.5 Определение ресурсной (ресурсосберегающей), финансовой, бюджетной, социальной и экономической эффективности исследования

Данный этап направлен на финальное определение эффективности проработки возможных направлений разработки для выбора наиболее оптимального. Для этих целей хорошо подходит интегральный показатель эффективности разработки, который рассчитывается на основе интегральных финансового показателя (таблица 13) и показателя ресурсоэффективности (приложение Р).

Таблица 13 – Интегральный финансовый показатель

Показатель	Напр. 1	Напр. 2	Напр. 3
Стоимость	449566	396036	576891
Интегральный финансовый показатель	0,90	0,79	1,15

Интегральный финансовый показатель показывает, во сколько раз разработка превысила допустимый бюджет (или удешевила разработку).

Интегральный показатель ресурсоэффективности приведен в приложении Р и показывает, насколько эффективно были / будут вложены средства в проект за счет оценки степени соответствия ключевым критериям оценки (в расчётах использована пятибалльная шкала).

Интегральный же показатель эффективности направлений разработки находится как отношение интегрального показателя ресурсоэффективности к интегральному финансовому показателю для каждого из направлений разработки. Результаты расчётов приведены в таблице 14.

Таблица 14 - Интегральный показатель эффективности направлений разработки

Интегральный показатель	Напр. 1	Напр. 2	Напр. 3
Ресурсоэффективности	3,00	3,66	2,69
Финансовый	0,90	0,79	1,15
Эффективности	**3,34**	**4,62**	**2,33**

Как видно из таблицы 14, наивысшим интегральным показателем эффективности обладает второе приоритетное направление разработки - смена системы версионирования целевого приложения. Для более наглядного сравнения направлений разработки можно рассчитать сравнительную эффективность направлений разработки относительно второго, как самого эффективного.

Результаты расчета сравнительной эффективности направлений разработки отображены в сводной таблице 15, в которой также приведены и все остальные интегральные показатели эффективности.

Таблица 15 – Сравнительная эффективность разработки

№ п/п	Показатели	Напр. 1	Напр. 2	Напр. 3
1	Интегральный финансовый показатель разработки	0,90	0,79	1,15
2	Интегральный показатель ресурсоэффективности разработки	3,00	3,66	2,69
3	Интегральный показатель эффективности	3,34	4,62	2,33
4	Сравнительная	0,72	1,00	0,50

Смена системы версионирования целевого приложения также имеет и остальные наибольшие интегральные показатели. Это свидетельствует о том, что, выбрав данное направление в качестве приоритетного при разработке системы, сама разработка получится наиболее дешевой, а ее результаты будут в большей степени соответствовать заданным критериям эффективности.

Если посмотреть на сравнительную эффективность направлений разработки (таблица 15), то выбор смены системы версионирования целевого приложения в качестве приоритетного направления разработки будет в два раза эффективнее фокусировки на актуализации зависимостей и стандартизации их объявлений.

4.6 Выводы по разделу

В данном разделе разрабатываемая система сборки и управления зависимостями многомодульных enterprise-приложений на Java была рассмотрена со множества критически важных для бизнеса сторон.

Прежде всего, был проведен анализ существующих на рынке решений, которые можно бы было использовать для достижения поставленных задач, в ходе которого было выявлено, что удовлетворяющих в полной мере подходов к решению поставленных задач по сути нет.

Выявленные ключевые критерии качества решения позволили лучше понять, какие направления разработки наиболее важны для проекта, а их первоначальная оценка дала понимание сильных и слабых сторон разрабатываемого решения, что было использовано для наиболее эффективного принятия решений по развитию проекта.

Благодаря проведенному анализу удалось выявить три наиболее перспективных приоритетных направления разработки, которые были тщательно проанализированы со множества позиций: трудоемкости работ по ним, предполагаемых бюджетов на разработку, вовлеченных в процесс разработки исполнителей, ключевых этапов разработки и предполагаемого графика работ.

В ходе сравнения результатов анализа было установлено, что наиболее эффективным направлением разработки является смена системы версионирования целевого приложения, а вторым по приоритетности направлением – максимальная автоматизация сборки приложения.

Более того, анализ установил, что приоритезация актуализации зависимостей и стандартизации их объявлений с большой долей вероятности может повлечь за собой превышение бюджетов разработки, что также делает ее наименее эффективной (практически в 2 раза менее эффективной, чем смена системы версионирования целевого приложения) среди исследованных направлений разработки.

5 Социальная ответственность

5.1 Введение

В работе рассматривается программное обеспечение, позволяющее централизовать управление зависимостями, а также автоматизировать сборку крупных корпоративных приложений, разрабатываемых и применяемых в Центральном Банке Российской Федерации.

Актуальность разработки обусловлена большими масштабами разрабатываемых приложений, процесс сборки которых усложняется при росте как самих приложений, так и количестве их зависимостей. Более того, во время разработки сборку необходимо выполнять постоянно, как вручную, так и автоматически при внесении изменений в исходный код для проверки работоспособности измененной системы.

Использовать данную систему будут как другие разработчики Банка России во время выполнения своих повседневных задач, так и автоматические системы сборки на сервере.

5.2 Правовые и организационные вопросы обеспечения безопасности

В первую очередь социальная защищенность сотрудника регламентируется Трудовым кодексом Российской Федерации (в последней редакции на момент написания раздела от 24.04.2020) [12]. ТК РФ охватывает обширные области обязанностей сторон, заключивших трудовой договор: работодателя и работника.

Так, при разработке программного обеспечения могут возникать непредвиденные обстоятельства выхода программного обеспечения из строя в среде эксплуатации, в связи с чем разработчики могут быть привлечены к сверхурочным работам, направленным на ликвидацию инцидента. Согласно статье 99 Трудового кодекса подобное привлечение сотрудника допускается с его письменного согласия, а, согласно статье 152, первые 2 часа сверхурочной

работы должны оплачиваться не менее чем в полуторном размере, а за последующие часы – не менее чем в двойном размере.

Разработчики программного обеспечения по сути являются офисными работниками, поэтому организация рабочего места разработчика регламентируется соответствующими нормативными документами.

Поскольку разработчик проводит на своем рабочем месте по 8 часов в день (при полной занятости), согласно СанПиН 2.2.4.3359-16 [13], в помещении, где располагается рабочее место, должно присутствовать естественное освещение.

На данный момент в организации выведены из эксплуатации и заменены на жидкокристаллические все мониторы на электро-лучевой трубке. В связи с этим, согласно СанПиН 2.2.2/2.4.1340-03 [14], площадь рабочего места для каждого сотрудника должна быть не менее 4,5 квадратных метров, а также помещение должно регулярно проветриваться. При этом расстояние между фронтальными поверхностями мониторов сотрудников должно быть не менее 2 метров, а между боковыми поверхностями – не менее 1,2 метра.

Эргономические характеристики к рабочему месту перечислены в ГОСТ 12.2.032-78 ССБТ [15], а также в ГОСТ 22269-76 [16]. Они определяют расположение предметов относительно сотрудника, высоты рабочих поверхностей и прочие характеристики, которые напрямую влияют на комфорт и, соответственно, на концентрацию во время работы.

5.3 Производственная безопасность

Несмотря на то, что разработчики являются офисными работниками, в их повседневных условиях труда все равно присутствуют факторы, являющиеся опасными или вредными. Они не столь очевидны, как у рабочих, скажем, трудящихся в цехах, однако все равно присутствуют и могут наносить вред здоровью сотрудников, а также снижать их работоспособность. В ходе выполнения работы был выявлен ряд опасных и вредных факторов, которые

могут присутствовать на рабочем месте разработчика. Они представлены в сводной таблице 16.

