МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)
09.04.03 Прикладная информатика
(код и наименование направления подготовки)

Информационные системы и технологии корпоративного управления
(направленность (профиль))

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему «Моделирование системы управления кастомизацией модулей ИТ-проекта»

Студент А.А. Жданова _____
(И.О. Фамилия) (личная подпись)

Научный руководитель С.В. Мкртычев _____
(И.О. Фамилия) (личная подпись)

Руководитель программы д.т.н., профессор, С.В. Мкртычев _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»_____20_____г.

Допустить к защите
Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____» _____20_____г.

Тольятти 2019

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Наука и техника пользуются огромной популярностью у современной молодежи. Они — путь к карьерным возможностям в будущем. Занятия технической деятельностью стимулируют у молодых людей интерес в этих областях, помогают им овладевает необходимыми технологиями и побуждают к действию всех студентов, школьников и дошкольников.

В процессе выполнения проекта менеджерам надо оперировать значительным количеством данных, которые могут быть собраны и организованы с помощью компьютера. К тому же, многие аналитические возможности, пересчет графика деятельности с учетом практических данных, ресурсный и стоимостной анализ предполагают достаточно сложные для неавтоматизированного расчета алгоритмы.

Изменение систем управления проектами для ПЭВМ прошло через ряд этапов. С ростом мощности ПК улучшалась функциональность устройств, повышались их ресурсы. С установлением стандартов обмена данными между системами, расширением сетевых и интернет-технологий открылись новые удобные случаи для дальнейшего изменения систем поддержки процессов управления проектами и их более совершенного использования. Сами проекты делаются все более сложными, что представляет специальные требования к формированию информационных технологий управлению проектами.

Итак, **актуальность магистерской работы** обусловлена потребностью моделирования системы управления кастомизацией модулей ИТ-проекта.

Объектом исследования является процесс управления кастомизацией модулей ИТ-проекта.

Предмет исследования — система управления кастомизацией модулей ИТ-проекта.

Целью работы является разработка модели системы управления кастомизацией модулей ИТ-проекта, обеспечивающей повышение эффективности управления процессом кастомизации модулей ИТ-проекта.

Задачи исследования:

1. Проанализировать современные механизмы управления кастомизацией модулей ИТ-проекта.
2. Провести обзор и анализ современных типовых решений для управления кастомизацией модулей ИТ-проекта.
3. Выполнить сравнительный анализ и выбор методологических подходов к моделированию системы управления кастомизацией модулей ИТ-проекта.
4. Разработать концептуальную, логическую и физическую модели системы управления кастомизацией модулей ИТ-проекта.
5. Проверить адекватность реализованной модели системы управления кастомизацией модулей ИТ-проекта.

Гипотеза исследования: применение предлагаемой системы управления кастомизацией модулей ИТ-проекта позволит повысить эффективность процесса управления кастомизацией модулей ИТ-проекта.

Научная новизна исследования заключается в разработке новой модели системы управления кастомизацией модулей ИТ-проекта.

Методы исследования: современная концепция управления кастомизацией модулей ИТ-проекта, объектно-ориентированный подход, моделирование системы по методологии UML (Unified Modeling Language).

Публикации. Основные публикации по теме магистерской диссертации отражены в 2 статьях, представленных на научно-практических конференциях и индексируемых РИНЦ [,].

Основные этапы исследования: исследование проводилось в период 2017 – 2019 гг.

На ***первом*** этапе исследования (2017 г.) – была определена актуальность исследования, выполнена проработка литературы по выбранной теме, формулировалась гипотеза, ставились цели, задачи, определялись предмет, объект исследования, уточнены методы исследования, изучались

существующие на рынке системы управления кастомизацией модулей ИТ-проекта и выявлялись «слабые» места этих систем.

На ***втором*** этапе исследования (2017 – 2018 гг.) – выполнялось уточнение поставленных целей и задач, проводилось исследование методологий моделирования системы управления кастомизацией модулей и изучение процессов, протекающих при разработке стандартного ИТ-проекта.

На ***третьем*** этапе (2018 – 2019 гг.) – проводились разработка и анализ модели системы управления кастомизацией модулей ИТ-проекта, сделаны выводы по итогам выполненной работы.

Практическая значимость заключается в увеличении эффективности работы подсистемы управления модулей ИТ-проекта.

Степень разработанности темы исследования. Важную роль в решении проблем обеспечения эффективного управления конструированием робототехнических моделей сыграли российские и зарубежные специалисты Гребенюк Е.И., Гребенюк Н.А., Келим Ю.М., Лавровская Ю.Б., Мюллер С.

На защиту выносится модель системы управления кастомизацией модулей ИТ-проекта.

Результатом работы является совокупность теоретической и практической деятельности по направлению «Прикладная информатика», выполненная в процессе обучения.

В структуру работы входят: введение, 3 главы, заключение, список используемой литературы и приложение.

Работа изложена на … страницах и включает … рисунков, … таблиц, … источников.

Глава 1 АНАЛИЗ АКТУАЛЬНЫХ СИСТЕМ УПРАВЛЕНИЯ КАСТОМИЗАЦИЕЙ МОДУЛЕЙ ИТ-ПРОЕКТА

1.1 Анализ проблем эффективности управления кастомизацией модулей ИТ-проекта

Сегодня в условиях лютой конкуренции и общего экономического кризиса предприятия всеми приемлемыми им средствами стараются поднять свой экономический коэффициент полезного действия путем снижения расходов и повышения эффективности своих сбережений. Прежде всего это касается непрофильных родов занятий, таких как ИТ. Но также ясно, что минимизация издержек на ИТ не может быть универсальным решением трудностей компании, потому что, без сомнения, как раз ИТ является одним среди генераторов конкурентных успехов компаний в настоящее время и без которого трудно вообразить хоть сколько-нибудь большой бизнес. Вот почему становится ясно, что необходимо действительно оптимизировать, но не минимизировать затраты фирмы на ИТ в надежде пройти через тяжелый период, не утратив своих положений на рынке. Следует заметить, во многочисленных западных холдингах управление как раз эффективностью ИТ-проектов давно стало одним в числе главных и первых инструментов антикризисного руководства. Таким образом, необходимо прежде всего оценивать воздействие ИТ-проектов на бизнес определенной компании.

Сейчас существует множество всяких методов мониторинга и анализа эффективности накопительной привлекательности ИТ-проектов. В общей сложности можно отобрать три главные группы: финансовые, вероятностные и качественные методы оценки. Все они направлены на оценку эффективности индивидуальных ИТ-проектов по всевозможным критериям, чтобы руководство предприятия могло принимать рассчитанные и обдуманные выводы об их исполнении и финансировании.

Под структурой ИТ-проекта понимаются решения, которые принимаются относительно того, как ИТ-проект наилучшим образом соответствует своей цели. Со стороны практического подхода создания структуры ИТ-проекта должно подразумевать под собой написание чистого кода. Важными требованиями к такому коду являются максимально понятные зависимости и логика, а также структурная организация файловой системы, в которой располагаются папки и файлы.

Какие функции нужны помещаться в какие плагины? Как данные двигаются по проекту? Какие роли можно собрать воедино и изолировать? Отзываясь на подобные пункты, можно в разнообразном смысле прикидывать, как будет выглядеть оконченный продукт.

Модули в информационной системе позволяют классифицировать и отслеживать различные аспекты вашего бизнеса, такие как продажи, маркетинг, клиенты, продукты, события и тому подобное. Они бывают двух видов: стандартные модули и изготовленные на заказ либо пользовательские модули.

Обычно информационная система позволяет работать с одним или несколькими стандартными модулями для продаж, маркетинга, поддержки клиентов и управления запасами. Эти предопределенные модули поставляются с набором полей по умолчанию и макетами. Можно редактировать большинство аспектов стандартного модуля в соответствии с требованиями конкретного заказчика. Например, если в модуле Модуль1 есть поле Поле1, и вы думаете, что ни один из ваших клиентов не использует Поле1, вы можете удалить это поле из макета Модуль1. Точно так же есть много других опций настройки, предоставляемых стандартными модулями. Параметры настройки и их исключения различаются в зависимости от модуля, который необходимо изменить.

Стандартные, предварительно определенные модули доступны всем пользователям системы, независимо от версии, на которую они подписаны.

Иногда стандартные, предопределенные модули, присутствующие в системе, не всегда соответствуют всем требованиям заказчика. В таком случае система позволяет создавать новый модуль в зависимости от потребностей вашего бизнеса. Например, модули «Контакты», «Руководство» и «Сделки» не будут идеальными для ИС образовательного учреждения. «Ученики», «Учителя» и «Родители» были бы более подходящими модулями для них, и они могут создавать эти модули. Больница может создавать новые модули под названием «Врачи», «Пациенты» и «Медсестры».

Чтобы удовлетворить эти уникальные бизнес-требования, можно создавать свои собственные модули. Благодаря функциональности пользовательских модулей в системе можно разрабатывать новые модули, используя встроенные инструменты, которые не требуют навыков программирования. Эти пользовательские модули могут легко интегрироваться с основными модулями стандартной системы и не должны быть автономными модулями.

В настоящее время большинство крупных и не очень ИТ-проектов зачастую обнаруживают, что решения, разрабатываемые и имеющиеся на рынке, включающие стандартные функциональные возможности, просто не подходят под требования самого проекта. Одним из возможных решений в данном случае можно считать переделку или доработку одного из предложенных решений под требования текущего ИТ-проекта. И каждый проект должен определить для себя архитектуру для будущего продукта, которая предполагает под собой возможное расширение функционала, не забывая при этом думать о возможных ловушках и неудачах, которые подстерегают почти на каждом шагу каждого разработчика в процессе создания такого решения.

В данной работе изучены некоторые из вероятных подходов к кастомизации ИТ-проекта.

1. Воспользоваться подходом с использованием атрибутов, которые имеют возможность динамически изменяться. К такому подходу можно отнести использование модели Entity-Attribyte-Value.

Такую модель, в свой черед, именуют Open Schema. Она думается быть использованной совместно с типовой реляционной моделью для трансформирующегося во времени установления и хранения новых величин новых признаков (атрибутов) сущностей. При применении сей модели все значения атрибутов главным образом вносятся в одну таблицу из трех столбцов. Столбцы по большей части называют как Entity, Attribute, Value:

- Entity — хранит ссылку на сущность, характеристики которой отображаются. Во многих случаях составляет определителем сущности;
- Attribute — сохраняет ссылку на обозначение атрибута;
- Value — нынешнее значение атрибута.

На рисунке 1.1 изображена схема данной модели «Сущность-Атрибут-Значение».

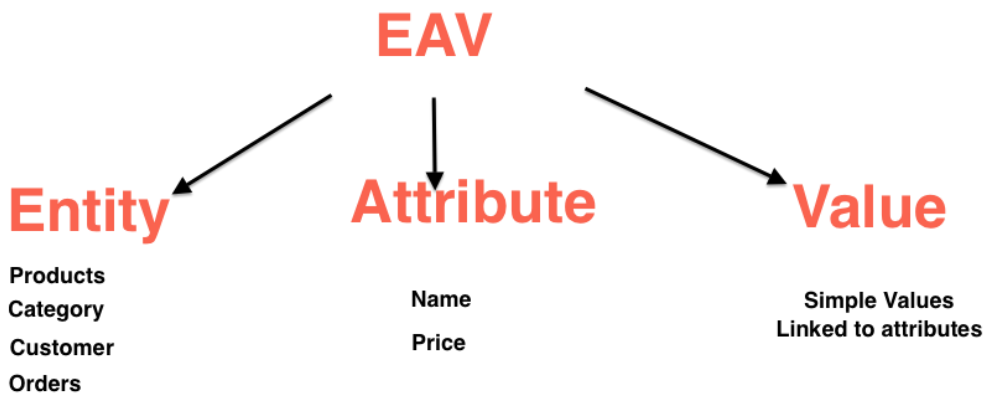

Рисунок 1.1 – Модель Entity-Attribute-Value

Определенно, таблица, которая будет содержать описание служебных данных, описывающих атрибуты, должна являться обязательным

составляющим схемы. Примеры таких служебных данных, используемых при описании атрибутов, могут быть следующими:

- описание типа атрибута;
- ограничения по содержанию информации, которое включает в себя атрибут;
- ссылка на компонент, который используется в UI для отображения данного атрибута;
- порядковый номер в очереди на отрисовку компонента, содержащего данный атрибут.

Для использования данной модели в проекте нужно выполнить две вещи:

1. Исполнить механизм установления метаданных. С его содействием удастся, например, отметить, что к сущностям типа "Постоянный покупатель" добавилось очередное свойство "Контактный номер", тип этого поля строковый, для его показа используется компонент текстового поля.
2. Создать приспособления индикации и введения содержаний меняющихся атрибутов на требующихся видеопанелях программного обеспечения. Конструкция должна находить эвентуальный набор атрибутов для имеющейся сущности в таблице с обзором метаданных, рекомендовать компоненты для их улучшения, а далее из таблицы с данными доставать и отражать их значения, отображая их при сворачивании экрана.

Одним из действительно значимых плюсов в использовании такого подхода можно считать отсутствие надобности в проектировании и имплементации проекта-расширения. Это значит, что заказчик получает готовый продукт, в рамках которого просто создаются различные динамические свойства для сущностей в процессе настройки или даже в процессе эксплуатации.

Правда, в таком подходе есть и свои недостатки. В первую очередь необходимо сказать об ограниченности использования такого подхода. При выборе модели динамических атрибутов должно понимать, что она позволит

только создать дополнительные свойства для сущности и отрисовать эти свойства на экране в заранее отведенном для них месте. При таком подходе об изменении свойств или даже UI компонентов не может быть и речи.

Далее необходимо отметить, что подход с использованием динамически настраиваемых атрибутов существенно увеличивает нагрузку на всю базу данных. Даже если не учитывать связи, которыми обладает каждая сущность, процесс выгрузки отдельного экземпляра любой сущности потребует больше времени и ресурсов, потому что придется считать не одну, а несколько строк из таблицы. А если встает необходимость в выгрузке целой коллекции сущностей, примером которой можно считать компонент выпадающего списка для UI, то придется сделать N+1 таких запросов, либо усложнять запросы путем добавления джойнов, количество которых будет равняться числу колонок в таблице. Все это вкупе с тем, что в большинстве ИТ-проектов БД является крайне плохо масштабируемым и одним из самых медленных компонентов звеном, очень легко и быстро может привести к тому, что абсолютно вся система перестанет функционировать.

В-третьих, ведь из-за структуры базы будет вполне мудрено подтягивать выборки данных для ведомостей — вместо того, чтобы сочинять следующий по очереди SQL-запрос для реляционных данных, станут необходимыми куда больше комплексные запросы.

2. Воспользоваться подходом, подразумевающим под собой использование модулей.

Использование подобной архитектуры даст возможность спроектировать и имплементировать дополнительные функциональные решения, методы и классы, которые можно будет держать в изолированных друг от друга и от «ядра» программы модулях. Если у заказчика возникает необходимость в одном из таких модулей, вместо того, чтобы переписывать проект, будет достаточно лишь написать новый модуль и прописать связи для того, чтобы интегрировать его в свой готовый проект. Правда, нужно помнить о

необходимости спроектировать точки расширения, чтобы интеграция модулей была возможна. Что же понимается под точкой расширения? Обычно это специальные классы, которые занимаются тем, что выбирают из уже доступных модулей те, которые имеют необходимую в данный момент времени исполнения программы функциональность и в случае обнаружения такого модуля передают управление процессом ему. Самыми простыми примерами могут послужить различные кнопки или пункты меню в UI.

На рисунке 1.2 показана схема модульной информационной системы.