Таблица 16 – Возможные опасные и вредные факторы

Факторы (ГОСТ 12.0.003-2015)	Этапы работ		Нормативные документы
	Разработка	Эксплуатация	
1. Отклонение показателей микроклимата помещений	+	+	СанПиН 2.2.4.548-96 [17]
2. Отсутствие или недостаток освещения на рабочем месте	+	+	СП 52.13330.2016 [18]
3. Умственное перенапряжение, в том числе вызванное информационной нагрузкой	+	+	Р 2.2.2006-05 [19]
4. Монотонность труда	+	+	Р 2.2.2006-05 [19]

Разрабатываемая система является программным обеспечением, облегчающим разработку другого программного обеспечения в одних и тех же условиях. Данный факт свидетельствует о том, что выявленные возможные и опасные факторы будут действительны как для этапа разработки системы, так и для этапа эксплуатации.

5.3.1 Анализ выявленных вредных и опасных факторов

Отклонение показателей микроклимата помещений.

Согласно СанПиН 2.2.4.548-96, микроклимат в производственных помещениях характеризуют такие показатели, как:

- Температура воздуха;
- Температура поверхностей;
- Относительная влажность воздуха;
- Скорость движения воздуха;
- Интенсивность теплового облучения.

Достаточно важно удерживать данные показатели в рамках оптимальных микроклиматических условий, так как такие условия создают плодотворную среду для повышенной работоспособности персонала, что делает их крайне

предпочтительными в рабочих помещениях. Несоблюдение оптимальных микроклиматических условий влечет за собой понижение работоспособности персонала, что негативно сказывается на производственном процессе.

В таблицах 17 и 18 приведены допустимые величины наиболее распространенных показателей микроклимата на рабочих местах в производственных помещениях. Прочие показатели в данной работе не рассматривались ввиду неактуальности для офисных помещений.

Таблица 17 – Допустимые величины показателей микроклимата на рабочих местах производственных помещений для температуры воздуха и поверхностей

Период года	Категория работ по уровню энергозатрат, Вт	Температура воздуха, °С		Температура поверхностей, °С
		Диапазон ниже оптимальных величин	Диапазон выше оптимальных величин	
Холодный	Iа (до 139)	20,0-21,9	24,1-25,0	19,0-26,0
	Iб (140 - 174)	19,0-20,9	23,1-24,0	18,0-25,0
	IIа (175 - 232)	17,0-18,9	21,1-23,0	16,0-24,0
	IIб (233 - 290)	15,0-16,9	19,1-22,0	14,0-23,0
	III (более 290)	13,0-15,9	18,1-21,0	12,0-22,0
Теплый	Iа (до 139)	21,0-22,9	25,1-28,0	20,0-29,0
	Iб (140 - 174)	20,0-21,9	24,1-28,0	19,0-29,0
	IIа (175 - 232)	18,0-19,9	22,1-27,0	17,0-28,0
	IIб (233 - 290)	16,0-18,9	21,1-27,0	15,0-28,0
	III (более 290)	15,0-17,9	20,1-26,0	14,0-27,0

Таблица 18 – Допустимые величины показателей микроклимата на рабочих местах производственных помещений для скорости движения воздуха и его относительно влажности

Период года	Категория работ по уровню энергозатрат, Вт	Скорость движения воздуха, м/с		Относительная влажность воздуха, %
		Для диапазона температур воздуха ниже оптимальных величин, не более	Для диапазона температур воздуха выше оптимальных величин, не более	
Холодный	Ia (до 139)	0,1	0,1	15 - 75
	Iб (140 - 174)	0,1	0,2	15 - 75
	IIa (175 - 232)	0,1	0,3	15 - 75
	IIб (233 - 290)	0,2	0,4	15 - 75
	III (более 290)	0,2	0,4	15 - 75
Теплый	Ia (до 139)	0,1	0,2	15 - 75
	Iб (140 - 174)	0,1	0,3	15 - 75
	IIa (175 - 232)	0,1	0,4	15 - 75
	IIб (233 - 290)	0,2	0,5	15 - 75
	III (более 290)	0,2	0,5	15 - 75

Работа программиста по уровню энергозатрат относится к категории Ia. Температура воздуха в офисах соответствует допустимым показателям, а также достаточно часто является оптимальной – зимой около 22 градусов, а летом около 24 градусов. Однако, из-за массивных стен, температура окружающих поверхностей временами может быть близка к минимально допустимой. Относительная влажность воздуха практически всегда равна 30%, что является допустимым значением. А вот движение воздуха часто отсутствует.

Для обеспечения рабочей зоны необходимым уровнем вентиляции можно использовать бытовые вентиляторы, а также каждый час осуществлять проветривание помещения.

Отсутствие или недостаток освещения на рабочем месте.

Недостаток освещения крайне негативно сказывается на производительности рабочих, вызывая быструю утомляемость и снижая

производительность труда сотрудников. Более того, разработчики программного обеспечения проводят на своем рабочем месте практически все свое рабочее время. Согласно СП 52.13330.2016, такое рабочее место является помещением с постоянным пребыванием людей.

Размер объектов, которые различает оператор ЭВМ на экране разнятся и чаще всего гибко настраиваются, однако знаки препинания и прочие мелкие символы могут быть равными 0,3-0,5 мм, причем из-за подсветки монитора отнесем текст на экране к светящимся объектам. Контраст объекта с фоном, как и его характеристика как правило тоже настраиваются в используемых редакторах. Учитывая все вышеперечисленное, в таблице 19 приведены сводные характеристики рабочего места разработчика. В таблице 20 приведены требования к освещению помещений промышленных предприятий согласно определенным ранее характеристикам.

Таблица 19 – Сводные характеристики рабочего места разработчика

Характе-ристика зрительной работы	Наименьший или эквивалентный размер объекта различения, мм	Разряд зрительной работы	Подразряд зрительной работы	Контраст объекта с фоном	Характе-ристика фона
Высокой точности	От 0,30 до 0,50	III	в	Большой	Светлый

Таблица 20 – Требования к освещению помещений промышленных предприятий для определенных в таблице 19 характеристик

Искусственное освещение					**Совмещенное освещение**	
Освещенность, лк			**Сочетание нормируемых величин объединенного показателя дискомфорта URG и коэффициента пульсации**		**КЕО, eH, %**	
При системе комбинированного освещения		**При системе общего освещения**	**URG, не более**	**КП, %, не более**	**При верхнем или комбинированном освещении**	**При боковом освещении**
Всего	**В том числе от общего**					
750	200	300	25	15	3,0	1,2

Для снижения влияния фактора следует повысить уровень освещения в соответствии с нормами, организуя перекрывающие друг друга освещенные участки, а также обеспечить рабочие места дополнительными персональными осветительными приборами.

Умственное перенапряжение, в том числе вызванное информационной нагрузкой.

Большую часть своего рабочего времени разработчики занимаются умственным трудом: проектированием решения поставленной задачи, проработкой алгоритмов работы программы, изучением новых инструментов, поиском ошибок в логике программы и прочими интеллектуальными задачами.

Избыток подобных задач в течении дня приводит к появлению нервно-психических перегрузок у рабочего, что может вести к появлению самых разных нейропсихологических заболеваний, а также крайне негативно сказывается на производительности разработчиков.

По ряду признаков нагрузки интеллектуального характера подразделяются на простые и сложные задачи. Разработчику постоянно попадаются совершенно разные задачи, которые могут относиться как к первой, так и ко второй категории.

Для более эффективного выполнения работы разработчику следует по возможности делать регулярные перерывы, а также разбивать комплексные задачи на более мелкие подзадачи.

Монотонность труда.

Монотонность труда представляет из себя постоянно повторяющиеся однотипные последовательности действий. Подобный образ работы со временем приводит к появлению монотонии, что проявляется в повышении утомляемости на работе, снижении интереса к ней. Подобные последствия часто влекут за собой снижение продуктивности и производительности труда.

В руководстве Р 2.2.2006-05 приведена классификация монотонности труда, однако, для разработчика рутинные задачи появляются лишь на определенный срок, или с определенной периодичностью, а не являются постоянным условием труда. В связи с этим нельзя определить класс монотонности работы.

Снизить монотонность труда можно путем автоматизации как можно больших процессов, а также стараясь чередовать разные виды интеллектуальных нагрузок на рабочих.

5.3.2 Обоснование мероприятий по снижению воздействия

Отклонение показателей микроклимата помещений.

В целях поддержания оптимальных показателей микроклимата на рабочем месте следует осуществлять терморегуляцию помещения согласно индивидуальным ощущениям, регулярно (желательно не реже 1 раза в час) проветривать помещение, а также при необходимости использовать увлажнители воздуха.

Отсутствие или недостаток освещения на рабочем месте.

Если предоставляется такая возможность, то следует выбирать рабочее место с боковым и/или верхним естественным освещением. При недостатке искусственного освещения следует снабдить свое рабочее место индивидуальным осветительным прибором, не допуская попадания прямого, или отраженного света в глаза. Также при мерцании или пульсации осветительных приборов следует незамедлительно заменить их на аналоги без с более подходящими характеристиками.

Умственное перенапряжение, в том числе вызванное информационной нагрузкой.

В целях уменьшения умственного перенапряжения следует проводить подробный предварительный анализ с декомпозицией поставленных задач, а также регулярно делать перерывы, желательно со сменой вида деятельности.

Монотонность труда.

Как правило, монотонные процессы с развитием технологий отлично поддаются оптимизации, что как раз наиболее заметно в сфере развития ПО. Поэтому монотонную работу по возможности следует выполнять с помощью автоматических / автоматизированных системам. Кроме того, разработчику следует стараться чередовать разные виды интеллектуальных нагрузок в течении дня, а также делать регулярные перерывы.

5.4 Экологическая безопасность

Разрабатываемая система сборки и управления зависимостями многомодульных enterprise-приложений на Java напрямую не несет вреда окружающей среде, однако разработчики, применяющие ее, также используют во время работы различные канцелярские принадлежности, в обеденные перерывы употребляют пищу, хранимую зачастую в одноразовых упаковочных материалах, а также используют средства личной гигиены. Все это влечет за собой появление различных бытовых отходов, при неправильной утилизации которых происходит загрязнение литосферы.

В ходе своего рабочего дня разработчик сталкивается с появлением преимущественно твердых бытовых отходов. В подавляющем большинстве такие отходы относятся к одной из следующих категорий:

- Бумага (картон);
- Пластик;
- Пищевые (органические) отходы;
- Стекло.

Для снижения воздействия на литосферу для данных категорий отходов предпочтительно организовать раздельный сбор мусора на территории организации.

Бумагу, пластик и стекло предпочтительно отправлять на дальнейшую переработку в качестве вторичного сырья. Органические же отходы прекрасно подвергаются процессу гниения.

5.5 Безопасность в чрезвычайных ситуациях

Основной чрезвычайной ситуацией, которая может возникнуть при эксплуатации разработанной системы сборки и управления зависимостями многомодульных enterprise-приложений на Java является возникновение пожара.

На случай возникновения пожара во всех корпусах Банка России предусмотрены эвакуационные выходы, обозначенные специальными знаками с подсветкой от аварийной сети. Данные выходы также отмечены на планах

эвакуации, которые имеются на каждом этаже любого корпуса. Помимо этого, помещения оборудованы системами автономного пожаротушения.

Все сотрудники регулярно проходят инструктаж по пожарной безопасности под роспись. В целях информирования сотрудников, в каждом помещении имеются информационные таблички с номерами экстренных служб, а также перечислением лиц, ответственных за пожаробезопасность в данном помещении. При ежедневном покидании помещения, помимо соблюдения мер по поддержанию безопасности и сохранности данных, сотрудники также отключают электрические приборы от сети в целях исключения их самовоспламенения.

Более того, все этажи, начиная с третьего, также оборудуются фильтрующими самоспасателями, а на первых этажах предусмотрена возможность открытия изнутри оконных решеток для покидания помещения непосредственно через окно.

5.6 Выводы по разделу

В данном разделе были всесторонне рассмотрены основные вопросы обеспечения безопасности во время эксплуатации разработанной системы сборки и управления зависимостями многомодульных enterprise-приложений на Java. В работе были как рассмотрены вопросы правильного обустройства рабочего места разработчика программного обеспечения, так и проанализированы чрезвычайные ситуации, которые способны возникнуть в офисных рабочих помещениях. Кроме того, были затронуты вопросы загрязнения окружающей среды твердыми бытовыми отходами и потенциальные способы их снижения.

Проведенное исследование в первую очередь нацелено на повышение комфорта разработчиков на протяжении каждого рабочего дня, а также на снижение влияния вредных факторов, таких как недостаточное освещение рабочего места.

Заключение

В ходе выполнения выпускной квалификационной работы были выполнены следующие поставленные задачи:

1. Была выбрана наиболее оптимальная система сборки проекта;
2. Были перепроектированы выявленные проблемные процессы;
3. Перепроектированные процессы были реализованы и протестированы.

В результате выполнения поставленных задач удалось достичь значительного сокращения времени, затрачиваемого разработчиками на такие рутинные, но необходимые операции как смена версии приложения, а также выполнения сборки приложения. Причем переработанная система сборки и управления зависимостями удовлетворяет всем заявленным требованиям, сохранила обратную совместимость с уже существующей инфраструктурой проекта и ее переработка выполнена по наиболее эффективным направлениям модернизации, что позволило выполнить ее переработку с минимальными затратами и максимальной пользой проекту. Все изменения, реализованные в рамках данной работы уже внедрены и активно применяются при разработке функциональной подсистемы «Досье НФО».