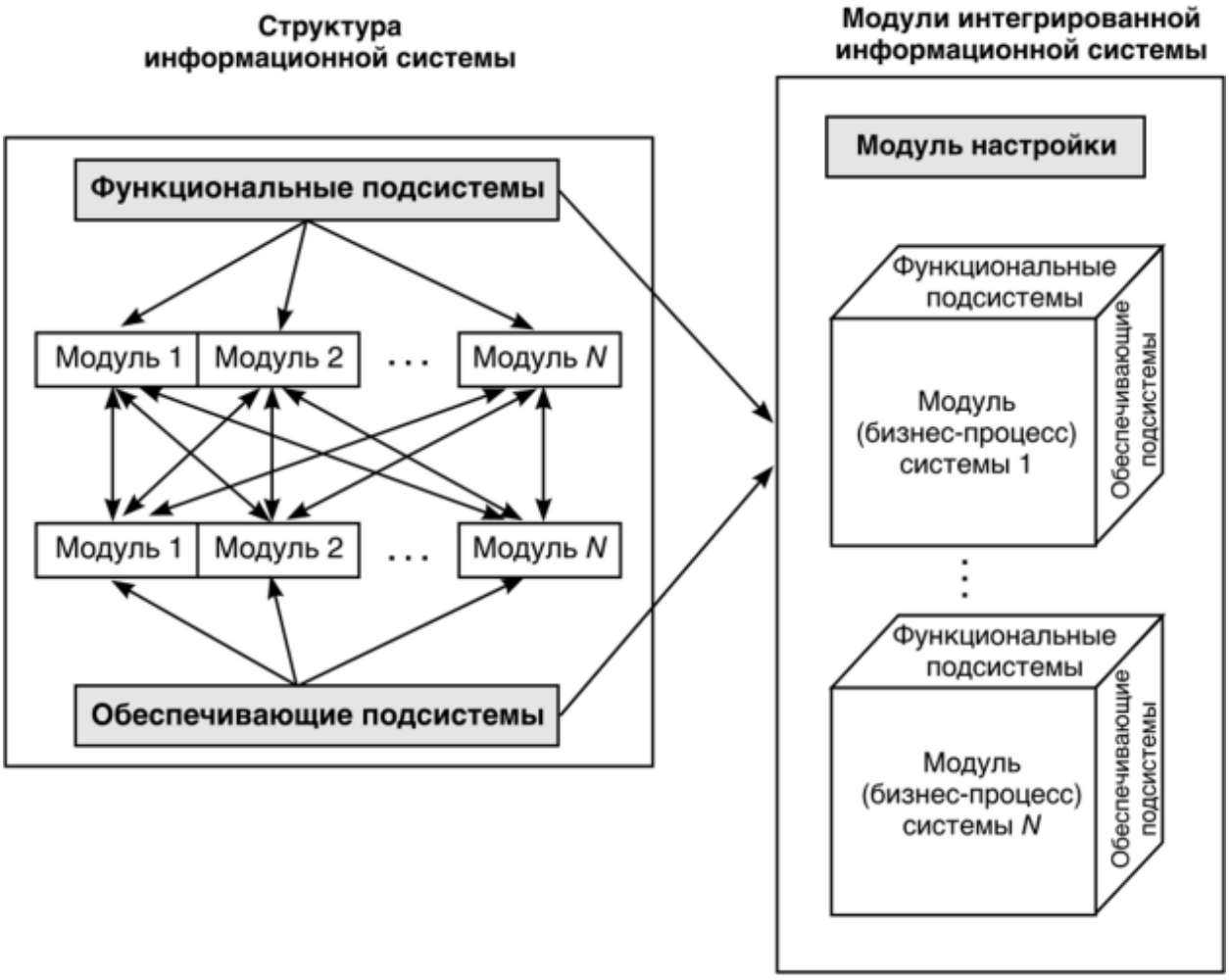

Рисунок 1.2 – Модульная информационная система

Сценарий логики и обработчиков действий во внешних скриптах — сплошь и рядом используемая разновидность углубления функциональности, которую равным образом имеется возможность прибавить к вариациям

плагинов оттого, что скрипты вызываются и выполняются конкретными точками программного обеспечения.

Модульная архитектура потворствует хранить совсем весь новый функционал по раздельности, среди прочего, по отдельности и от продукта, что без вопросов является очень хорошим достоинством такого типа архитектуры. Абсолютная физическая дифференциация продукта и плагинов значительно упрощает процедуру обновления версий программы или модуля, во всяком случае, достаточно только заместить существующий компонент свежим.

Но и в этом подходе есть свои подводные камни. Иногда, правда, они не сильно повлияют на работоспособность продукта, но может произойти и так, что эти камни могут привести к невозможности использования модульной архитектуры. В действительности использование модулей может быть оправдано только при условии, что в системе были спроектированы заранее точки расширения. Конечно, существуют такие проектные решения, где подобные точки практически отсутствуют и известны заранее, но в то же время имеют место и другие проекты, в которых приходится только гадать в каком месте потребуется улучшить функционал. Анализирование же подобных ситуаций с обнаружением таких мест расширения на будущее является процессом непростым и отнимет много ресурсов, к тому же, оно не дает никаких гарантий успеха. Да и сами точки расширения — это места кода, которые приводят к его усложнению, что, в свою очередь, приведет к увеличению возможности возникновения ошибок и сложности сопровождения. Так что к проектированию и имплементации таких точек стоит отнестись очень серьезно. [12].

Модульная архитектура позволяет предприятиям предлагать разнообразные продукты, чтобы быстро реагировать на меняющиеся требования рынка, улучшая разнообразие продуктов и уменьшая сложность инженерной системы. Модульная технология, как наиболее эффективная и

широко используемая технология, стала самой важной особенностью проектирования структуры продукта.

При массовой настройке модули продукта организованы и управляются в соответствии с определенными правилами, а различные типы модулей собираются для формирования модульной организационной структуры. Модульная структура сложных продуктов имеет много характеристик, таких как различные типы модулей, децентрализованные источники модулей и несбалансированное использование модулей. Это сложная система, которая влияет и взаимодействует со многими видами факторов. Различные факторы приводят к нестабильности модульной организационной структуры, такие как случайные факторы, включая частоту отказов при поступлении внешнего модуля и низкую частоту выполнения самодельных модулей, а также объективные факторы, включая большой спрос на модуль и высокие показатели использования модулей. Эти факторы приведут к отсутствию своевременной поставки модулей, задержке поставки продукции и т.д., и в конечном итоге скажутся на стабильности производства. Поэтому, учитывая различные факторы, стабильность модульной организационной структуры улучшается путем изменения типа модуля, улучшения структуры модуля, разработки новых модулей и т.д., следовательно, изучение эволюции модулей имеет важное значение для инженерных приложений.

По мере увеличения сложности и типа механических изделий, тип и количество модулей продуктов быстро увеличиваются. Это важный вопрос в руководстве эволюции модуля. Организационная структура модуля на предприятии – это не простая статистика количества различных модулей. Чтобы оптимизировать организационную структуру модулей, необходимо рассмотреть ряд проблем, таких как базовые отношения между модулями, количество модулей и изменения в модулях.

Два важных соображения при реализации эволюции и оптимизации организационной структуры модуля следующие:

1. Как установить модель организационной структуры модуля.

Во-первых, модель модульной структуры должна быть установлена при исследовании эволюции модуля. В настоящее время матрица структурной структуры, теория графов и сложная сеть являются основными методами моделирования организационной структуры модуля. Методы исследования описывают отношения между компонентами и модулями, устанавливая модель ненаправленной, направленной или взвешенной сети. Однако большинство объектов моделирования зависят от продукта и не учитывают организационную структуру модуля различных продуктов. Было предложено использовать те же части в семействе продуктов в качестве точки совпадения и создать модели многопродуктовых сетей посредством суперпозиции дерева продуктов для размещения продуктов и компонентов в одной сети для исследований, но существуют многоуровневые отношения между продуктами, модулями, компонентами, что не способствует эволюции исследовательских модулей.

2. Как обеспечить стабильность организационной структуры модуля при исследовании эволюции модуля.

Эволюция модулей способствует изменениям в организационной структуре модулей, а направление эволюции модулей напрямую влияет на оптимизацию организационной структуры модулей. Как сложная система, стабильность организации модуля имеет решающее значение. В области исследования стабильности системы была предложена концепция уязвимости энергосистемы и создан слабый метод анализа, основанный на переходной энергетической функции и искусственной нейронной сети. В 2000 году Альберт Фуад исследовал хрупкие источники, основанные на сложной теории, которая привела к хрупкости системы в новую эпоху. Ван предложил модель решетки каскадных отказов, связанная с картой, которая обеспечивает хорошие математические средства для моделей каскадных отказов сложных сетей. Джин и его команда предложили хрупкость системы с помощью теории энтропии и теории мутаций системы. Кроме того, в последнее время появились новые

достижения в теории хрупкости сложных сетей и приложениях хрупкости сложных сетей. Однако, по сравнению с другими сложными системами, механизм распространения хрупкого поведения, хрупкие источники и другие факторы отличаются из-за уникальных характеристик организационной структуры модуля, и соответствующий механизм требует глубокого анализа. Кроме того, динамика эволюции модуля определяет характеристики динамических изменений в системе. Стабильность системы, как важный показатель оптимизации системы, должна следить за динамикой обновления [20].

1.2 Современная концепция управления кастомизацией модулей ИТ-проекта

Начав написание небольшого, да реального и перспективного проекта, легко на собственном опыте вынести убеждение, насколько значительно то, чтобы программа как хорошо работала, так и была как следует организована. Не стоит уповать, что обоснованная архитектура нужна только огромным проектам (очевидно, что для больших проектов отсутствие архитектуры может обернуться «летальностью»). Сложность большею частью растет гораздо резче размеров программы. Также если не поберечься об этом авансом, то довольно живо наступает минута, когда она прекращает быть контролируемой. Хорошая архитектура экономит на удивление много сил, дней и денег, что нередко вообще указывает на то, выживет данный проект или ни за что. И даже в том случае, когда речь идет не более того, чем о построении упрощенного проекта, все равно на первых порах очень желательно ее спроектировать.

Имеется возможность сформулировать список во всех отношениях разумных и всеобъемлющих критериев высококачественной архитектуры:

- **Эффективность системы**. Сначала программа, безусловно, должна решать назначенные задачи и как следует выполнять свое назначение, причем во всякого рода условиях. Сюда не возбраняется отнести такие параметры, как

надежность, безопасность, производительность, способность преодолевать увеличение нагрузки (масштабируемость) и т.п.

- **Гибкость системы**. Какое угодно приложение приходится пересматривать со временем — трансформируются требования, добавляются современные. Чем живее и удобнее можно вставить изменения в содержащийся функционал, чем меньше головных болей и заблуждений это произведет, тем гибче и способнее составить конкуренцию система. По этой причине в рамках разработки следует стараться взвешивать то, что получается, на тему, как это в будущем, возможно, придется модифицировать. Изменение одной части системы не должно оказывать действие на ее прочие части.

- **Расширяемость системы**. Допустимость приписывать в систему очередные сущности и функции, не преступая ее основной композиции. На начальной стадии в систему есть расчет закладывать лишь самый важный и вынужденный функционал, но наряду с этим архитектура должна разрешать легко наращивать вспомогательный функционал сообразно необходимости. Причем затем, чтобы внесение в наибольшей степени вероятных изменении стоило наименьших усилий.

- **Масштабируемость процесса разработки**. Возможность снизить срок разработки из-за приобщения к проекту новых лиц. Архитектура должна разрешать распараллелить процесс разработки, чтобы целый ряд людей могли действовать над программой все вместе.

- **Тестируемость**. Код, который проще тестировать, будет сохранять меньше промахов и надежнее исполняться. Но тесты не шлифуют лишь качество кода. Многие разработчики прибывают к заключению, что требование «высокой тестируемости» является также гонящей силой, автоматически сводящей к на совесть спроектированному дизайну, и сразу одним из важнейших мер, позволяющих рассчитать его качество.

- **Возможность повторного использования**. Систему нужно конструировать так, чтобы ее части можно было вторично использовать в разных системах.

- **Хорошо структурированный, читаемый и понятный код. Сопровождаемость**. Над программным обеспечением, как правило, трудится множество людей — некоторые уходят, приходят очередные. После написания дополнять программу тоже, большей частью, необходимо людям, не фигурирующим в ее производстве. Поэтому правильная архитектура должна давать шанс относительно просто и быстро постичь систему новой публике. Программа должна быть хорошо структурализована, не содержать дублирования, насчитывать хорошо составленный код и не мешало бы документацию. А также крайне желательно использовать стандартные, общеизвестные и часто используемые решения, которыми пользуются программисты по всему миру. Чем вычурнее система, там более сложным становится ее понять для людей, которые будут эту систему эксплуатировать.

Критерии плохой архитектуры системы:

1. **Жесткость.** Ее тяжело изменить, ибо любое изменение имеет влияние на слишком большую численность других частей информационной системы.

2. **Хрупкость.** При внесении преобразований неожиданно ломаются прочие элементы системы.

3. **Неподвижность.** Код тяжело использовать во второй раз в другом проекте, поскольку его не в меру тяжело «освободить» из текущего приложения.

Несмотря на отсутствие однообразия критериев, все-таки основной при разработке больших систем видится проблема снижения сложности. А для ослабления сложности ничего вне деления на элементы пока не изобретено. Иногда это именуют принципом «разделяй и властвуй», но по существу говорится об иерархической декомпозиции. Комплексная система должна

основываться на небольшом числе более элементарных подсистем, каждая из коих равным образом строится из компонент меньшего размера, и т.п., до того времени, пока самые мелкие фрагменты не будут достаточно легки для непосредственного осознания и создания.

На рисунке 1.3 показан процесс создания архитектуры информационной системы.

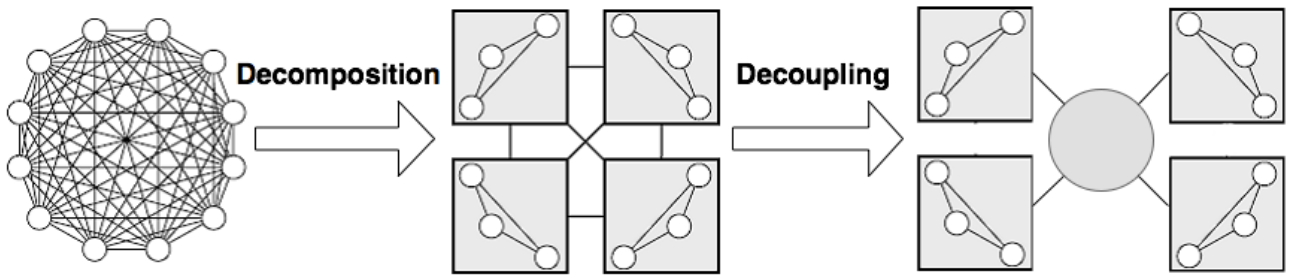

Рисунок 1.3 – Создание архитектуры системы

Предоставленное выше решение является как единственно известным, так и многофункциональным. Помимо снижения запутанности, оно одновременно гарантирует гибкость системы, обеспечивает хорошие шансы для масштабирования, ко всему прочему, позволяет повышать надежность благодаря дублированию критически важных звеньев.

Соответственно, когда повествуется об устройстве архитектуры программы, основании ее структуры, под тем, главным образом, предполагается декомпозиция программного обеспечения на подсистемы (функциональные модули, сервисы, слои, подпрограммы) и подготовка их взаимодействия между собой и внешней средой. При этом, чем более самостоятельны подсистемы, тем безопаснее сконцентрироваться на разработке каждой из них по отдельности в конкретный временной промежуток и наряду с этим не следить за всеми остальными частями.

При данных обстоятельствах программа из плохо спроектированной, слабо структурированной и запутанной трансформируется в конструктор, состоящий из последовательности модулей/подпрограмм, коммуницирующих между собой по хорошо известным и простым принципам, что, действительно,

и позволяет проверять ее сложность, кроме этого, дает возможность брать все те достижения, которые обычно сопоставляются с понятием архитектуры высокого качества:

- **масштабируемость** — способность системы расширяться и преумножать собственную производительность благодаря внедрению новых модулей;
- **ремонтопригодность** — при модификации одного модуля, остальным модулям нет необходимости изменяться;
- **заменимость модулей —** один модуль может быть без труда заменен другим модулем;
- **возможность тестирования** — каждый модуль можно протестировать/изменить/починить независимо от других модулей;
- **переиспользование** — модуль можно использовать в разных проектах и на разном окружении без каких-либо модификаций;
- **сопровождаемость** — программу, разбитую на отдельные модули, гораздо легче сопровождать по мере эксплуатации.

Одним словом, целью большинства различных методов проектирования является разбиение одной сложной задачи на более простые составляющие.

Исходя из всего выше сказанного можно сделать вывод, что примером хорошей и качественной архитектуры является модульная/блочная архитектура. Для того, чтобы спроектировать и получить на выходе качественное архитектурное решение необходимо знать о том, как правильно реализовывать декомпозицию системы. А это в свою очередь означает, что очень нужно понимать – какую декомпозицию считать «правильной» и какой способ лучше всего подходит для ее проведения.