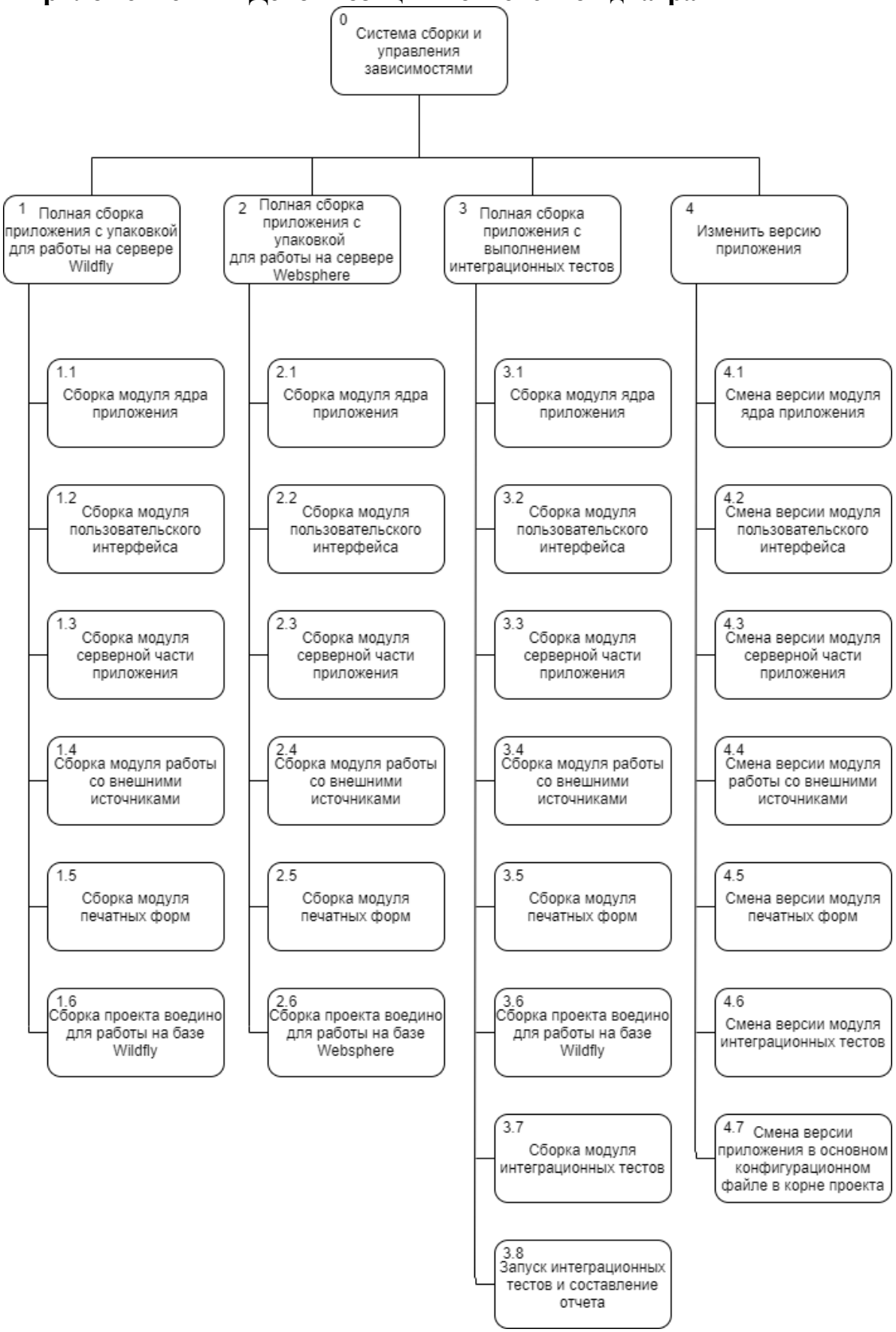
Список используемых источников

1. Steve Loughran, Erik Hatcher Ant in Action. - 2 ed. - Manning, 2007.
2. Balaji Varanasi, Sudha Belida Introducing Maven. - Apress, 2014.
3. Apache Maven Project [Электронный ресурс]. Режим доступа: https://maven.apache.org/, свободный.
4. Хабр. Ant + Ivy VS Maven: давайте жить дружно [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/174721/, свободный.
5. Benjamin Muschko Gradle in Action. - Manning, 2014.
6. Хабр. Многомодульный Java-проект с Gradle. Шаг за шагом [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/225189/, свободный.
7. Gradle. Authoring Multi-Project Builds [Электронный ресурс]. Режим доступа: https://docs.gradle.org/current/userguide/multi_project_builds.html, свободный.
8. Хабр. Идеальный мавен. Часть 2: структура проекта [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/344480/, свободный.
9. Apache Maven Project. Maven CI Friendly Versions [Электронный ресурс]. Режим доступа: https://maven.apache.org/maven-ci-friendly.html, свободный.
10. Majohaus. Versions Maven Plugin [Электронный ресурс]. Режим доступа: https://www.mojohaus.org/versions-maven-plugin/index.html, свободный.
11. Видяев И.Г., Серикова Г.Н., Гаврикова Н.А. ФИНАНСОВЫЙ МЕНЕДЖМЕНТ, РЕСУРСОЭФФЕКТИВНОСТЬ И РЕСУРСОСБЕРЕЖЕНИЕ. - Томск: Издательство Томского политехнического университета, 2014.

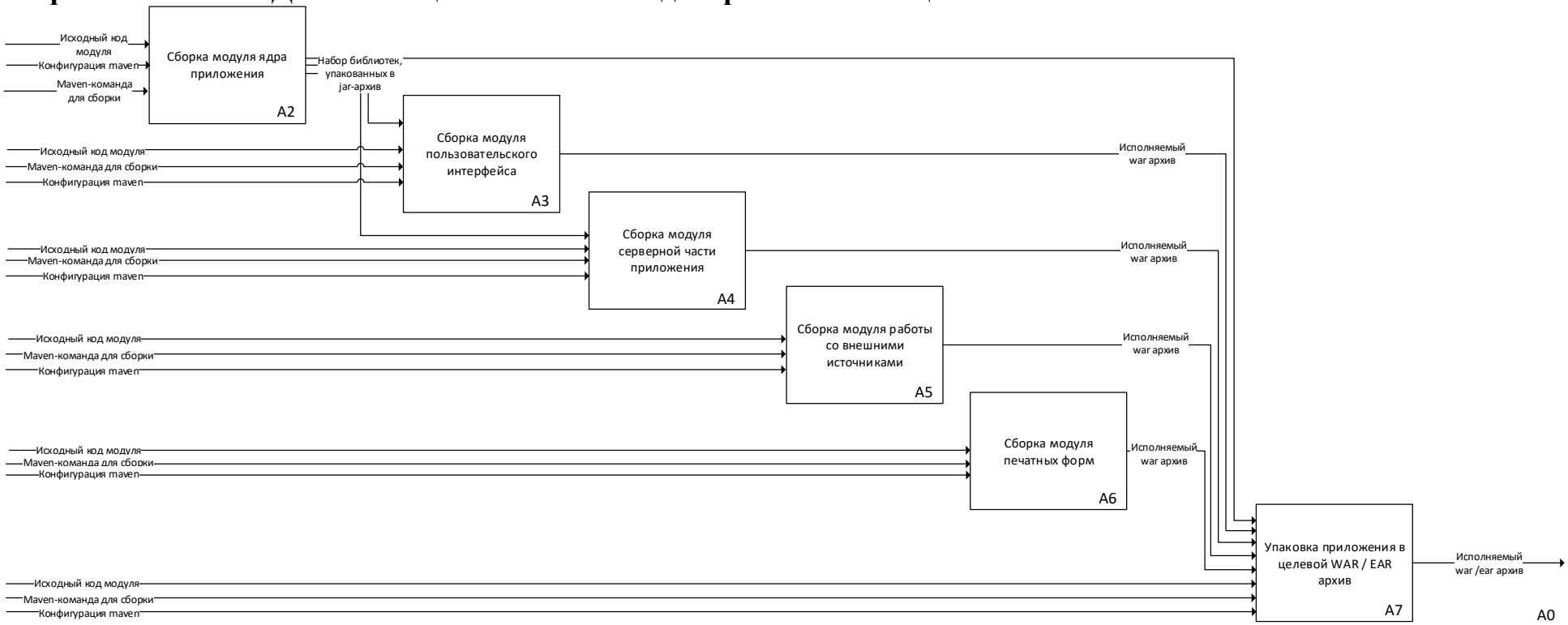
12. Трудовой кодекс Российской Федерации от 30.12.2001 N 197-ФЗ (ред. от 24.04.2020) // Собрание законодательства РФ. - 07.01.2002. - № 1 (ч. 1). - Ст. 3.
13. СанПиН 2.2.4.3359-16 Санитарно-эпидемиологические требования к физическим факторам на рабочих местах, утв. Постановлением Главного государственного санитарного врача РФ 21.06.2016 г.
14. СанПиН 2.2.2/2.4.1340-03. Санитарно-эпидемиологические правила и нормативы «Гигиенические требования к персональным электронно-вычислительным машинам и организации работы». М.: Госкомсанэпиднадзор, 2003.
15. ГОСТ 12.2.032-78 Система стандартов безопасности труда (ССБТ). Рабочее место при выполнении работ сидя. Общие эргономические требования.
16. ГОСТ 22269-76. Система «человек-машина». Рабочее место оператора. Взаимное расположение элементов рабочего места. Общие эргономические требования.
17. СанПиН 2.2.4.548-96. Гигиенические требования к микроклимату производственных помещений. – М.: Минздрав России, 1997.
18. Свод правил СП 52.13330.2016 «Естественное и искусственное освещение». Актуализированная редакция СНиП 23-05- 95*. – М: НИИСФ РААСН и ООО «ЦЕРЕРА-ЭКСПЕРТ», 2017.
19. Р 2.2.2006-05. Руководство по гигиенической оценке факторов рабочей среды и трудового процесса. Критерии и классификация условий труда. – М.: Минздрав России, 2006.

Приложения

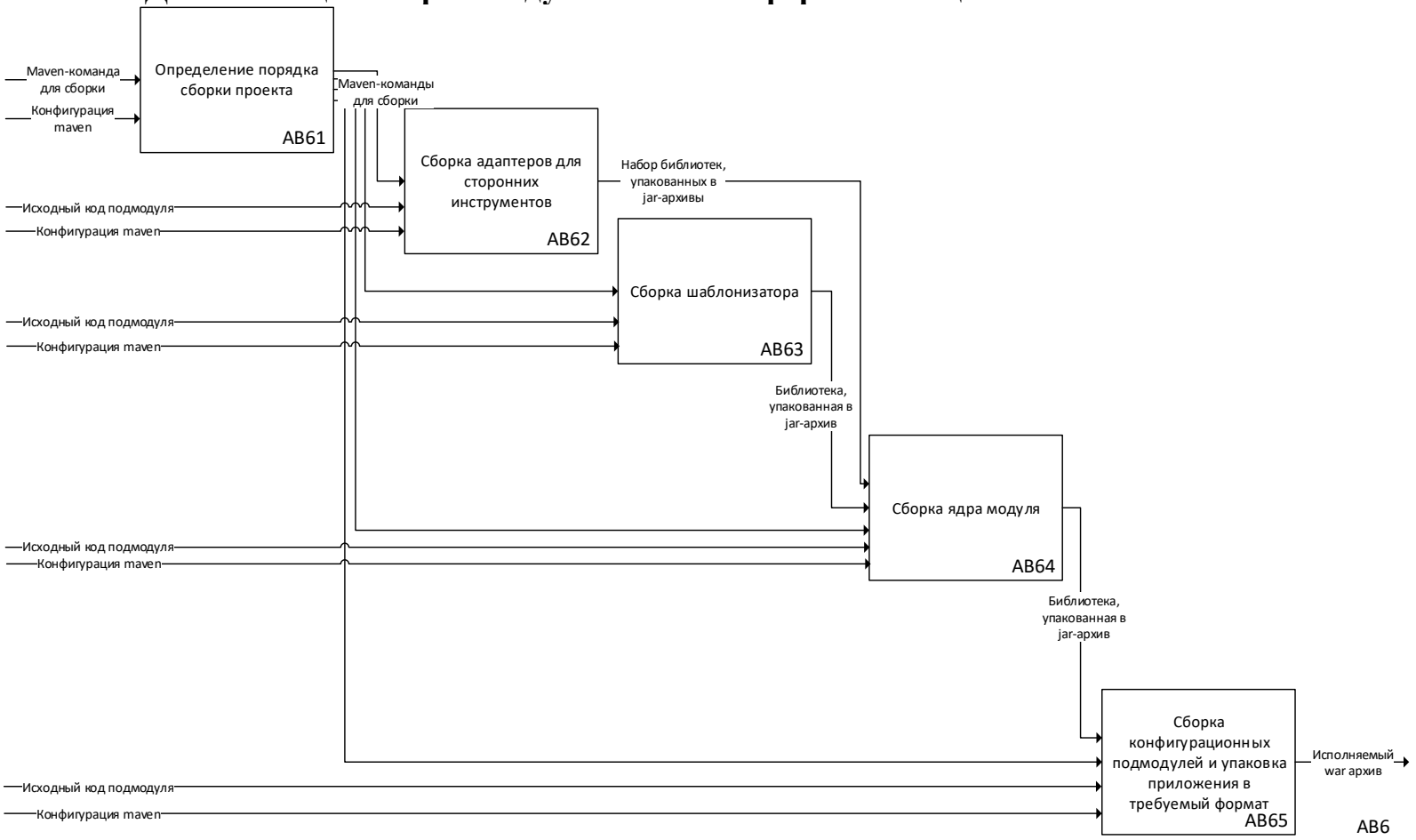
Приложение А - Декомпозиция контекстной диаграммы DFD



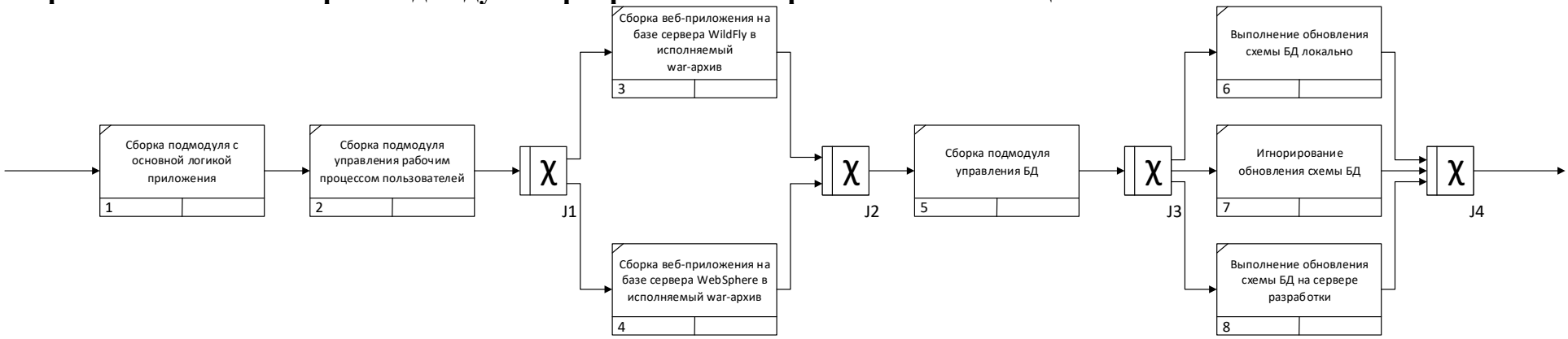
Приложение Б - Декомпозиция контекстной диаграммы в нотации IDEF0



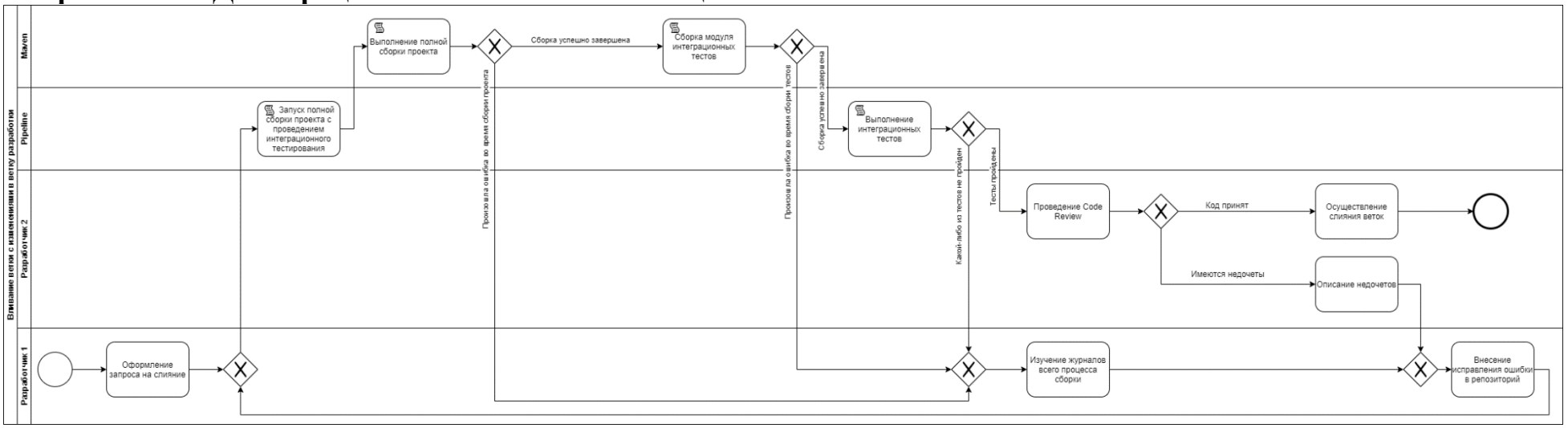
Приложение В - Декомпозиция сборки модуля печатных форм в нотации IDEF0