1.3 Основа для создания системы управления кастомизацией модулей ИТ-проекта

Кастомизация (от английского *to customize* – регулировать, изменять что-то, производя отвечающее нуждам конкретного клиента) — это индивидуализирование продукции под заказы определенных потребителей. Это добивается путем внесения применимых или дизайнерских преобразований.

Основная миссия кастомизации — создать у покупателя ощущение, что продукт делается именно для него и утоляет его личные запросы. Кастомизация считается высшей целью взаимодействия в сценарии «поставщик материальных благ или услуг — покупатель». Она обеспечивает высококонкурентное преимущество благодаря образованию более высокой ценности (преимущества) для заказчика.

Процесс кастомизации внедряется специально на принципах индивидуальных распоряжений и запросов относительно продукта. Кастомизация проекта призывает пересмотреть параметры по желанию пользователя, взять, цветовую гамму графического интерфейса пользователя (среди прочего, и некоторых его частей). Это может выступать, например, нанесение логотипа заказчика на шапку с меню.

Система кастомизации — это замечательная психологическая кампания: пока клиент комплектует конфигурацию продукта под свои нужды, он начинает воспринимать себя его собственником.

Современные контуры в разработке ИС настаивают вкладывать в архитектуру систем проблематичность динамического расширения их функционала. И, несмотря на существование видного количества сведений в этом течении, всеобщего урегулирования строения плагинного приложения не существует. Применение же сделанного кем-то способа разрешения не всякий раз возможно ввиду специфики языка программирования либо эксплуатируемой системы. По аналогии, чужие решения плагинных систем не всегда доступны для усвоения, а иногда не в меру хитры.

Плагины во всякого рода системах зачастую обладают различными границами функциональности. В ИС могут быть выдвинуты некоторые жестко определенные точки углубления — часть функционала, дополняемая сторонними разработчиками. Иначе в полном объеме система может реализовывать собой лишь механизм управления модулями, а функционал весь вынесен в особенные плагины.

На сегодняшний день одним из известных и используемых подходов к организации расширяемого приложения с возможностью кастомизации его функционала является модель «сущность-атрибут-значение» (EAV).

Модель «сущность-атрибут-значение» (EAV) — это модель данных для кодирования, с эффективным использованием пространства, сущностей, в которых количество атрибутов (свойств, параметров), которые могут использоваться для их описания, потенциально огромно, но число, которое будет на самом деле относится к данной сущности относительно скромно. Такие объекты соответствуют математическому понятию разреженной матрицы.

EAV также известен как модель «объект-атрибут-значение», «модель вертикальной базы данных» и «открытая схема».

Если использовать этот подход для представления данных, то можно заметить существенные сходства с хранением только ненулевых значений в разреженной матрице. При использовании модели данных, использующую динамические изменяемые атрибуты, можно заметить, что фактом, который описывает сущность, будет являться пара атрибут-значение. При этом одна строка в таблице базы данных, спроектированной по такой модели, хранит один факт. Таблицы EAV часто описываются как «длинные и тонкие»: «длинный» относится к числу строк, «тонкий» к нескольким столбцам.

Данные записываются в виде трех столбцов:

- **Сущность**: описываемый объект.

- **Атрибут** или **параметр**: в большинстве случаев его реализация являет собой внешний ключ, имеющий отношение к таблице, в которой этот атрибут определен. Эта таблица обычно содержит такие столбцы, как имя параметра, его описание и идентифицирующий номер, набор каких-либо ограничений по содержанию для информации, хранящейся в этом атрибуте и т.д.
- **Значение** атрибута.

Далее рассмотрен пример, как можно попытаться представить медицинскую карту пациента в реляционной БД. Ясно, что образование таблицы (или набора таблиц) с сотнями столбцов тяжело, поскольку значительное большинство столбцов будет нулевым, так как разные пациенты могут иметь совершенно разный набор заболеваний. Чтобы усложнить ситуацию, в многолетней медицинской карте, которая закреплена за пациентом на протяжении времени, может быть ряд значений одной и той же характеристики: рост и вес, например, меняются по мере антропоморфных изменений в теле ребенка. Наконец, вселенная клинических данных продолжает расти: например, открываются болезни и воплощаются новые лабораторные испытания; это вызовет не перестающее добавление столбцов и постоянный пересмотр пользовательского интерфейса. Ситуация, когда список атрибутов часто изменяется, на языке базы данных называется «изменчивость атрибутов».

Ниже приведено статическое представление таблицы EAV для клинических данных, полученных во время визита к врачу с лихорадкой утром 6 февраля 1999 года. Записи, показанные в угловых скобках, являются ссылками на записи в других таблицах, показанные здесь как текст, а не как закодированные значения внешнего ключа для простоты понимания. В примере, приведенном выше, значения, которые были там использованы, были буквенными. Однако стоит понимать, что эти значения могут иметь и другие форматы, которые можно заранее сохранить в списке значений. Такие списки

особенно полезно создавать в случае, если становится известно о том, что некоторые значения ограничены (то есть, их можно пересчитать).

- **Сущность**. Для клинических результатов субъектом является пациент: внешний ключ в таблице, который содержит как минимум идентификатор пациента и одну или несколько меток времени (например, начало и конец даты/времени обследования), которые записываются, когда описываемое событие произошло.
- **Атрибут или параметр**. Внешний ключ в таблице определений атрибутов (в этом примере - определения клинических результатов). Как минимум, таблица определений атрибутов будет хранить соответствующие столбцы: идентификатор атрибута, его имя, описание, тип данных, единицы измерения и столбцы, содействующие проверять ввод, например, максимальная длина строки и регулярное выражение, максимально и минимально допустимое значения, набор допустимых значений и т. д.
- **Значение атрибута**. Оно зависит от типа данных.

Приведенный ниже пример иллюстрирует симптомы, которые могут наблюдаться у пациента с пневмонией.

(<patient XYZ, 2/6/99 9:30 AM>, <Temperature in degrees Celsius>, "39")

(<patient XYZ, 2/6/99 9:30 AM>, <Presence of Cough>, "True")

(<patient XYZ, 2/6/99 9:30 AM>, <Type of Cough>, "With phlegm, yellowish, streaks of blood")

(<patient XYZ, 2/6/99 9:30 AM>, <Heart Rate in beats per minute>, "98")

Описанные выше данные EAV-модели сопоставимы с содержимым чека из супермаркета (которые будут отражены в таблице Sales Line Items в базе данных). В чеке указана только информация о фактически купленных товарах, вместо того, чтобы перечислять все товары в магазине, которые покупатель мог купить, но не сделал. Как и клинические данные для данного пациента, товарный чек скуден.

«Сущность» — это идентификатор продажи/транзакции, то есть внешний ключ в таблице транзакций продажи. Она используется для внутренней маркировки каждой позиции, хотя в чеке информация о продаже появляется сверху (местоположение магазина, дата/время продажи) и снизу (общая стоимость продажи).

«Атрибут» — это внешний ключ в таблице продуктов, откуда можно посмотреть экспликацию, стоимость за экземпляр, бонусы и рекламные скидки и т.д. (Съестное так же непостоянно, как и клинические данные, возможно, даже более того: очередная провизия включается постоянно, в то время как другие снимаются с рынка, если спрос на них низкий. Ни один сведущий создатель БД не будет безоговорочно зашифровывать некоторые продукты питания, такие, как молоко или пельмени, как столбцы в таблице.)

«Значения» — это количество купленных товаров и общая цена позиции.

Моделирование строк, где факты о чем-либо (в данном случае транзакция продажи) записываются в виде нескольких строк, а не нескольких столбцов, является стандартной техникой моделирования данных. Различия между моделированием строк и EAV-моделированием (которое можно считать обобщением моделирования строк) приведены ниже:

- Строковая таблица однородна по фактам, которые она описывает: таблица «Позиции» описывает только проданные продукты. Напротив, таблица EAV-модели содержит практически любой тип факта.
- Тип данных столбца(-ов) значений в таблице, моделируемой по строкам, заранее определяется характером записываемых им фактов. В то время, как в таблице EAV-модели тип данных значения в конкретной строке находится во власти атрибута в этой строке. Именно поэтому в производственных системах прямой ввод данных в таблицу EAV-модели может стать причиной катастрофы, поскольку сам механизм базы данных не сможет выполнить надежную проверку входных данных.

В хранилище клинических данных моделирование строк также находит многочисленные применения. Подсхема лабораторных испытаний обычно моделируется именно таким образом, потому что результаты лабораторных испытаний обычно являются числовыми или могут быть закодированы численно.

1.4 Программно-техническое обеспечение в моделировании системы управления кастомизацией модулей ИТ-проекта

Существенную функцию управления составляет информационно-аналитическая функция. Одним из возможных методов, которым можно воспользоваться для проектирования и имплементации информационно-аналитического сопровождения, является метод, в котором идет проектирование и реализация информационной системы управления.

В данном примере, под информационной системой следует понимать такую систему, которая будет построена на базе компьютерных технологий и осуществлять хранение, обработку, поиск и передачу крайне больших по объему кусков информации, а также будет обладать практической областью эксплуатации. В различных компаниях такая система, в большинстве своем, реализована с применением технологии локальной сети. Благодаря такому подходу, текущая по каналам связи информация может быть выдана по запросу для разных пользователей, при этом эта информация не обязательно будет храниться на одном компьютере, а может быть распределена среди разных устройств, входящих в эту локальную сеть.

Абсолютно любая система, представляющая собой информационную систему, включает в себя различные компоненты, среди которых можно выделить следующие:

- база данных, находящаяся, в основном, не во внутренней памяти устройства, а во внешней, и хранящая в себе абсолютно всю информацию, необходимую для работы;

- программы и подпрограммы, выполняющие различные действия с данными путем обращения к базе данных, хранящих эти данные, при помощи запросов;
- графический интерфейс, с которым работает пользователь, созданный и спроектированный таким образом, чтобы доступно объяснить правильный порядок для общения этого пользователя с информационной системой.

Ниже будет описан хронологический порядок различных этапов, призванных для реализации информационной системы. Все начинается с организации полного системного анализа области, предмет которой и будет представлен в функционале конечной информационной системы. В конце, в качестве итога всех этапов, будет показана инфологическая модель.

Еще одной стадией оформляется выбор отличной СУБД, которая будет употреблена для создания базы данных и исполнения программных приложений.

Через какой-то промежуток временного интервала инициируется создание модели, данные в которой будут определены на основании инфологической модели, которая была развернута в первой стадии. Структура такой модели будет зависеть от того, какой порядок представления будет выбран в качестве порядка представления данных в той СУБД, которая будет использована в проекте. Модель, созданная по предложенному описанию, обычно называют датологической моделью данных.

Описанные этапы являются теоретическими (проектными). Через некоторое время начинаются действительные работы в среде СУБД — строится конституция базы данных и реализуется введение данных.

Следующий этап включает в себя написание приложения, которое будет работать с информационными запросами, поступающими от пользователя. Такое приложение должно быть написано на языке, носящем название DML, или язык манипулирования данными.

Получается, что для создания ИС нужно подвергнуть анализу как предмет изысканий, так и цель производства под углом системного подхода.

Если имеются ввиду две фазы проектирования, являющиеся основными – макропроектирование и микропроектирование, то за базу системного подхода можно взять некую обоснованность процесса, использованного при разработке моделей.

На стадии макропроектирования в соответствии с данными о материальной системе S и окружающей обстановке E основывается модель окружающей обстановки, обнаруживаются факторы выпуска и квоты для создания модели системы, выбирается модель системы и мерки, делающие возможным вычислить адекватность модели M материальной системы S. Составив модель системы и модель окружающей обстановки, на почве критерия продуктивности работы системы при исполнении моделирования присматривают самую лучшую стратегию управления, что дает возможность осуществить потенциал модели по показу некоторых качеств работы материальной системы S.

От типа модели, определенного на этапе проектирования этой модели, в большой степени зависит стадия микропроектирования. К примеру, если была избрана имитационная модель, то необходимо спроектировать возможность создания технического, информационного, программного и математического обеспечения системы, которая проходит процесс моделирования. На этом этапе имеется возможность утвердить основные свойства спроектированной модели, а также установить расчетное время работы с этой моделью и вычислить возможные расходы ресурсов, которые пойдут на получение требуемого качества соответствия этой модели процедуре, которая является, в свою очередь, процедурой функционирования системы, обозначенной выше как S.

Независимо от того, какой тип модели M был привлечен, при проектировании этой модели необходимо иметь ввиду некоторые особенности используемого системного подхода, а именно:

1) передвижение по пунктам и различным направлениям в процессе создания модели должно быть соразмерно-последовательным;

2) должно быть выполнено сбалансирование таких характеристик, как ресурсные, надежностные и информационные и другие характеристики;

3) в системе моделирования должно быть установлено правильное соотношение всех особенных уровней, входящих в общую иерархию;

4) все изолированные от других стадии проектирования и создания модели должны быть целостными.

Модель М должна подходить заданной миссии ее образования, поэтому отдельные составные части должны составляться обоюдно, не исключая объединенную системную функцию. Она может быть поставлена на хорошем уровне, в таком разе она будет иметь отличительной способностью большую наполненность сути и на протяжении многих лет может выставлять объективные альтернативы предоставленной системы моделирования. При численном формулировании назначения строится целевая функция, которая по существу фиксирует наиболее важные факторы, имеющие влияние на свершение цели.

Ряд системных проблем возникает при проектировании модели. Для решения таких проблем часто искусственно создаются способы их исправления, основанные на большом количестве исходной информации, которая зависит от разных предложений целых групп специалистов, включающих в себя большое количество людей. В такой ситуации принято применять именно системный подход, потому как он дает возможность не только смоделировать реальный объект, но и, используя этот объект в качестве базы, примерно наметить количество управляющей информации, которая будет необходима. Ко всему прочему, такой подход позволит дать оценку функционирования и использовать эту оценку при моделировании, чтобы найти самый эффективный способ, которым можно воспользоваться для построения, а

также для выбора подходящего режима функционирования практической системы S.

Направление системных исследований постоянно развивается, расширяя и экспериментальные методы изучения явлений. В связи с этим растет значение абстрактных методов, начинают появляться все новые научные направления и дисциплины, а элементы умственного труда подвергаются автоматизации. Огромным значением при создании материальных систем S обладают именно математические методы анализа, так как большое количество открытий имеют в качестве базы лишь теоретические исследования. С другой стороны, неправильно забывать, что, в любом случае, одним из основных критериев абсолютно любой теории является практика, и даже в случае абсолютно математические науки базируются на фундаменте знаний, полученных на практике.

На рисунке 1.4 показано деление бизнес-процессов моделирования любого проекта на мелкие транзакции.

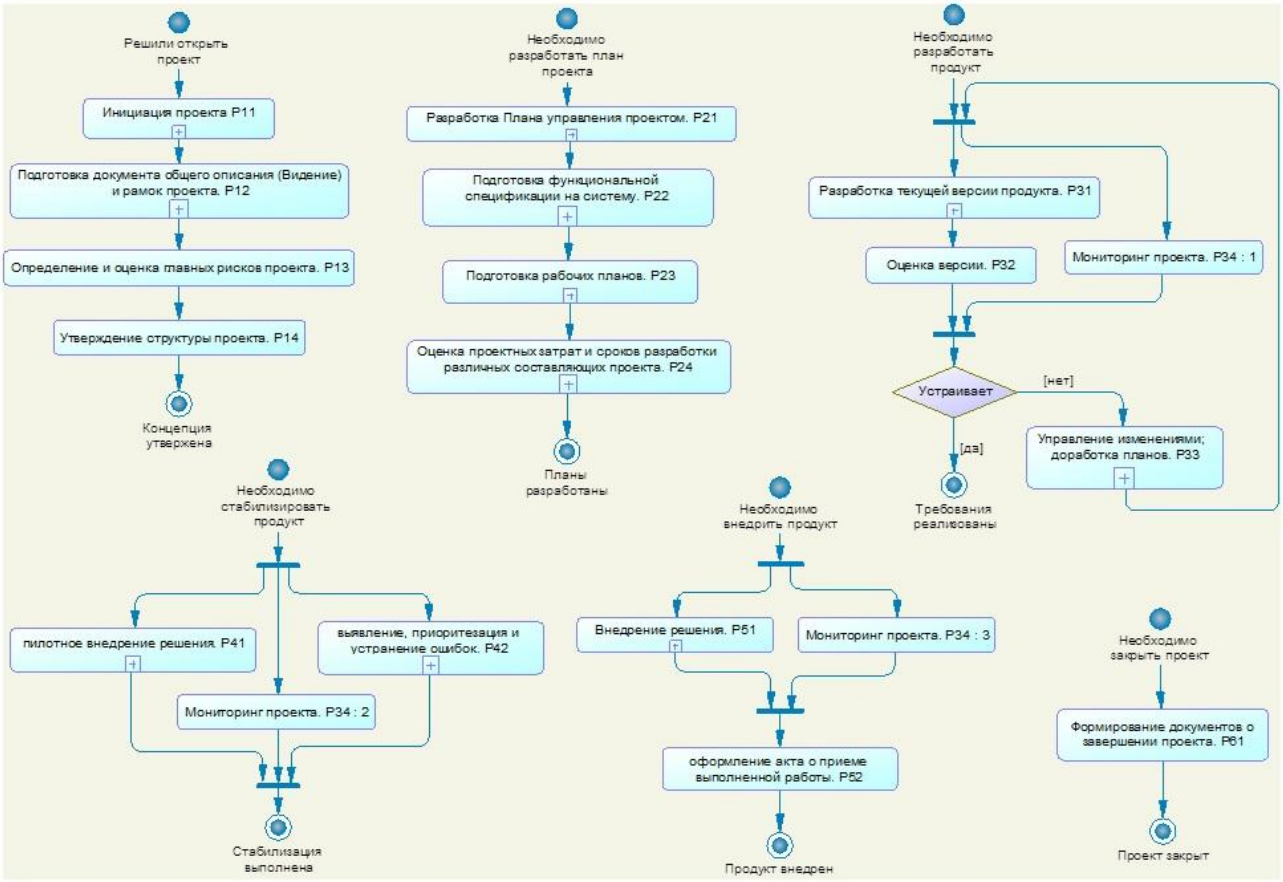

Рисунок 1.4 – Итерационная схема бизнес-процессов моделирования проекта

Испытательные исследования систем. Наряду с развитием академических методов анализа и синтеза прогрессируют и методы испытательного изучения реальных предметов, появляются новые возможности для исследования. Однако опыт был и сохраняется одним из самых важных и существенных средств познания. Ассоциирование и моделирование делают возможным по-новому описать действительный процесс и облегчить его экспериментальное прохождение. Прогрессирует и само понятие моделирования. Если только раньше моделирование определяло реальное физическое исследование либо построение образца, имитирующего фактический процесс, то сегодня появились новые типы моделирования, фундаментом которых служит инсценировка не только вещественных, но также и математических проверок.

Познание того, что происходит вокруг в действительности есть процесс сложный и длительный. Основные вопросы при создании моделей систем сегодня это: как определить насколько качественно функционирует крупная система, как выбрать подходящие алгоритмы поведения и структуру такой системы, как построить системы S, чтобы она соответствовала поставленной задаче. Из-за того, что такие вопросы имеют место быть, моделирование можно считать одним из существующих методов, которые используются в проектировании и исследовании крупных систем.

Выводы по главе 1

1. Развитие и внедрение систем управления кастомизацией является важным опытным путем для крупных и средних учреждений, что содействует в повышении продуктивности управления.

2. Использование систем управления кастомизацией модулей ИТ-проекта связано с определенными сложностями, обусловленными плохой осуществимостью использовать одно программное обеспечение под различные организации. Поэтому представляет актуальность разработка системы

управления кастомизацией модулей ИТ-проекта, легко адаптируемой к специфике проектов конкретных заказчиков.

Глава 2 МЕТОДОЛОГИЧЕСКИЕ ОСНОВЫ МОДЕЛИРОВАНИЯ СИСТЕМЫ УПРАВЛЕНИЯ КАСТОМИЗАЦИЕЙ МОДУЛЕЙ ИТ-ПРОЕКТА

2.1 Методология моделирования системы управления кастомизацией модулей

Моделирование — это описание явление действительности моделью для накапливания информации о нем через проведение исследований с его макетом.

Используя понятие «моделирование», обычно имеют ввиду такой процесс планирования и имплементации, в результате которого на выходе получается система, которая максимально точно отображает предположения, вложенные в планирование. В дополнение к этому можно сказать, что моделирование — это еще и метод познания, в который входят различные компоненты и процессы, примером которых могут служить создание моделей и исследование моделей.

Моделирование адаптирует изучение предмета для покрытия его создания, нужных реконструкции и основания. Оно мобилизуется для исследования рабочей системы, если фактический опыт проводить бесполезно из-за огромных имущественных и трудовых расходований, а также необходимости осуществления анализа планируемой системы, иными словами, которая в данное время вещественно не существует в данном заведении.

В процессе формирования модели очень часто имеют место такие алгоритмы и схемы, как:

- топологическое распределение технических средств и ее схема;
- шаблоны различных функциональных алгоритмов системы;
- структурная схема системы или объекта;
- схема связи;
- система или объект в виде структурно-функциональной схемы и т.п.

Концептуальное моделирование является структурированной процедурой формирования систем, образовывающийся из следующих этапов:

1. Анализ (исследование).
2. Проектирование (дизайн).
3. Кодирование (программирование).
4. Тестирование (отлаживание).
5. Внедрение (ввод в эксплуатацию).

Работу системы в виде алгоритмов можно описать, используя одну из форм системного анализа, который применяется к крупным и сложным системам. Такой формой является имитационное моделирование. В большинстве случаев такая форма применяется тогда, когда необходимо изучить то, как функционируют различные процессы в зависимости от условий либо когда количество входных данных составляет очень большой объем. Одним из ее плюсов является то, что имитация может быть приостановлена на любом из этапов, если необходимо провести научный эксперимент и описать его. В дальнейшем результаты такого опыта можно будет использовать в качестве дополнительных входных данных для других этапов процесса имитации, но только после их обработки и оценки.

Существует ряд методов и принципов разработки ИС, в числе которых имеется возможность выдвинуть сопутствующие: методы «снизу-вверх» и «сверху-вниз», принципы «дуализма», многокомпонентности и прочие.

Модель, которую создают по примеру реального явления, объекта или процесса, содержащая в себе информацию о свойствах происхождения или о самом объекте, явлении или процессе, принято называть информационной моделью.

Обычно для изображения текстовых информационных моделей практикуются естественные языки.

Наряду с естественными языками (русский, английский и т.д.) развиты и применяются на практике формальные языки: системы счисления, алгебра высказываний, языки программирования и другие.

Формальные языки обязаны иметь зафиксированный алфавит, а правила построения команд и слов неотступно следуют правилам синтаксиса и грамматики. Этим все формальные языки и отличаются от естественных, в которых таких четких и обязательных к исполнению рамок и правил нет.

С помощью формальных языков описывают информационные модели известного рода — формально-логические модели.

На рисунке 2.1 представлена схема информационных связей процесса проектирования.

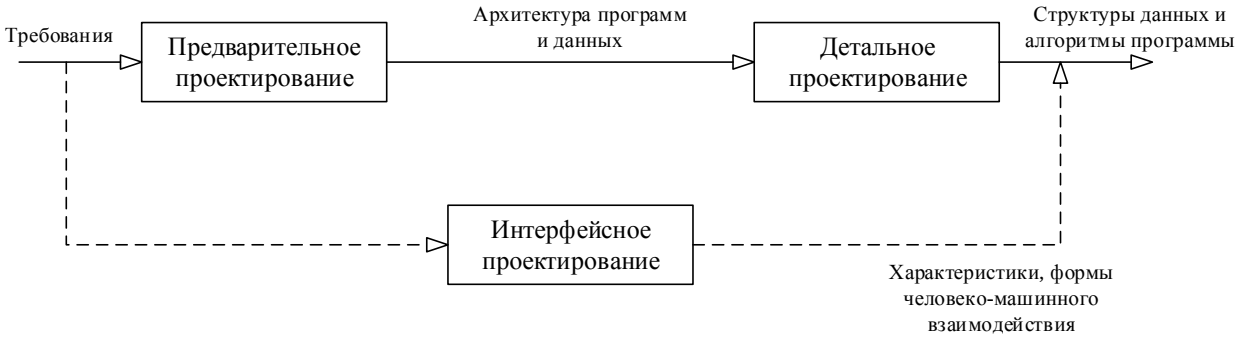

Рисунок 2.1 – Информационные связи процесса проектирования

При проработке нового объекта сначала по большей части устанавливается его упрощенная модель, после она делается формальной, иными словами, обозначается с использованием математических формул, геометрических объектов и т.п.

Процесс построения всех информационных моделей, в котором используются формальные языки, называет формализацией.

Модели, организованные посредством математических оснований и формул, называют математическими моделями.

Метод «снизу-вверх». Когда-то все приемы работы не иностранных ИТ-специалистов были организованы на крупных электронно-вычислительных предприятиях либо вычислительных центрах. Самым важным заданием в таком

роде центров было осуществление узконаправленных поручений какого-либо определенного учреждения, но не труды над разработкой тиражируемых творений. Современные начальники почти всегда обращаются к такому подходу, рассчитывая, что куда полезнее и удобнее располагать собственными профессионалами. Метод производства программ «снизу-вверх» благоприятствует автоматизировать лишь некоторые рабочие процедуры, при всем том, что вышеозначенный метод привлекается профессиональными ИТ-специалистами. Данный путь является как никогда расходным, и его норовят не практиковать сегодня, именно от него силятся отойти малые и средние экономические субъекты.

Метод «сверху-вниз». Развитие негосударственных и иных современных направлений послужило смыслом формирования рынка разнообразных программных средств. Информационные системы, главными задачами которых была автоматизация различных аспектов предприятия, например, бухгалтерского учета или технологических процессов, развиваются более быстрыми темпами в связи со спросом. Как результат получается, что большинство информационных систем были созданы и внедрены отдельными организациями со стороны или разными командами квалифицированных программистов «сверху». Это произошло потому, что от такой ИС ожидалось покрытие запросов, связанных с разными прикладными областями от многих пользователей.

Программисты, работающие с различными информационными базами данных, были резко ограничены в возможностях из-за подобной идеи. В частности, они лишились возможности выбора различных графических форм, а также в выборе разных алгоритмических подходов к расчетам. Это приводит к выводу, что такая идея перекрыла возможность разработки систем, способных расширятся согласно требованиям. Причиной для этого стало то, что возможности ИС резко ограничиваются жесткими рамками, закладываемыми «сверху». Чтобы избежать подобных недочетов в проектировании и реализации

решений задачи автоматизации организации стало необходимо поменять идеологию построения предназначенных для этого ИС.

Принципы «дуализма» и многокомпонентности. Развитие хозяйствующих субъектов, увеличение численности их отделений и клиентов, улучшение качества предоставления помощи и не только произвели серьезные реконструкции в эксплуатации и деятельности автоматизированных информационных систем, во многом покоящиеся на соразмеренной совокупности двух вышесказанных методов.

На основе анализа определений модели можно символически отобразить определение модели J_{def} как носителя информации об оригинале (формула 2.1):

$$J_{def} = <M, N, Z, IS, L>, \quad (2.1)$$

где M — оригинал (моделируемое явление, объект, источник информации); N — субъект ("наблюдатель" по Эшби), т.е. лицо, которому потребовалась информация об оригинале для достижения своей цели (исследования, принятия решения и т.п.);

Z — цель или совокупность целей;

IS — инфраструктура, обеспечивающая моделирование, т.е. включающая технологии и условия моделирования: TECH = {METH, MEANS, ALG, COND, ...} — совокупность технологий (METH – методы, MEANS – средства, ALG – алгоритмы и т.п.), реализующих модель; COND = $\{\varphi_{ex}, \varphi_{in}\}$ – условия существования модели, т.е. факторы, влияющие на ее создание и функционирование (φ_{ex} – внешние, φ_{in} – внутренние);

L — язык для исследования гносеологических аспектов отношения "модель – оригинал".

Выбирать тип модели можно разными способами, но необходимо понимать, что этот выбор зависит не только целей моделирования. Свойства исходных данных, например, объем и характер данных, также сильно влияют на этот выбор. Еще одним важным фактором в выборе типа модели является ограничение в возможностях самого исследователя.

Как следует собранная модель определяется понятиями, которым ей приходится удовлетворять. В таблице 2.1 приведены основные принципы имитации информационных систем управления.

Таблица 2.1 – Принципы имитации информационных систем управления

Принципы имитации	Наименование	Характеристика принципов
	Адекватность	Соответствие моделируемой системы управления целям и задачам предприятия заказчика.
	Согласованность	Система управления строится для решения точного ряда задач. Начальные два принципа скоррелированы.
	Адаптивность	Система управления должна в полной мере подходить всем аспектам активности предприятия заказчика.
	Точность	Система управления должна быть настолько точной, чтобы вырабатываемые решения не оказались слишком затруднительными.

Продолжение таблицы 2.1 – Принципы имитации информационных систем управления

	Сбалансированность	Система управления должна заключать в себе все сферы функционирования, в которой она будет применяться для исключения неохваченных областей.
	Многовариантность	Система должна иметь заготовленные на случай нужды пути реализации, если что-то пойдет не так.

Так как анализ позволяет установить какие-то ключевые направленности деятельности системы, то приступать к формированию информационной системы управления необходимо на основе этого анализа, примененного к исходным данным, полученным на разных уровнях управления, которые либо уже известны, либо могут быть получены.

2.2 Разработка логической модели системы управления кастомизацией модулей ИТ-проекта

На принципах особенностей процедуры урегулирования проектных проблем пользователем выработана концептуальная модель системы управления. Модель описывает перечень целей, требуемый для решения проблем, сопряженных с проектной деятельностью.

Диаграмму вариантов использования можно использовать в данном случае как объектную модель системы управления кастомизацией модулей. На такой диаграмме несложно отобразить основные системные процессы и связи между такими процессами.

Помимо прочего, с помощью диаграммы прецедентов можно отобразить всю структуру упомянутой системы. Используя эту диаграмму в качестве базы,

можно составить план следующих шагов по проектированию системы, управляющей кастомизацией модулей.

Созданная диаграмма вариантов использования предложена на рисунке 2.2.

Рисунок 2.2 – Диаграмма прецедентов системы управления кастомизацией модулей

На диаграмме представлены следующие действующие лица (актеры):

- система управления кастомизацией модулей — блок классов, который является прослойкой между пользователем и основными классами, входящими в состав системы управления кастомизацией модулей;

пользователь — лицо, являющееся сотрудником компании-заказчика, ответственное за настройку и эксплуатацию данной системы управления кастомизацией.

В таблице 2.2 приведена краткая характеристика прецедентов проектируемой системы управления кастомизацией.

Таблица 2.2 – Прецеденты

Прецедент	**Характеристика**
Отправить запрос на подключение плагина	Нажатие пользователем кнопки «Подключить плагин»
Отправить запрос на отключение плагина	Нажатие пользователем кнопки «Отключить плагин»
Подключить плагин	Процесс подключения плагина после нажатия кнопки «Подключить плагин» или иного события, являющегося частью кода
Отключить плагин	Процесс отключения плагина после нажатия кнопки «Отключить плагин» или иного события, являющегося частью кода
Подключить плагины автоматически при загрузке приложения	Процесс подключения плагинов, имеющих свойство автозапуска, на старте приложения

В таблицах 2.3 — 2.7 описаны спецификации основных прецедентов.