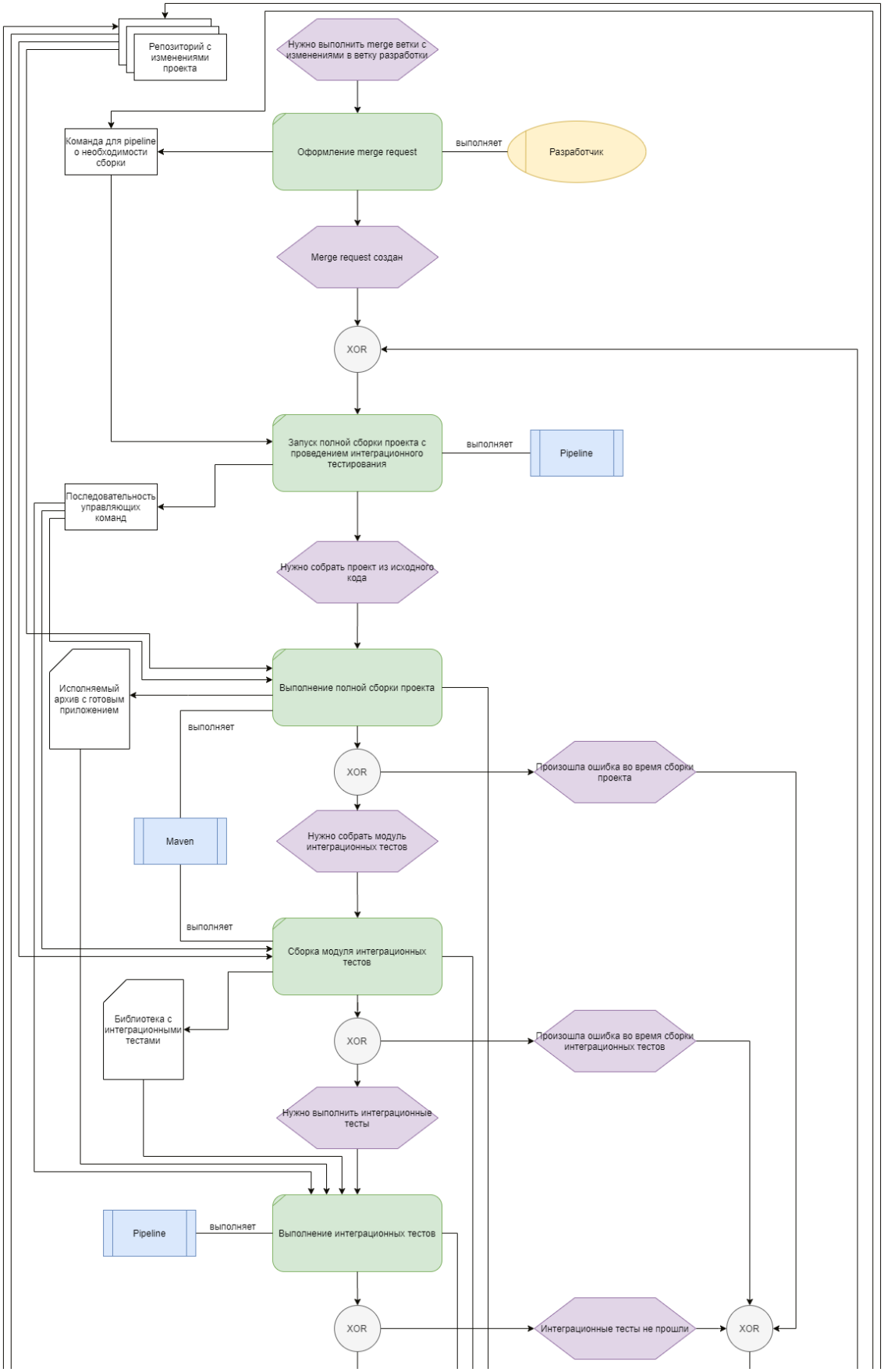
Приложение Г - Сборка подмодуля серверной части приложения в нотации IDEF3



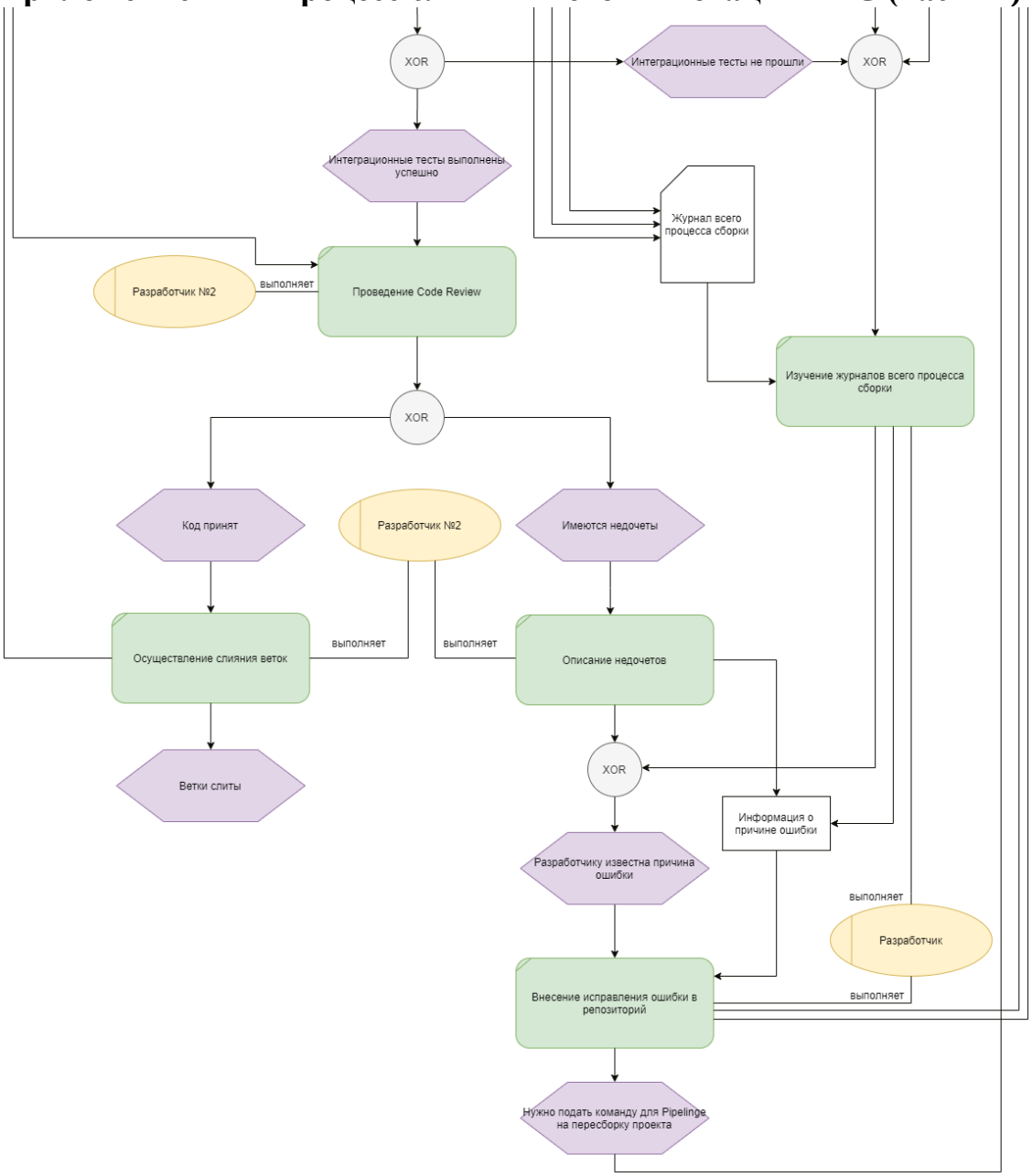
Приложение Д - Процесс слияния веток в нотации BPMN