Таблица 2.3 – Описание прецедента: Отправить запрос на подключение плагина

Прецедент: Отправить запрос на подключение плагина
ID: 1
Краткое описание: Нажатие на кнопку «Подключить плагин»
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: Прецедент начинается по инициативе пользователя

Продолжение таблицы 2.3 – Описание прецедента: Отправить запрос на подключение плагина

Основной поток: 1. Пользователь нажимает на кнопку «Подключить плагин»
Постусловие: Запрос обрабатывается приложением
Альтернативные потоки: Нет

Таблица 2.4 – Описание прецедента: Отправить запрос на отключение плагина

Прецедент: Отправить запрос на отключение плагина
ID: 2
Краткое описание: Нажатие на кнопку «Отключить плагин»
Главные актеры: Пользователь
Второстепенные актеры: Нет
Предусловие: Прецедент начинается по инициативе пользователя
Основной поток: 1. Пользователь нажимает на кнопку «Отключить плагин»
Постусловие: Запрос обрабатывается приложением
Альтернативные потоки: Нет

Таблица 2.5 – Описание прецедента: Подключить плагин

Прецедент: Подключить плагин
ID: 3
Краткое описание: Нажатие на кнопку «Подключить плагин» или срабатывание системного события
Главные актеры: Приложение
Второстепенные актеры: Нет

Продолжение таблицы 2.5 – Описание прецедента: Подключить плагин

Предусловие: Прецедент начинается по инициативе приложения
Основной поток: 1. Приложение обрабатывает нажатие на кнопку или системное событие 2. Приложение проверяет доступность плагина
Постусловие: Система действует исходя из результатов проверки
Альтернативные потоки: Нет

Таблица 2.6 – Описание прецедента: Отключить плагин

Прецедент: Отключить плагин
ID: 4
Краткое описание: Нажатие на кнопку «Отключить плагин» или срабатывание системного события
Главные актеры: Приложение
Второстепенные актеры: Нет
Предусловие: Прецедент начинается по инициативе приложения
Основной поток: 1. Приложение обрабатывает нажатие на кнопку или системное событие 2. Приложение проверяет доступность плагина
Постусловие: Система действует исходя из результатов проверки
Альтернативные потоки: Нет

Таблица 2.7 – Описание прецедента: Подключить плагин автоматически при загрузке приложения

Прецедент: Подключить плагин автоматически при загрузке приложения
ID: 5
Краткое описание: Подключение плагинов при старте приложения

Продолжение таблицы 2.7– Подключить плагин автоматически при загрузке приложения

Главные актеры: Приложение
Второстепенные актеры: Нет
Предусловие: Прецедент начинается по инициативе приложения
Основной поток: 1. Приложение проверяет необходимость автозапуска плагина 2. Приложение проверяет доступность плагина
Постусловие: Система действует исходя из результатов проверки
Альтернативные потоки: Нет

С помощью построенной диаграммы вариантов использования можно увидеть не только пределы предметной области, подвергшейся моделированию, но и определенные требования к поведенческим функциям проектируемой системы управления кастомизацией. Реализация представленной диаграммы позволит описать функции, включенные в проектируемую систему.

Логическое моделирование — это формирование причинно-следственных соединений между главными факторами, квалифицирующими информационные процедуры, для отражения этих процедур при исследовании и оценке характеристик предметов. Одну из стадий логического моделирования составляет разработка диаграммы последовательности.

На рисунке 2.3 выведена диаграмма последовательности, на которой указано взаимодействие объектов при автоматическом подключении плагинов на старте системы.

Принцип, описанный ниже, показывает, как происходит взаимосвязь между объектами и субъектами:

1. Система управления кастомизацией модулей инициирует загрузку доступных плагинов в систему.

2. Класс загрузки модулей загружает все доступные модули, находящиеся в папке проекта.

3. Класс загрузки модулей инициирует цикл, в котором для каждого модуля осуществляется проверка необходимости автоподключения, для чего в класс управления модулями подается имя модуля.

4. Класс управления модулями осуществляет проверку доступности модуля и проверку необходимости подключения этого модуля, для чего осуществляются запросы к базе данных.

5. Если обе проверки вернули значение «ИСТИНА», класс управления модулями инициирует отрисовку функционала модуля на графической форме, отправляя необходимую для этого информацию в менеджер форм графического интерфейса.

6. Цикл начинает следующую итерацию. Это происходит до тех пор, пока все загруженные плагины не пройдут проверку.

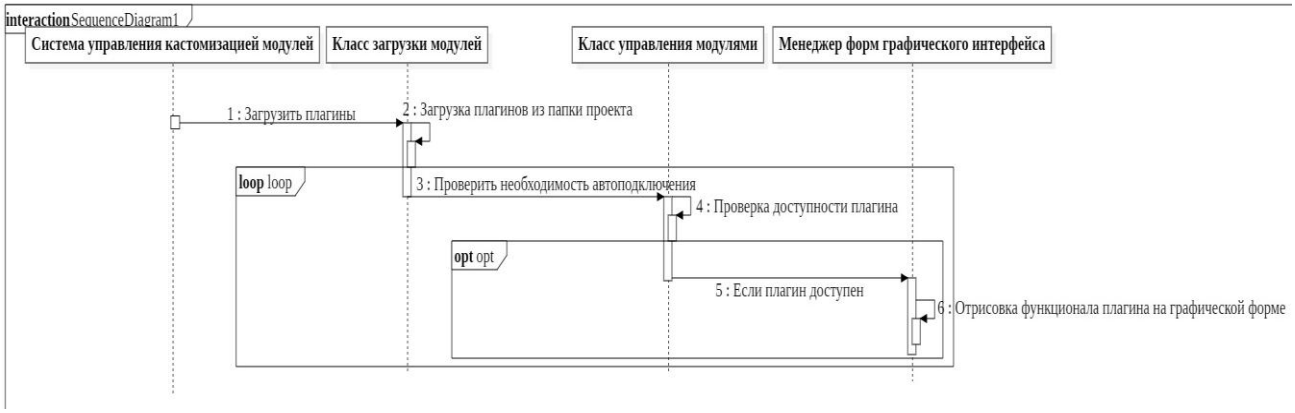

Рисунок 2.3 – Диаграмма последовательности сценария управления стартом системы

На рисунке 2.4 изображена диаграмма последовательности, на которой описано взаимодействие объектов при подключении плагинов после нажатия пользователем на кнопку «Подключить плагин».

Принцип, описанный ниже, показывает, как происходит взаимосвязь между объектами и субъектами:

1. Пользователь нажимает на кнопку «Подключить плагин» на графической форме.

2. Менеджер форм графического интерфейса обрабатывает событие добавления нового плагина.

3. В рамках пункта 2 инициируется проверка плагина на доступность, для чего данные передаются в класс управления модулями

4. Класс управления модулями осуществляет проверку доступности плагина путем запросов к базе данных.

5. Если проверка плагина на доступность прошла успешно, то класс управления модулями передает эту информацию обратно и инициирует отрисовку плагина.

6. Менеджер форм графического интерфейса производит отрисовку функционала модуля на форме.

7. Если проверка плагина на доступность провалилась, то класс управления модулями возвращает значение «ЛОЖЬ».

8. Менеджер форм графического интерфейса показывает пользователю сообщение об ошибке.

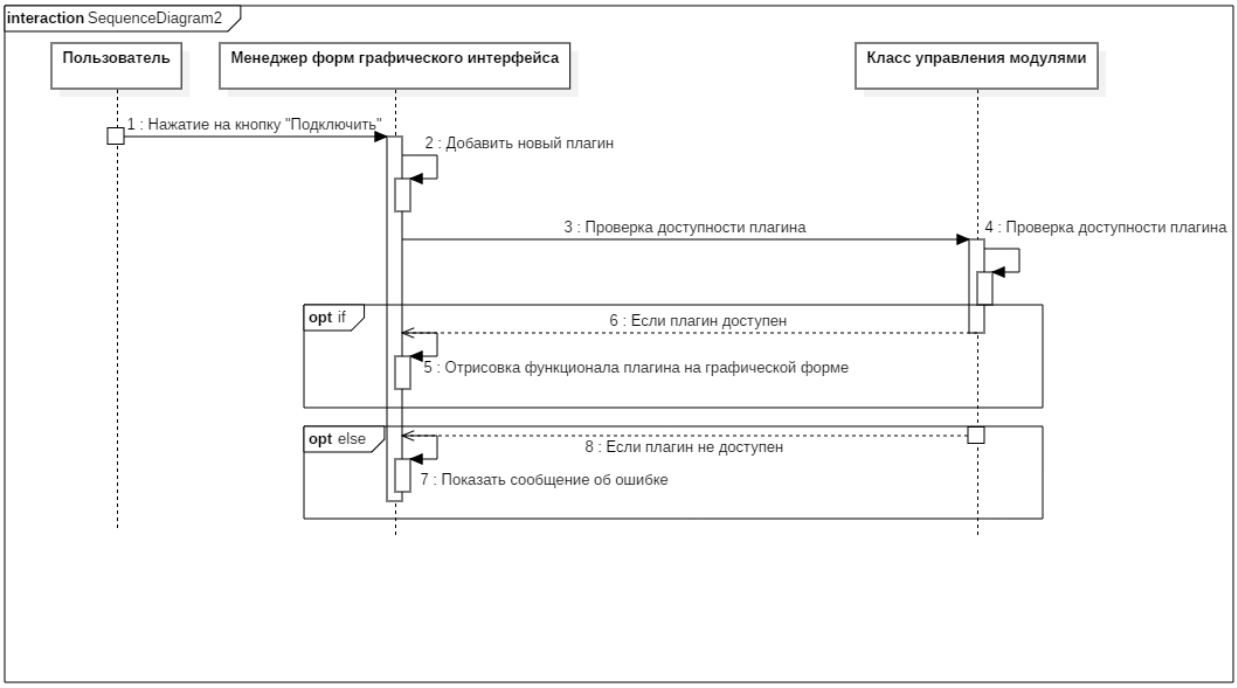

Рисунок 2.4 – Диаграмма последовательности сценария управления подключением плагина

Следовательно, можно увидеть, что проверка логики действий для системы была успешно проведена, а также объяснена последовательность работы системы управления кастомизацией модулей.

С помощью данной логической модели было произведено описание всех используемых понятий из предметной области и связи между ними. Также можно увидеть описанные ограничения, наложенные на данные исходя из выбранной предметной области. Данная модель будет использована в качестве прототипа для реальной базы данных проекта и на текущем этапе не привязана ни к какой СУБД.

На рисунке 2.5 представлена логическая модель данных системы управления кастомизацией модулей, которая показывает все сущности с атрибутами, первичными ключами сущностей и отношениями между ними.

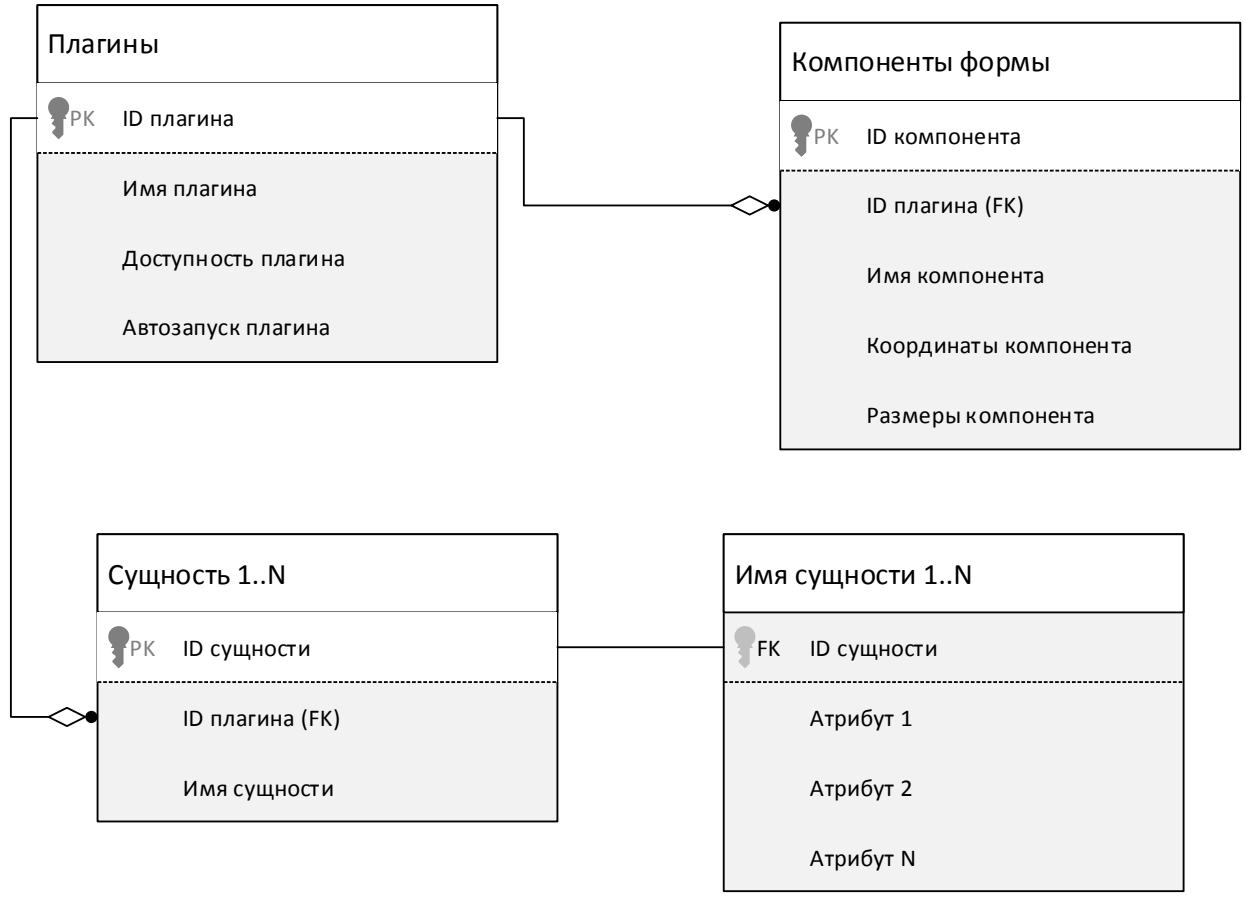

Рисунок 2.5 – Логическая модель данных системы управления кастомизацией модулей

Ниже приведено краткое описание сущностей с атрибутами.

- Сущность «Плагины» с атрибутами: «ID плагина», «Имя плагина», «Доступность плагина», «Автозапуск плагина». Атрибут «ID плагина» выступает в роли первичного ключа.
- Сущность «Компоненты формы» с атрибутами: «ID компонента», «ID плагина», «Имя компонента», «Координаты компонента», «Размеры компонента». В этой сущности первичным ключом является атрибут «ID компонента», а атрибут «ID плагина» выступает в роли внешнего ключа.
- Сущность «Сущность1..N» с атрибутами: «ID сущности», «ID плагина», «Имя сущности». Здесь в роли первичного ключа выступает атрибут «ID сущности», а в роли внешнего ключа – атрибут «ID плагина».
- Сущность «Имя сущности 1..N» с атрибутами: «ID сущности», «Атрибут 1», «Атрибут 2», «Атрибут N». В данной сущности отсутствует первичный ключ, а в роли внешнего ключа выступает атрибут «ID сущности».

При помощи диаграммы классов моделируется статическая структура системы. Данная диаграмма призвана показать, как относятся между собой классы, атрибуты, объекты и операции.

В этом типе диаграмм классы представляются в виде абстракций объектов с похожими, связанными между собой характеристиками. Эти связи на диаграмме классов и показывают отношение классов между собой.

Ниже показана диаграмма классов системы управления кастомизацией модулей, представленной на рисунке 2.6.

Далее приведена спецификация представленной диаграммы классов:

- «Класс загрузки модулей» — класс, использующийся для загрузки доступных плагинов из папки проекта в систему для последующих манипуляций. Хранит имя плагина и его свойства.
- «Класс управления модулями» — класс, который служит для любых манипуляций с модулями на уровне приложения, а также осуществляет

подключение к базе данных и манипуляцию с данными в этой базе. Хранит имя и свойства всех загруженных плагинов в коллекции.

- Класс «Менеджер форм графического интерфейса» — класс, использующийся для отрисовки формы и служащий для общения приложения с пользователем.

Рисунок 2.6 – Диаграмма классов системы управления кастомизацией модулей

Очень важным преимуществом построения диаграммы классов в процессе логического проектирования системы управления кастомизацией модулей является возможность переноса объектов классов этой диаграммы на логическую модель базы данных.

На рисунке 2.7 представлен алгоритм проверки необходимости автоподключения плагина.

В данной блок-схеме описан алгоритм, по которому программа определяет необходимость подключения плагина и отрисовки его функционала на форме графического интерфейса в момент запуска программы.

Для того, чтобы осуществить подобную проверку, необходимо первым делом загрузить плагины. Каждый плагин является отдельным объектом со своим уникальным набором свойств, которые помещаются в коллекцию,

хранящуюся в классе, который и будет осуществлять проверку необходимости автоподключения каждого плагина.

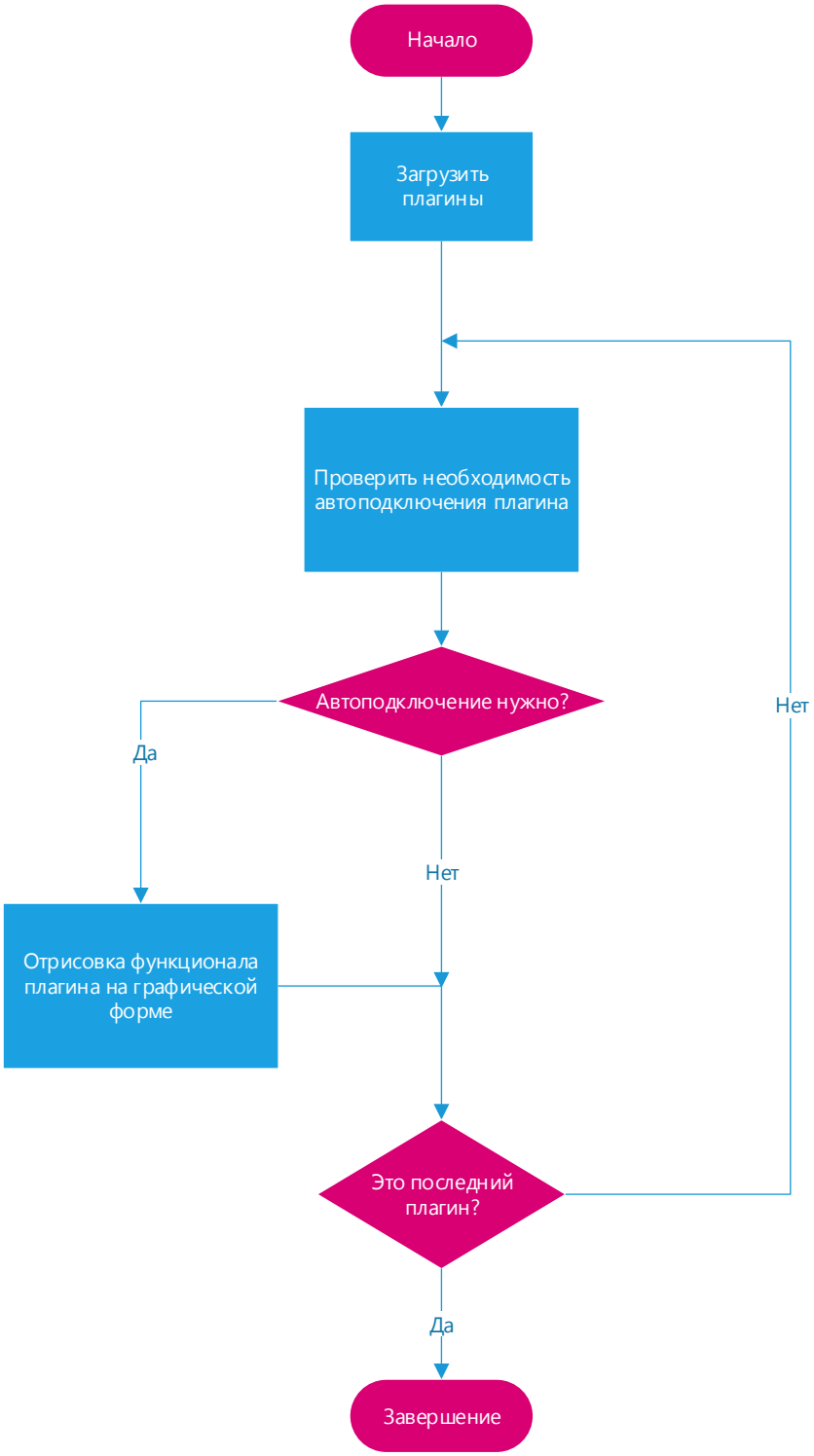

Рисунок 2.7 – Блок-схема алгоритма проверки необходимости автоподключения плагина

Сама же проверка необходимости автоподключения плагина заключается в опросе базы данных, в которой и хранится необходимое для успешного осуществления проверки значение параметра.

В случае, когда метод проверки необходимости автоподключения возвращает отрицательное значение, программа переходит к проверке содержимого коллекции на наличие следующего плагина в списке. Если текущий плагин не является последним в коллекции, программа переходит к началу цикла и начинает осуществлять проверку для следующего плагина в коллекции. В противном случае, программа выходит из цикла проверки автоподключения и завершает данный этап своей работы.

В случае возвращения методом проверки необходимости подключения плагина при старте приложения положительного значения, программа инициирует процесс отрисовки плагина и его функционала на форме графического интерфейса.

На рисунке 2.8 представлен алгоритм процесса подключения плагина и его функционала, который был инициирован путем срабатывания определенных системных событий.

Процесс начинается с загрузки плагина и его свойств в класс, который будет осуществлять различные проверки, необходимые для корректного подключения плагина в системе.

Далее классом осуществляется запрос в базу данных для загрузки свойств плагина, специфических для конкретного ИТ-проекта. Такие свойства необходимы для более простой и гибкой настройки функционала программы, потому что изменять данные свойства можно путем использования простых запросов в базу данных.

После загрузки свойств плагина из БД управляющим классом осуществляется проверка доступности плагина для подключения.

В случае, когда метод, реализующий проверку, возвращает отрицательный результат, программа должна отобразить сообщение о

недоступности данного плагина для подключения к основному проекту. Затем информация на форме графического интерфейса обновляется, и программа завершает процесс подключения модуля.

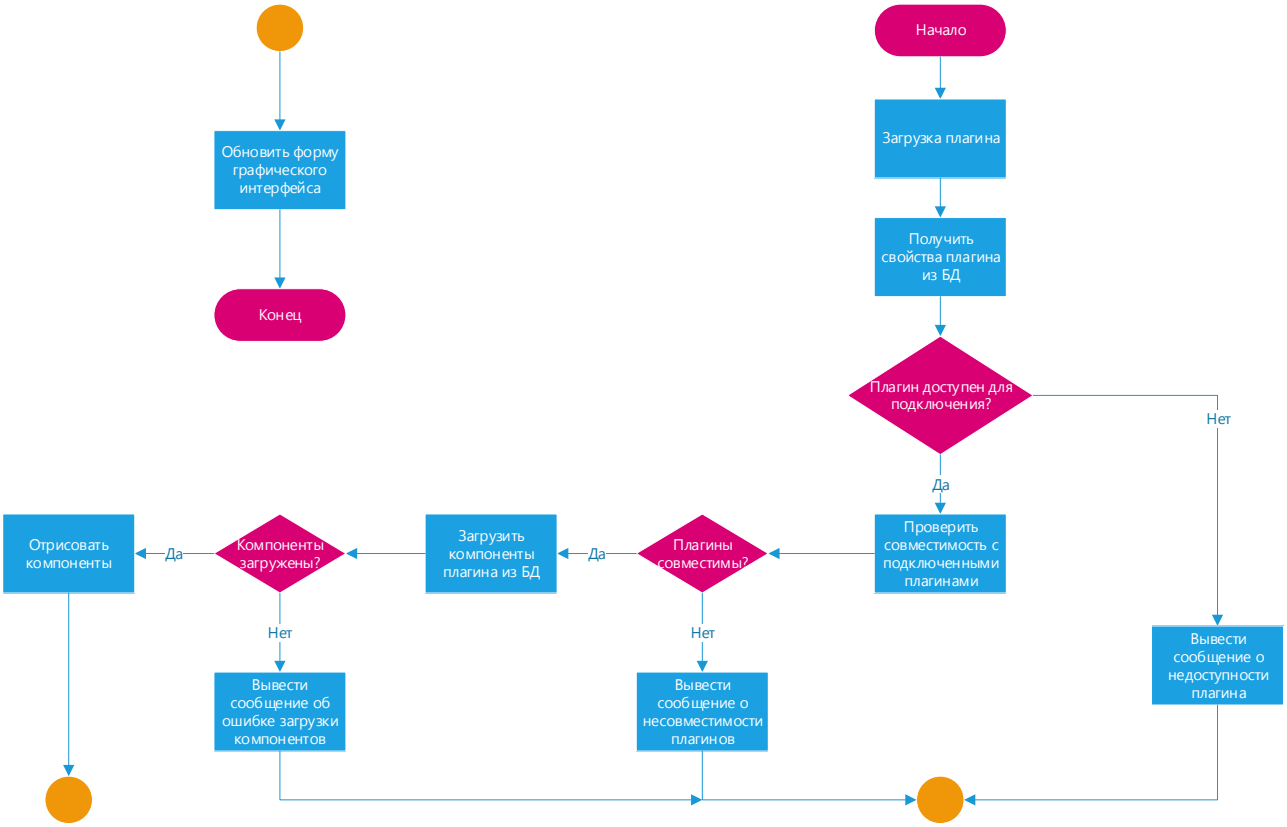

Рисунок 2.8 – Блок-схема алгоритма процесса подключения плагина и его функционала

Если проверяющий метод вернул положительное значение, в работу вступает метод для проверки на совместимость с уже подключенными ранее плагинами.

Этот метод сравнивает основные свойства выбранного модуля, отвечающие за совместимость с такими же свойствами подключенных ранее плагинов.

Если находится хотя бы одна несовместимость, то метод проверки на совместимость возвращает отрицательное значение. В этом случае программа выводит сообщение, призванное сообщить пользователю о несовместимости выбранного им модуля с одним или несколькими модулями. После чего

происходит обновление информации на форме графического интерфейса и завершение процесса подключения плагина.

Когда несовместимостей в ходе проверки не находится программа передает управление методу, который загружает компоненты плагина из базы данных для дальнейшей работы с ними. В ходе этого метода выполняется проверка этих компонентов.

В случае, если не все компоненты были загружены корректно, либо какие-то из компонентов отсутствуют, программа выводит сообщение об ошибке загрузки компонентов, которая содержит информацию о компонентах, которые стали причиной ошибки.

Так как случай с ошибкой загрузки компонентов очень редок, в большинстве случаев метод загрузки компонентов передаст информацию о компонентах в метод, который предназначен для отрисовки этих компонентов на форме графического интерфейса. Данный метод расставляет компоненты на форме исходя из их свойств, хранящихся в таблице в базе данных.

После отрисовки компонентов будет вызван метод обновления информации на форме, чтобы применить все изменения, сделанные в предыдущем методе. И затем программа завершает процесс подключения нового плагина.

В процессе подключения плагина одной из важных процедур является его отрисовка. На рисунке 2.9 изображен алгоритм данного процесса.

Все начинается с загрузки компонентов и их свойств в класс, ответственный за отрисовку модулей ИТ-проекта. После чего программой инициируется цикл отрисовки компонентов.

Сначала проверяется, что коллекция компонентов непустая. После чего вызывается метод, который выставляет компоненту правильную позицию на форме, исходя из свойств этого компонента.

Далее вызывается метод, задающий размеры компонента, который также обращается к свойствам компонента.

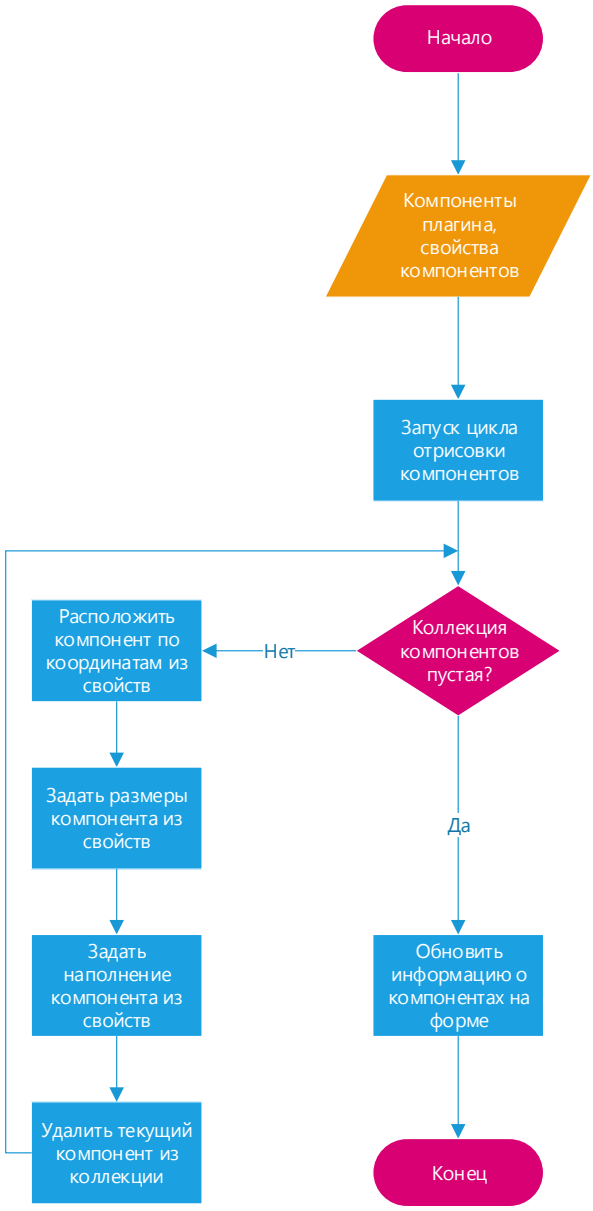

Рисунок 2.9 – Блок-схема алгоритма отрисовки компонентов плагина

Впоследствии компоненту задаются второстепенные свойства, например, цвет, текст, изображение и т.п.

Окончанием сей итерации цикла является удаление оного компонента из коллекции и возвращение к началу цикла.

После того, как все компоненты были успешно отрисованы на форме графического интерфейса и коллекция компонентов пуста, запускается очередной метод, который обновляет информацию о компонентах на данной форме.

Выводы по главе 2

1. Моделирование ИТ-проекта — сложный, ресурсоемкий процесс, который нуждается в автоматизации с использованием системы управления кастомизацией ИТ-модулей.

2. Было выполнено функциональное моделирование системы управления, спроектирована логическая модель данных, а также построены диаграмма прецедентов, диаграммы последовательности, диаграмма классов и блок-схемы основных алгоритмов работы для наглядного представления о моделируемой системе управления кастомизацией модулей ИТ-проекта.

Глава 3 ПРОЕКТИРОВАНИЕ И АНАЛИЗ РЕШЕНИЯ СИСТЕМЫ УПРАВЛЕНИЯ КАСТОМИЗАЦИЕЙ МОДУЛЕЙ ИТ-ПРОЕКТА

3.1 Процедура моделирования стандартного ИТ-проекта

В наши дни, когда информационные технологии все больше входят в повседневную жизнь, программное обеспечение, настроенное под конкретную организацию, требуется практически для любой фирмы, будь то крупный или малый бизнес. Подавляющее большинство проектов делается только для конкретного заказчика и ни коим образом не предполагает кастомизации под другие похожие проекты. А процедура доработки под определенный проект занимает слишком много времени и трудовых и денежных ресурсов.

Из раза в раз при имплементации проектов внедрения возникают неразрешенные проблемы:

- увеличение объема процессов и процедур;
- снижение скорости работы системы;
- невозможность использования системы на полную мощность;
- растягивание процессов внедрения на долгий срок;
- частые изменения требований заказчика и т.п.

Почти каждый ИТ-проект сталкивается с этими проблемы во время внедрения. Об этом известно уже давно, но точного ответа на вопрос почему так происходит нет до сих пор. Одними из возможных причин могут быть недостаточно четкие исходные требования заказчика, недостаточное качество управления проектом. Суммируя эти причины, на выходе получаем то, что проект почти не совпадает с реальными требованиями. Из чего напрашивается вывод о том, что во время внедрения проекта какие-то важные аспекты были упущены.

В условиях жесткой конкуренции ИТ-проект можно рассматривать как несомненный способ улучшить управленческие и экономические показатели деятельности организации.

В любой организации ИТ-проект стоит рассматривать с точки зрения, когда он является неотъемлемой частью крупной системы. Причиной такой тесной связи проекта с бизнес-процессами является то, что на сегодняшний день такие проекты нужны абсолютно везде. Но из-за такой тесной связи разворачивание нового или обновленного старого ИТ-проекта абсолютно все упомянутые процессы будут затронуты. Организационная структура этого предприятия тоже не останется нетронутой.