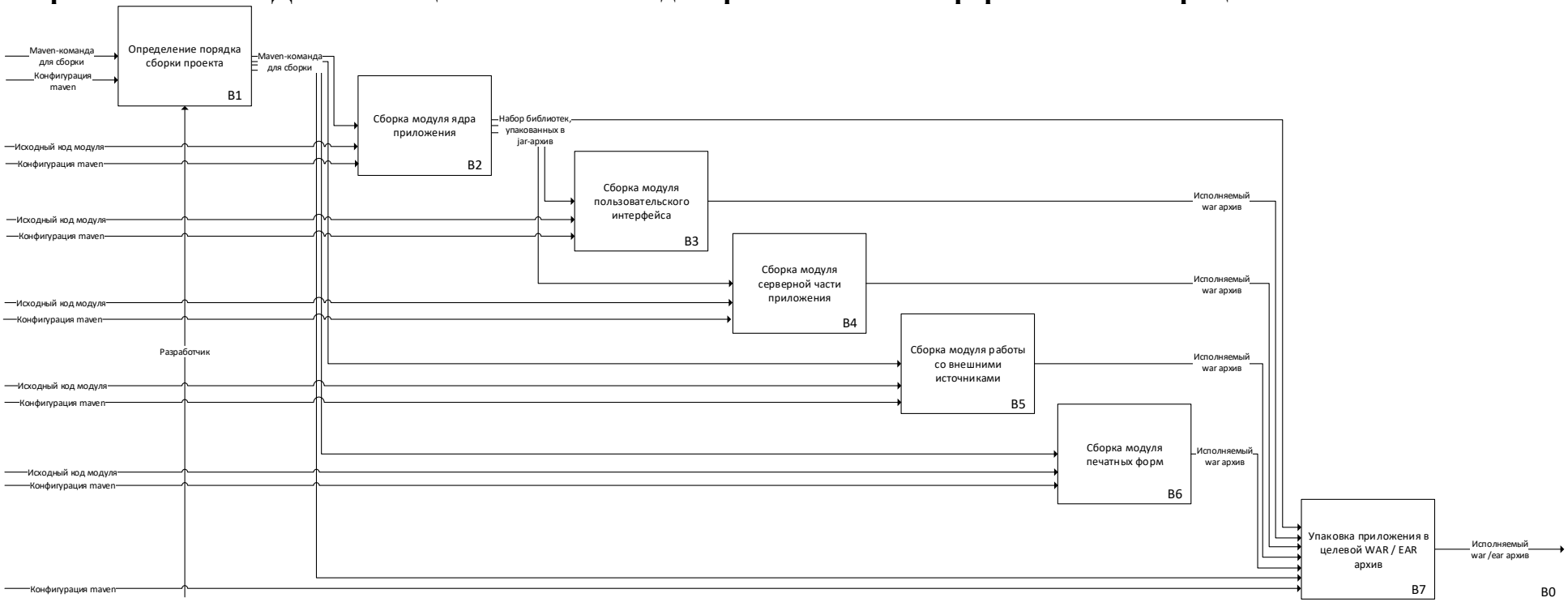
Приложение Е - Процесс слияния веток в нотации EPC (часть 1)



Приложение Ж - Процесс слияния веток в нотации EPC (часть 2)



Приложение З - Декомпозиция контекстной диаграммы IDEF0 переработанного процесса



Приложение И - Оценочная карта для сравнения подходов к решению проблемы

Критерии оценки	Вес критерия	Баллы				Конкурентоспособность			
		$Б_1$	$Б_2$	$Б_3$	$Б_4$	$К_1$	$К_2$	$К_3$	$К_4$
Надежность конфигурации	0,15	5	4	3	2	0,75	0,60	0,45	0,30
Простота алгоритмов обратки конфигурации в Maven	0,04	5	4	3	2	0,20	0,16	0,12	0,08
Наличие примеров реализации и исследований (распространенность)	0,09	5	3	3	2	0,45	0,27	0,27	0,18
Возможность гибкой настройки задач с помощью профилей	0,04	5	5	3	3	0,20	0,20	0,12	0,12
Централизованное управление зависимостями	0,03	5	2	4	3	0,15	0,06	0,12	0,09
Возможность применения в многомодульном проекте	0,32	1	3	3	5	0,32	0,96	0,96	1,60
Удобство чтения конфигурационных файлов	0,04	3	3	2	4	0,12	0,12	0,08	0,16
Простота версионирования приложения	0,12	5	2	2	4	0,60	0,24	0,24	0,48
Простота поддержания структуры в актуальном состоянии	0,09	2	4	2	3	0,18	0,36	0,18	0,27
Простота первичного создания структуры	0,03	4	3	2	2	0,12	0,09	0,06	0,06
Уменьшение дублирования фрагментов конфигурации	0,05	5	2	3	4	0,25	0,10	0,15	0,20
Итого	**1,00**					**3,34**	**3,16**	**2,75**	**3,54**

Примечания:

1. Индекс «1» соответствует подходу «Стандартное конфигурирование Maven в одном файле», «2» - «Иерархическое применение агрегации более мелких модулей», «3» - «Стандартное конфигурирование Maven в одном файле», а «4» - «Стандартное конфигурирование Maven в одном файле»;
2. Бi и Кi в строке j показывают баллы, полученные подходом i в критерии j, и конкурентоспособность, рассчитанную на основе проставленных баллов, соответственно. В последней строке рассчитана сумма всех полученных выше показателей для каждого отдельного столбца.

Приложение К - Оценка качества разработки по технологии QuaD

Критерии оценки	Вес критерия	Баллы	Максимальный балл	Относительное значение	Средневзвешенное значение
Сокращение количества операций при сборке проекта	0,08	60	100	0,6	4,8
Сокращение количества операций при смене версии проекта	0,09	100	100	1	9
Ускорение смены версии проекта	0,09	100	100	1	9
Увеличение прозрачности управления зависимостями	0,08	60	100	0,6	4,8
Устранение дублирования зависимостей	0,09	60	100	0,6	5,4
Устранение необходимости выполнения строгой последовательности сборок для успешного завершения	0,06	60	100	0,6	3,6
Устойчивость к внесению изменений	0,06	50	100	0,5	3
Экономия внимания разработчиков	0,13	100	100	1	13
Экономия времени разработчиков	0,13	100	100	1	13
Возможность применения в других проектах	0,05	60	100	0,6	3
Вероятность отсутствия конфликтов и неисправностей при переработке решения	0,09	40	100	0,4	3,6
Время реализации	0,05	30	100	0,3	1,5
Итого	1				73,7

Приложение Л - Итоговая матрица SWOT-анализа

	Сильные стороны проекта: С1. Имеется очевидная необходимость в разработке; С2. Значительное упрощение версионирования проекта; С3. Значительное упрощение выполнения рутинных операций сборок для разработчиков.	Слабые стороны проекта: Сл1. Невозможность изменения структуры проекта из-за необходимости сохранять историю изменений; Сл2. Необходимость поддержки совместимости с уже существующей инфраструктурой проекта; Сл3. Необходимость сотрудника при внесении изменений делать это согласно разработанной схеме.
Возможности: В1. Внутренняя разработка - возможность проектировать под любую удобную версию Apache Maven; В2. Потенциальное устранение сложных в обнаружении ошибок, связанных с использованием одних и тех же зависимостей разных версий; В3. Обобщение и документирование подхода для использования в других новых проектах компании.	1. Благодаря использованию более современных версий Apache Maven есть возможность применить новые инструменты для еще большего упрощения версионирования проекта. 2. Появление необходимости в переработке системы сборки и управления зависимостями в одном проекте означает, что такая потребность может появиться и в другом проекте, поэтому документирование подхода может принести большую пользу в будущем.	1. Благодаря тому, что вся инфраструктура вместе с самой разработкой переходит внутрь компании, в случае необходимости в возможностях, которые доступны лишь в новых версиях Apache Maven, есть возможность также заменить Apache Maven и в системах автоматической сборки.
Угрозы: У1. Возможная скорая переработка целевого проекта – изменение / смена структуры; У2. Инструменты, выбранные для решения отдельных подзадач, могут не корректно работать в	1. Хоть возможность скорой переработки целевого проекта и ставит под вопрос целесообразность данной разработки, однако сроки и границы переработки пока что весьма расплывчаты и удалены, а текущая разработка все еще достаточно востребована в рабочем процессе. 2. Инструмент, который потенциально может очень сильно упростить версионирование проекта, далеко не	1. Появление новых ошибок во время разработки может замедлить работу команды из-за выхода систем автоматической сборки из строя и потратить множество сил на их поиск и устранение. Чтобы этого избежать, необходимо тщательно тестировать решение перед введением в эксплуатацию, в том числе

таком большом проекте; У3. Появление новых ошибок в результате переработки системы.	всегда работает корректно на крупных проектах, однако все же стоит внимательно его протестировать, т.к. он дает значительные преимущества.	и с системами автоматической сборки.

Приложение М - Перечень этапов, работ и распределение исполнителей

Основные этапы	№ раб	Содержание работ	Должность исполнителя
Предварительный аналитический этап.	1	Согласование разработки.	Лидер команды разработки проекта, начальник отдела.
	2	Формулирование требований к разрабатываемой системе.	Лидер команды разработки проекта, разработчик.
	3	Изучение текущего решения.	Разработчик.
	4	Подбор и изучение материалов по теме. Анализ существующих подходов.	Разработчик.
	5	Выработка наиболее оптимального подхода и его параллельное прототипирование.	Разработчик.
	6	Внесение корректировок в требования и выбранный подход.	Лидер команды разработки проекта, разработчик, программный архитектор.
	7	Ранжирование требований.	Разработчик.
Разработка	8	Разработка и параллельное тестирование.	Разработчик.
	9	Полное тестирование разработки.	Разработчик.
	10	Документирование внесенных изменений и создание инструкций по эксплуатации.	Разработчик.
	11	Проверка решения.	Другие члены команды.
	12	Исправление замечаний.	Разработчик.
	13	Внесение наработок в основной проект.	Разработчик.
Формирование сводной отчетности	14	Оценка и документирование эффективности изменений.	Разработчик.
	15	Публикация описания подхода во внутренней сети организации.	Разработчик.

Приложение Н - Расчёт трудоемкости выполнения работ (часть 1)

Название работы	Трудоемкость работ								
	t_{mini}, Ч/Д			t_{maxi}, Ч/Д			$t_{ожi}$, Ч/Д		
	Н. 1	Н. 2	Н. 3	Н. 1	Н. 2	Н. 3	Н. 1	Н. 2	Н. 3
Согласование разработки.	1	1	1	3	3	3	1,8	1,8	1,8
Формулирование требований к разрабатываемой системе.	1	1	1	5	5	5	2,6	2,6	2,6
Изучение текущего решения.	5	3	3	9	7	7	6,6	4,6	4,6
Подбор и изучение материалов по теме. Анализ существующих подходов.	4	2	3	7	5	6	5,2	3,2	4,2
Выработка наиболее оптимального подхода и его параллельное прототипирование.	4	4	5	10	8	10	6,4	5,6	7
Внесение корректировок в требования и выбранный подход.	1	1	1	3	3	4	1,8	1,8	2,2
Ранжирование требований.	1	1	1	2	2	2	1,4	1,4	1,4
Разработка и параллельное тестирование.	10	10	15	35	30	40	20	18	25
Полное тестирование разработки.	4	3	7	7	5	14	5,2	3,8	9,8
Документирование внесенных изменений и создание инструкций по эксплуатации.	1	1	1	3	3	3	1,8	1,8	1,8
Проверка решения.	3	3	7	6	6	10	4,2	4,2	8,2
Исправление замечаний.	2	1	3	4	3	5	2,8	1,8	3,8
Внесение наработок в основной проект.	1	1	1	2	2	2	1,4	1,4	1,4
Оценка и документирование эффективности изменений.	3	2	7	10	5	10	5,8	3,2	8,2
Публикация описания подхода во внутренней сети организации.	1	1	1	2	2	2	1,4	1,4	1,4

Приложение О - Расчёт трудоемкости выполнения работ (часть 2)

Название работы	Исполнители	T_{pi}, раб. дни			T_{ki}, кал. дни		
		Н. 1	Н. 2	Н. 3	Н. 1	Н. 2	Н. 3
Согласование разработки.	Начальник отдела, лидер команды разработки проекта	1	1	1	1	1	1
Формулирование требований к разрабатываемой системе.	Лидер команды разработки проекта, разработчик	1	1	1	2	2	2
Изучение текущего решения.	Разработчик	7	5	5	10	7	7
Подбор и изучение материалов по теме. Анализ существующих подходов.	Разработчик	5	3	4	8	5	6
Выработка наиболее оптимального подхода и его параллельное прототипирование.	Разработчик	6	6	7	9	8	10
Внесение корректировок в требования и выбранный подход.	Лидер команды разработки проекта, разработчик, программный архитектор.	1	1	1	1	1	1
Ранжирование требований.	Разработчик	1	1	1	2	2	2
Разработка и параллельное тестирование.	Разработчик	20	18	25	30	27	37
Полное тестирование разработки.	Разработчик	5	4	10	8	6	14
Документирование внесенных изменений и создание инструкций по эксплуатации.	Разработчик	2	2	2	3	3	3
Проверка решения.	Другие члены команды	1	1	2	2	2	3
Исправление замечаний.	Разработчик	3	2	4	4	3	6
Внесение наработок в основной проект.	Разработчик	1	1	1	2	2	2

Продолжение приложения О – Расчёт трудоемкости выполнения работ (часть 2)

Название работы	**Исполнители**	**T_{pi}, раб. дни**			**T_{ki}, кал. дни**		
		Н. 1	**Н. 2**	**Н. 3**	**Н. 1**	**Н. 2**	**Н. 3**
Оценка и документирование эффективности изменений.	Разработчик	6	3	8	9	5	12
Публикация описания подхода во внутренней сети организации.	Разработчик	1	1	1	2	2	2
Итого		**62**	**50**	**74**	**91**	**74**	**109**

Приложение П - Диаграмма Ганта для третьего приоритетного направления разработки

№ раб	Название работы	Исполнители	Tкi, кал. дни	Продолжительность выполнения работ														
				Январь		Февраль			Март			Апрель			Май			Июнь	
				2	3	1	2	3	1	2	3	1	2	3	1	2	3	1
1	Согласование разработки.	Начальник отдела, лидер команды разработки проекта	1															
2	Формулирование требований к разрабатываемой системе.	Лидер команды разработки проекта, разработчик	2															
3	Изучение текущего решения.	Разработчик	7															
4	Подбор и изучение материалов по теме. Анализ существующих подходов.	Разработчик	6															
5	Выработка наиболее оптимального подхода и его параллельное прототипирование.	Разработчик	10															
6	Внесение корректировок в требования и выбранный подход.	Лидер команды разработки проекта, разработчик, программный архитектор.	1															
7	Ранжирование требований.	Разработчик	2															
8	Разработка и параллельное тестирование.	Разработчик	37															
9	Полное тестирование разработки.	Разработчик	14															
10	Документирование внесенных изменений и создание инструкций по эксплуатации.	Разработчик	3															
11	Проверка решения.	Другие члены команды	3															
12	Исправление замечаний.	Разработчик	6															
13	Внесение наработок в основной проект.	Разработчик	2															
14	Оценка и документирование эффективности изменений.	Разработчик	12															
15	Публикация описания подхода во внутренней сети организации.	Разработчик	2															

Разработчик Лидер команды разработки проекта Начальник отдела Программный архитектор Другие члены команды

Приложение Р - Интегральный показатель ресурсоэффективности

Критерии оценки	Вес	Направление исследования		
		Напр. 1	Напр. 2	Напр. 3
Сокращение количества операций при сборке проекта	0,08	4	2	3
Сокращение количества операций при смене версии проекта	0,09	2	5	2
Ускорение смены версии проекта	0,09	2	5	2
Увеличение прозрачности управления зависимостями	0,08	2	2	4
Устранение дублирования зависимостей	0,09	2	2	4
Устранение необходимости выполнения строгой последовательности сборок для успешного завершения	0,06	4	2	2
Устойчивость к внесению изменений	0,06	3	4	4
Экономия внимания разработчиков	0,13	4	4	2
Экономия времени разработчиков	0,13	4	4	2
Возможность применения в других проектах	0,05	2	5	5
Вероятность отсутствия конфликтов и неисправностей при переработке решения	0,09	3	4	2
Время реализации	0,05	3	5	2
Итого	**1,00**	**3,00**	**3,66**	**2,69**