В довершение всего, одна из специфичных черт ИТ-проекта — это наличие модификаций, которое задевает не исключительно условия претворения проекта в жизнь, да и функцию и/или качественные параметры. Поэтому до того, как приняться за воплощение, формируется техническая документация, в которой ясно обозначаются назначение проекта, способы его определения и соответствующие параметры, как то в формате технического задания. В этом есть необходимость, дабы при проведении воплощении в жизнь и ликвидации проекта вправе было разобрать по косточкам выводы его осуществления, а мерками такого анализа прямо и представляются вышеперечисленные величины и значения. Затем установка, которую можно диагностировать, направлена на то, чтобы увернуться от субъективных позиций в анализе процентов выполнения проекта.

Ниже приведен список особенностей практически любого ИТ-проекта:

1. Процесс внедрения любого ИТ-проекта в функционирующую структуру предприятия всегда подразумевает под собой изменения в этой структуре.

2. В большинстве случаев к разработке и внедрению нового ИТ-проекта привлекаются сторонние компании или разработчики.

3. Внедрение нового ИТ-проекта всегда сопряжено с рисками.

4. Очень часто ИТ-проект оказывает влияние не все или почти все структурные подразделения предприятия.

Из-за того, что этих особенностей избежать невозможно, возникает острая необходимость в применении различных методов управления проектами в процессе реализации любого ИТ-проекта.

Взаимосвязи между субъектами, осуществляемые бизнес-логикой и зафиксированные в череде бизнес-правил, и образовывают деятельность заведения. ИС "отражает" выразительность понятий и правила, сводя и реформируя информационные множества, автоматизирует последовательности действий с данными и информацией и строит итоги в виде пакетов отчетных форм. Поэтому сначала следует составить модель компании, которая видится отображением предприятия и ее информационно-управляющей системы.

Бизнес-цели любого предприятия в подавляющем большинстве случаев полностью определяют состав большинства возможных компонентов любой модели и лежат в основе этой модели:

- бизнес-функции, описывающие то, ЧТО делает бизнес;
- то, КАК выполняются бизнес-функции компании, описывается различными процессами;
- организационно-функциональная структура определяет, ГДЕ будут выполняться те или иные процессы и функции;
- роли, которые определяют, КТО будет функции исполнять, а КТО будет «хозяином» процессов;
- фазы, из которых будет понятно, КОГДА внедрять все эти функции;
- взаимосвязь между все КАК, ЧТО, КТО, ГДЕ и КОГДА определяется правилами.

Опыт разработки и использования кастомизированных ИС делает возможным условно выделить нижеследующие основные стадии их жизненного цикла:

- анализ требований к системе определяет то, что должно осуществляться этой системой;
- проектирование представляет собой различные спецификации подсистем и компонентов и их взаимосвязь, необходимые для того, чтобы точно сказать, что должна система производить;
- разработка — программное решение задачи о формировании компонентов и подсистем, а также сборка подсистем в единую связанную систему;
- тестирование — сравнение показателей, полученных после проверки работоспособности системы, с показателями, заданными на этапе оценки;
- внедрение — процесс внедрения системы в эксплуатацию;
- функционирование — непосредственно процесс эксплуатации, проводящийся согласно основным задачам ИС;
- сопровождение — обеспечение процесса эксплуатации в штатном режиме на предприятии заказчика.

Главная функция системного анализа составляет преобразования каких-либо общих познаний о первоначальном плане (скажем, требований клиента) в правильно сформулированные обозначения и указания для кодировщиков, среди прочего и процедура создания функционального отображения программы. Особенно этот этап берется тем уровнем, на ком ставят и специфицируют:

- не исключительно внутренние, но также внешние правила работы программного обеспечения;
- функциональная характеризация программы;
- порядок разделения функций между софтом и пользователем и интерфейсы;

- конкретные запросы к такого рода частям системы, как информационные, программные и технические;
- требования как к качеству, так и к безопасности;
- пользовательская и техническая документация и требования по ее форме;
- эксплуатационные и внедренческие условия.

Первый этап анализа — структурный анализ компании — наступает с ознакомления с тем, как составлена система управления предприятием, с рассматривания функциональной и информационной организации системы управления, формирования действующих и вероятных покупателей информации.

После анализа аналитик строит обобщенную логическую модель на основании исходных требований, отображающих ее структуру. В то же время эта модель показывает принцип основной деятельности и предметную область выполнения. На основании этой модели аналитиком проектируется модель «Как есть» (As-Is).

Следующая стадия создания осуществляется с привлечением представителей клиента, знающих требования к системе. Также могут быть приглашены независимые консультанты. Сама стадия осуществляется путем разбора построенной модели «Как есть». На этой стадии раскрываются уязвимые и слабые участки существующей системы и определяются пути и методы их укрепления на основании требований заказчика до необходимого уровня.

Третьей фазой анализа, каковая должна охватывать элементы конструирования, является создание более полной распространенной логической модели, которая бы выставила предметную область, претерпевшую реорганизацию, или некоторые элементы этой области, какая равным образом подходит для автоматизации — модель «Как должно быть» (To-Be).

Четвертой и последней фазой процедуры анализа является процесс разработки «Карты автоматизации». Она представляет из себя макет обновленной предметной области и содержит в себе четкие «границы автоматизации».

В подавляющем большинстве случаев при проектировании системы стадия анализа требований предполагает под собой введение:

- диаграмм бизнес-транзакций и соответствующие им типы пользователей;
- моделей, которые соответствуют разным используемым методам в прикладной деятельности, а также списки соответствующих задач;
- классы объектов необходимой области и отражающие суть информационной модели диаграммы, называемые диаграммами «сущность-связь»;
- топологии того, как расположены пользователи и подразделения, которые обслуживаются этой ИС;
- параметров самой системы, защиты данных и параметры информации.

Следующий пункт — проектирование. В данном положении проектирование — это отыскание и моделирование путей разработки, который подходит для добавления функциональности системе методами имеющихся в наличии технологий в свете отмеченных начальных данных и норм. Проектирование ИС всякий раз начинается с формулировки назначения проекта. Основная задача всякого удачливого ИТ-проекта определяется тем, чтобы к началу приема системы и в течение всех дней ее применения можно было обеспечить:

- требуемую работоспособность системы и величину адаптации к изменяющимся принципам ее работы;

- предписываемую производительность ИС и минимальное время реакции ИС на запрос;
- бесперебойную эксплуатацию ИС в требуемом режиме, компетентность и открытость ИС для обработки запросов пользователей;
- недвусмысленность использования и сопровождения ИС;
- нужную ИТ-безопасность данных и права доступа пользователей.

Проектирование любой информационной системы обычно охватывает три области:

- область, связанная с проектированием структуры данных, которые будут реализовываться в базе данных;
- область, связанная с разработкой программ, графических форм и различных отчетов для осуществления обращений к базе данных;
- область, связанная с моделированием среды. К этому относит проектирование архитектуры, топологии сетей и т.п.

По результатам проведенного системного анализа происходит переход к стадии промежуточного проекта, на которой улучшаются:

- проект внедрения программно-аппаратного окружения, а также проекты, связанные с работой пользователей в системе;
- спецификации сети и архитектура системы;
- диаграммы, которые описывают потоки данных, присутствующих в данной модели;
- программное обеспечение системы и блок-схемы прикладного обеспечения (первые — согласно допущенным моделям среды ИС и профилями стандартов).

Стадия детального проектирования подразумевает под собой разработку:

- проекта, в котором будет реализована среда ИС и комплекс ее функциональных программ;

- средств, предназначенных для ведения БД, а также структуры данных;
- входящих в состав ИС сетевых адресов, протоколов обмена информацией и прочих всякого рода составляющих;
- правил, направленных на дифференцирование доступа для пользователей, кроме этого, средств претворения в жизнь этих правил.

Разработка и тестирование как всей системы, так и ее отдельных компонентов входят в стадию внедрения информационной системы.

На фазе использования функциональности системы и ее технической поддержки предусматривается управление функционированием вышеозначенной ИС, простановка необходимых трансформаций в БД текущей системы в рамках создания и улучшение назначения самой системы, которое выполняется посредством прикладных ИТ-специалистов, использующих дополнительные средства, какие вмонтированы в систему.

Этапы разработки модели ИТ-проекта, которые были рассмотрены выше, показывают, что это очень сложный, ресурсозатратный и долгий процесс, который нуждается в повышении эффективности. Следовательно, имеет место быть разработка модели системы управления кастомизацией модулей ИТ-проекта, что позволит сократить время и ресурсы на проектирование и имплементацию связей между сущностями, а также на доработку и переделывание кода после внесения изменений в требования заказчика.

3.2 Разработка программного обеспечения системы управления кастомизацией модулей ИТ-проекта

Процесс разработки программного обеспечения был организован с учетом всех ранее построенных диаграмм.

Далее приведены основные экраны системы управления кастомизацией модулей ИТ-проекта.

Основным экраном представляется главное окно системы управления кастомизацией модулей ИТ-проекта, появляющееся при запуске системы и предложенное на рисунке 3.1.

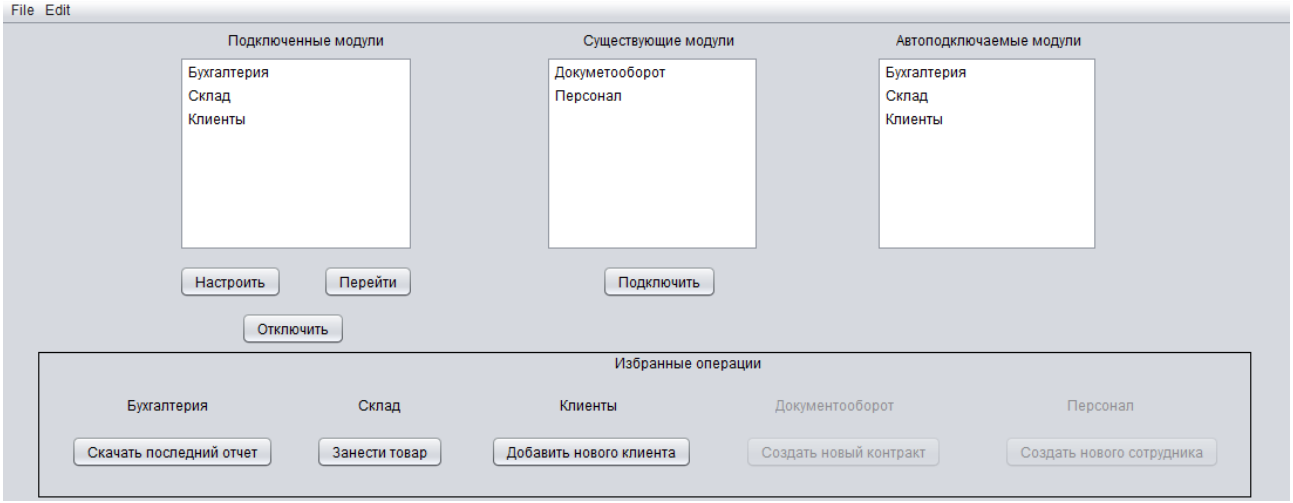

Рисунок 3.1 – Основное окно системы управления кастомизацией модулей ИТ-проекта

Обращаясь к этому окну, пользователь может настроить подключенные модули, перенаправиться к функционалу конкретного плагина, а также просмотреть список автоподключаемых модулей. Ко всему прочему, в этом окне можно увидеть доступные для подключения плагины, подключить один или несколько из них, а также отключить любой из уже добавленных ранее.

В свой черед в данном окне представлена панель с избранными операциями для каждого модуля. Используя кнопки на этой панели, пользователь может быстро перейти к часто используемому функционалу каждого плагина.

Ниже приведен функционал настройки подключенного модуля на примере модуля «Клиенты», который изображен на рисунке 3.2.

В данном окне существуют самые важные настраиваемые элементы модуля системы управления кастомизацией модулей ИТ-проекта. К примеру, избранная операция, которая рекомендована пользователю на главном окне,

может быть сконфигурирована путем выбора одной из опций в выпадающем списке, содержащим коллекцию основных пользовательских действий конкретного плагина. Сам же выпадающий список может быть пополнен при помощи кнопки «Добавить операцию».

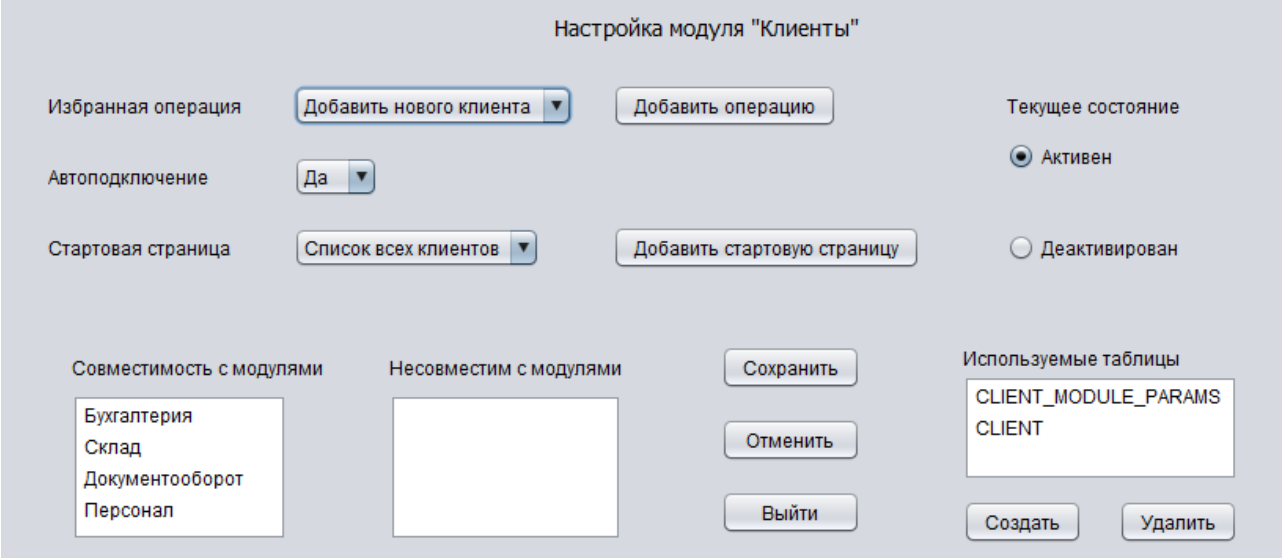

Рисунок 3.2 – Окно настройки модуля «Клиенты» системы управления кастомизацией модулей ИТ-проекта

Одной из самых главных настроек, представленных в данном окне, является возможность создания кастомизированной таблицы, если не устраивают существующие или недостаточно функционала в рекомендованных по умолчанию.

3.3 Структурно-функциональная модель системы управления кастомизацией модулей ИТ-проекта

Принято считать, что для создания расширяемого приложения с использованием плагинов нужно совсем немного: описать в classpath файлы .jar с плагинами, спроектировать интерфейс и файлы, содержащие описание этих плагинов. После старта системы, она должна обработать файл с описанием и инициализировать плагины.

В реальности же такой подход, хоть и прост в проектировании и имплементации, имеет несколько больших недостатков.

1. Плагины обязательно должны быть описаны в classpath программы, а это возможно далеко не всегда.

2. В случае, когда в classpath попадают файлы с расширением .jar, которые содержат некоторые классы, имеющие одинаковые названия, поведение системы становится непредсказуемым.

3. При таком подходе плагины имеют доступ к объектам «ядра» программы, что является нежелательным, ведь плагины должны быть изолированы.

4. Подключение нового плагина подразумевает под собой перезапуск приложения, что зачастую является недопустимым. К примеру, если рассмотреть архитектуру клиент-серверного приложения, где такие плагины выполняют роль сервисов, перезагрузка сервера ради того, чтобы добавить такой сервис является крайне неоправданной.

Во избежание перечисленных выше недостатков необходимо имплементировать такой механизм, который бы дал возможность загружать необходимые классы во время исполнения программы и ограничить их «зону видимости». Таким механизмом являются ClassLoader’ы.

Для изолирования плагинов друг от друга достаточно просто загрузить их в разных ClassLoader'ax. Для этого будет создан public интерфейс, от которого и будут наследоваться плагины.

Пример такого интерфейса показан ниже:

```
public interface PluginInt {
public void invoke();
}
```

Так как подобный интерфейс будет необходим при создании абсолютно любого плагина в будущем, то будет целесообразно собрать этот интерфейс в файл с расширением .jar и использовать как библиотеку при создании плагинов.

Для каждого модуля создадим отдельный экземпляр класса ClassLoader. Чтобы не создавать свой собственный ClassLoader, воспользуемся готовым URLClassLoader’ом, который прекрасно подходит для данной задачи.

URLClassLoader — это стандартная библиотека Java SE 7. Этот загрузчик классов используется для загрузки классов и ресурсов из пути поиска URL-адресов, относящихся как к файлам .jar, так и к каталогам. Предполагается, что любой URL, заканчивающийся символом «/», ссылается на каталог. В противном случае предполагается, что URL ссылается на файл .jar, который будет открыт при необходимости.

AccessControlContext (класс AccessControlContext используется для принятия решений о доступе к системным ресурсам на основе контекста, который он инкапсулирует) потока, который создал экземпляр URLClassLoader, будет использоваться при последующей загрузке классов и ресурсов.

Загруженные классы по умолчанию получают разрешение только на доступ к URL-адресам, указанным при создании URLClassLoader.

Для создания нового экземпляра этого класса, в его конструктор необходимо подать массив URL папок и файлов .jar и явно указать объект класса ClassLoader, от которого наш URLClassLoader будет наследоваться.

Таким образом реализуется изолирование плагинов друг от друга, так что одинаковые имена классов внутри разных плагинов больше не являются проблемой.

Пример реализации такого подхода можно увидеть в конструкторе класса, представленного в приложении А.

Для каждого плагина удобно использовать заранее подготовленные свойства, которые можно описать в файле-дескрипторе типа .properties. Эти свойства удобно загружать вместе с самим плагином и передавать в класс, осуществляющий управление модулями, в котором эти данные (имя модуля и его свойства) будут держаться в коллекции типа Map вместе с другими плагинами.

Класс для загрузки модуля и его свойств представлен в приложении А, а класс, осуществляющий управление с примером формы и кнопки описан в приложении Б.

Для динамической загрузки и выгрузки плагинов реализуются методы addPluginByName() и removePluginByName() в классе, описанном в приложении Б.

Таким образом, представленное ИТ-решение обладает всеми необходимыми возможностями для дальнейшего развития и доработки.

Выводы по главе 3

1. Была рассмотрена процедура моделирования стандартного ИТ-проекта, которая в большинстве случаев обладает рядом проблем, вызванных нехваткой различного рода ресурсов и нуждается в автоматизации.
2. Было разработано программное обеспечение системы управления кастомизацией модулей ИТ-проекта.
3. Представлены в виде программного кода и рассмотрены основные функции разрабатываемой системы управления кастомизацией модулей ИТ-проекта, которая вполне может в дальнейшем расширяться и дорабатываться.

ЗАКЛЮЧЕНИЕ

Целью данной магистерской диссертации является разработка модели системы управления кастомизацией модулей ИТ-проекта, обеспечивающей повышение эффективности управления процессом кастомизации модулей ИТ-проекта.

Внедрение системы управления в большинстве случаев положительно влияет на некоторые показатели организации, такие, как, например, экономические показатели. Это связано с автоматизацией некоторых задач или повышения эффективности и продуктивности работы лиц, ответственных за кастомизацию модулей в любом ИТ-проекте.

Решением такого типа вопросов может служить правильное внедрение системы управления кастомизацией для обеспечения оптимизации трудовых и финансовых затрат.

Осуществленные в данной магистерской диссертации научные исследования демонстрируют следующие главные результаты:

1. Были проанализированы актуальные механизмы управления процесса кастомизации модулей ИТ-проекта. Как показал анализ, для эффективного управления кастомизации модулей используются модель Entity-Attribute-Value и архитектура с использованием модулей (плагинов).

2. Проведен анализ существующих подходов к моделированию систем управления кастомизацией, который показал, что использование систем управления кастомизацией модулей ИТ-проекта связано с определенными сложностями, обусловленными плохой осуществимостью использовать одно программное обеспечение под различные организации.

Поэтому представляет актуальность разработка системы управления кастомизацией модулей ИТ-проекта, легко адаптируемой к специфике проектов конкретных заказчиков.

3. Проведен сравнительный анализ методологических подходов к моделированию разработки системы управления кастомизацией модулей ИТ-

проекта. По его итогам было принято решение использовать в качестве методологической основы моделирования системы управления объектно-структурный подход.

4. Проведено функциональное моделирование системы управления, спроектирована логическая модель данных, а также построены диаграмма прецедентов, диаграммы последовательности, диаграмма классов и блок-схемы основных алгоритмов работы для наглядного представления о моделируемой системе управления кастомизацией модулей ИТ-проекта.

Из этого следует, что в данной магистерской диссертации решена актуальная научно-исследовательская проблема разработки системы управления, которая призвана обеспечить эффективное управление кастомизацией модулей ИТ-проекта.

Важнейшим научным результатом магистерской диссертации является вывод о том, что разработанная модель и система управления кастомизацией модулей ИТ-проекта позволяет увеличить скорость кастомизации ИТ-проекта.

Таким образом, внедрение в компании системы управления кастомизацией модулей ИТ-проекта является эффективным и целесообразным.

Гипотеза исследования подтверждена.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Нормативно-правовые акты

1. ГОСТ 34.003-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Термины и определения. – Взамен ГОСТ 24.003-84, ГОСТ 22487-77; введ. 1992-01-01. – М.: Издательство стандартов, 1992. – 14 с.

2. ГОСТ 34.601-90. Информационные технологии. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания. – Введ. 1992-01-01. М.: Издательство стандартов, 1992. – 6 с. – (Основополагающие стандарты).

3. ГОСТ 34.602-89. Информационные технологии. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы. – Введ. 1990-01-01. – М.: Издательство стандартов, 1990. – 12 с. – (Основополагающие стандарты).

4. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения (ИСО 5807-85). Введ. 1992-01-01. – М.: Издательство стандартов, 1992 – 14 с. – (Единая система программной документации).

5. ГОСТ 2.105-95. Общие требования к текстовым документам. – М.: Издательство стандартов, 1996. – 29 с. – (Единая система конструкторской документации).

6. ГОСТ 34.320-96. Информационная технология. Система стандартов по базам данных. Концепции и терминология для концептуальной схемы и информационной базы. – Введ. 2001-07-01. – М.: Издательство стандартов, 2001. – 46 с. – (Основополагающие стандарты).

7. ГОСТ Р ИСО/МЭК 12207-99. Информационная технология. Процессы жизненного цикла программных средств. Введ. 2000-07-01. – М.: Издательство стандартов, 2000. – 30 с.

8. ГОСТ 7.1-2003. Библиографическая запись. Библиографическое описание документа.

9. ГОСТ 7.82-2001. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления. – Введ. 2002-07-01. – Минск: Издательство стандартов, 2001. – 35 с. – (Система стандартов по информации, библиотечному и издательскому делу).

10. Методология функционального моделирования IDEF0. Руководящий документ. Издание официальное. — М.: ИПК Издательство стандартов, 2000. — 75 с.

11. Федеральный закон РФ от 27 июля 2006 года № 152-ФЗ «О персональных данных».

Научная и методическая литература

12. Башкатова Ю. И., Решетько Н. И. Современные информационные системы как фактор повышения качества управленческих решений и конкурентоспособности организаций / Интернет-журнал Науковедение. – 2014. – №2 (21) – С.8.

13. Борисов С. А., Плеханова А. Ф. Особенности управления проектами в области информационных систем / Фундаментальные исследования. – 2014. – №9-3 – С.625-629.

14. Брумштейн Ю. М., Дюдиков И. А. Модели оптимизации подбора ресурсов при управлении совокупностью проектов с учетом зависимости качества результатов, рисков и затрат / Вестник АГТУ. Серия: Управление, вычислительная техника и информатика. – 2015. – №1 – С.78-89.

15. Варламов С. В., Скородумов П. В. Система управления проектами организации: анализ подходов и существующих программных решений / Вопросы территориального развития. – 2015. – №5 (25) – С.5.

16. Гаджиев Н. К. Применение информационно-аналитических систем предприятий в России / Фундаментальные исследования. – 2014. – №5-4 – С.816-819.

17. Жданова А.А. Применение робототехнических конструкторов в процессе обучения младших школьников / Прикладная математика и информатика: современные исследования в области естественных и технических наук: Сборник научных статей IV научно-практической международной конференции (школы-семинара) молодых ученых: 23-25 апреля 2018 г. В двух частях. – Тольятти: Издатель Качалин Александр Васильевич, 2018. – С. 106-109.

18. Жданова А.А. Современные проблемы кастомизации модулей ИТ-проекта / Международная научно-практическая конференция: Наука и образование в XXI веке (Тамбов, 31 мая 2019 г.). (принята к публикации).

19. Логинов М.Д., Логинова Т.А. Техническое обслуживание средств вычислительной техники: Учебное пособие - М.: БИНОМ. Лаборатория знаний, 2014. - 319 с.

20. Остроухов, А.В. Интеллектуальные информационные системы и технологии / А.В. Остроухов, Н.Е. Суркова. - Красноярск: Научно-инновационный центр, 2015. – 370 с.

21. Партыка Т.Л., Попов И.И. Периферийные устройства вычислительной техники: Уч. пособие. - М.: Форум: ИНФРА-М, 2014. - 432 с.

22. Певченко С. С. Методы анализа данных // Молодой ученый. — 2015. — №13. — С. 167-169.

23. Плотников А. Н., Плотников Д. А. Актуальные проблемы управления проектами / Изв. Сарат. ун-та Нов. сер. Сер. Экономика. Управление. Право. – 2014. – №1-2 – С.152-158.

24. Романов В.П. Техническое обслуживание средств вычислительной техники: Учебно-методическое пособие. - Новокузнецк: ФГОУ СПО «Кузнецкий индустриальный техникум», 2014. - 191 с.

25. Севастьянова И. Г., Стегний В. Н. Делегирование полномочий для принятия эффективных решений / Власть. – 2014. – №1 – С.55-57.

26. Степанова Ю. Н., Шипилова Е. Г. Основные принципы и этапы моделирования информационных систем управленческого учета // Молодой ученый. — 2017. — №10. — С. 273-276.

27. Терентьев Н.Ю. Стили и процесс принятия решений / Проблемы учета и финансов. – 2014. – №1 (13) – С.63-67.

Электронные ресурсы

28. Википедия – свободная энциклопедия [Электронный ресурс]. – Режим доступа: http://ru.wikipedia.org (дата обращения 21.05.2018 г.).

29. Идеальная архитектура [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/133287/ (дата обращения 22.09.2018 г.).

30. Национальный Открытый Университет "ИНТУИТ" | Бесплатное образование [Электронный ресурс]. – Режим доступа: https://www.intuit.ru/ (дата обращения 23.10.2018 г.).

31. О кастомизации информационных систем [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/haulmont/blog/214787/ (дата обращения 24.11.2018 г.).

32. Создание архитектуры программы или как проектировать табуретку [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/276593/ (дата обращения 25.12.2018 г.).

Литература на иностранном языке

33. A. Czarnecki, L. Klich, C. Orłowski. Simulation of the IT Service and Project Management Environment. Biuletyn WAT, vol. LXII, Nr 1, pp.162 – 165, 2013.

34. A.-G. Dosne, M. Bergstrand, K. Harling, M. O. Karlsson. Improving the estimation of parameter uncertainty distributions in nonlinear mixed effects models using sampling importance resampling. Journal of Pharmacokinetics and Pharmacodynamics, December 2016, Volume 43, Issue 6, pp 583–596.

35. G. Grambow, R. Oberhauser, M. Reichert. Contextual Injection of Quality Measures into Software Engineering Processes. Int'l Journal on Advances in Software, 4 (1&2), pp. 76-99, 2011.

36. Product Module Network Modeling and Evolution Analysis. Computational Intelligence and Neuroscience, vol. 2019 (8), pp.1 – 8, 2019.

37. R. Oberhauser. Leveraging Semantic Web Computing for Context-Aware Software Engineering Environments. Semantic Web, Gang Wu (Ed.), 2011, pp.157 – 180

ПРИЛОЖЕНИЕ А

КЛАСС ЗАГРУЗКИ ПЛАГИНОВ

```
import javax.swing.*;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.net.URLClassLoader;
import java.util.Enumeration;
import java.util.Properties;
import java.util.jar.JarEntry;
import java.util.jar.JarFile;

public class PluginInfo {
private Plugin instance;

private JButton associatedButton;

public PluginInfo(File jarFile) throws Exception
{
try {
Properties props = getPluginProps(jarFile); //Получение свойств конкретного плагина
if (props == null)
throw new IllegalArgumentException("No props file found");

String pluginClassName = props.getProperty("main.class"); //Загрузка свойства по имени
```

```
if (pluginClassName == null || pluginClassName.length() == 0) {
throw new Exception("Missing property main.class");
}

//Загрузка класса, представленного в плагине и создание объекта этого класса
URL jarURL = jarFile.toURI().toURL();
URLClassLoader classLoader = new URLClassLoader(new URL[]{jarURL});
Class pluginClass = classLoader.loadClass(pluginClassName);
instance = (Plugin) pluginClass.newInstance();
} catch (Exception e) {
throw new Exception(e);
}
}

public Plugin getPluginInstance()
{
return instance;
}


private Properties getPluginProps(File file) throws IOException //Загрузка файла со свойствами плагина
{
Properties result = null;
JarFile jar = new JarFile(file);
Enumeration entries = jar.entries();

while (entries.hasMoreElements()) {
```

```
JarEntry entry = (JarEntry) entries.nextElement();
if (entry.getName().equals("desc.properties")) {
// That's it! Load props
InputStream is = null;
try {
is = jar.getInputStream(entry);
result = new Properties();
result.load(is);
} finally {
if (is != null)
is.close();
}
}
}
return result;
}

public void setAssociatedButton(JButton associatedButton)
{
this.associatedButton = associatedButton;
}

public JButton getAssociatedButton()
{
return associatedButton;
}
}
```

ПРИЛОЖЕНИЕ Б

КЛАСС УПРАВЛЕНИЯ МОДУЛЯМИ

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileFilter;
import java.util.HashMap;
import java.util.Map;

public class Module {
private Map<String, PluginInfo> plugins;

private JFrame mainFrame;

//В этом методе происходит заполнение коллекции плагинами и их свойствами,
а также отрисовка формы
public void start()
{
File pluginDir = new File("plugins");
File[] jars = pluginDir.listFiles(new FileFilter() {
public boolean accept(File file) {
return file.isFile() && file.getName().endsWith(".jar");
}
});

plugins = new HashMap();
```

```
for (File file : jars) {
try {
plugins.put(file.getName(), new PluginInfo(file));
} catch (Exception e) {
e.printStackTrace();
}
}

mainFrame = new JFrame("Plugin test");
final JFrame frame = mainFrame;
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(600, 300);
frame.getContentPane().setLayout(new FlowLayout());

synchronized (plugins) {
for (PluginInfo pluginInfo : plugins.values()) {
final PluginInfo plugin = pluginInfo;
final JButton button = new JButton("button");
plugin.setAssociatedButton(button);
button.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
plugin.getPluginInstance().init(new PluginContext(){
public JButton getButton() {
return button;
}
public JFrame getFrame() {
return frame;
}
```

```
});
plugin.getPluginInstance().invoke();
}
});
frame.getContentPane().add(button);
}
}

frame.setVisible(true);
}

//Метод для отключения плагина динамически
public void removePluginByName(String jarName) throws Exception
{
if (!plugins.containsKey(jarName)) {
throw new Exception(jarName + " not loaded");
}
PluginInfo pluginInfo = plugins.get(jarName);

mainFrame.remove(pluginInfo.getAssociatedButton());
mainFrame.validate();
mainFrame.repaint();

synchronized (plugins) {
plugins.remove(jarName);
}
}

//Метод для подключения плагина динамически
```

```
public void addPluginByName(String jarName) throws Exception {
if (plugins.containsKey(jarName)) {
throw new Exception(jarName + " already loaded");
}
File jar = new File(jarName);

synchronized (plugins) {
plugins.put(jarName, new PluginInfo(jar));
PluginInfo pluginInfo = plugins.get(jarName);
JButton newjb = new JButton("button");
pluginInfo.setAssociatedButton(newjb);
mainFrame.add(newjb);
}
mainFrame.validate();
mainFrame.repaint();
}

public static void main(String[] args) {
new Module().start();
}
}
```