Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский Томский государственный университет»

На правах рукописи

Змеев Денис Олегович

СИСТЕМА ПОДДЕРЖКИ ПРИНЯТИЯ РЕШЕНИЙ ДЛЯ УПРАВЛЕНИЯ ПРОЕКТАМИ В ПРОГРАММНОЙ ИНЖЕНЕРИИ НА ОСНОВЕ СТАНДАРТА OMG ESSENCE С ПРИМЕНЕНИЕМ ДИНАМИЧЕСКИХ БАЙЕСОВСКИХ СЕТЕЙ

05.13.11 - Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация
на соискание ученой степени
кандидата технических наук

Научный руководитель
Доктор физико-математически наук,
Профессор Змеев Олег Алексеевич

Томск – 2021

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С момента возникновения программной инженерии как отдельной профессиональной и научной дисциплины в 1960-х гг. можно выделить два основных направления теоретических и прикладных исследований в этой области. С одной стороны, особенно на первоначальном этапе, это вопросы, связанные с обеспечением повышения качества программного обеспечения. Под качеством в рамках программной инженерии понимается обширный блок разнообразных проблем, связанных непосредственно с методами создания кода программных систем, эффективностью эксплуатации и обслуживания решений, стабильностью, скоростью, удобством использования, тестируемостью, удобочитаемостью, объемом, стоимостью, безопасностью и количеством недостатков или «ошибок», а также с менее измеримыми качествами, такими как элегантность, лаконичность и удовлетворенность клиента, а также многими другими атрибутами [1]. С другой стороны, в виде отдельного направления исследований выделились вопросы, связанные с процессами создания программного обеспечения. Это достаточно широкая область, которая начинается от принципов проектирования программного обеспечения, а завершается более сложными проблемами управления, такими как оптимальный размер команды, собственно методы организации процесса разработки программного обеспечения, организация взаимодействия внутри команды разработчиков и т. д. [2].

Наличие такого деления характерно для любой инженерной дисциплины и объясняется последовательным применением конструктивного мышления [3] на первом этапе непосредственно к продукту деятельности, а на втором – к процессу его создания. С другой стороны, с точки зрения технических наук программная инженерия – это молодая дисциплина в области компьютерных наук, которая находится на достаточно раннем этапе своего развития [4]. Для этого этапа характерно отсутствие общепризнанного набора научных основ, и идет очень тяжелый процесс формирования этих основ на базе обобщения накопленного опыта, успехов и неудач [5]. Более того, быстрое развитие информационных технологий накладывает определенную специфику на это стандартное развитие. На каждом этапе формирования того или иного стека технологий в рамках программной инженерии достаточно быстро удается решить задачи, характерные для непосредственного формирования кода программного продукта, при этом построение оптимального процесса разработки программного обеспечения до сих пор является актуальным и дискуссионным вопросом.

Актуальность. Начиная с 1995 г. компания «The Standish Group» публикует данные аналитического исследования «CHAOS Report», в рамках которого рассматриваются результаты

проектов по разработке программного обеспечения по всему миру. В результате этих наблюдений складывается статистика проектов, выполненных в течение двадцати лет, которая показывает, что доля удачных проектов составляет в среднем около 30 %. Таким образом, в лучшем случае проблемными, а в худшем – неудачными признаются 70 % проектов [6, 7, 8, 9]. Влияние этих цифр на экономику индустрии разработки программного обеспечения огромно: за ними стоят убытки реальных компаний, которые исчисляются в сотнях миллиардов долларов. Таким образом, тематика работ, связанная с эффективным управлением процессом разработки программного обеспечения, остается достаточно актуальной.

Состояние проблемы. С момента формирования самого термина «процесс разработки программного обеспечения»[1], который достаточно часто называют также «жизненным циклом программного обеспечения»[2] можно выделить следующие основные направления научных исследований в этой области.

Во-первых, применение различного рода формальных математических методов для создания моделей процесса разработки или проектного управления (т. к. во многих аспектах процессы разработки направлены именно на проектную разработку программного обеспечения) с целью зафиксировать статическую и / или динамическую составляющую процесса и на основе модели проанализировать количественные и качественные характеристики, чтобы, в свою очередь, на их основе получить правила или алгоритмы улучшения различных аспектов управления проектами в программной инженерии. В целом работы в этой области можно разделить на несколько основных направлений.

1. Методы, основанные на формальных математических моделях проекта, используя структуру которых можно формировать аналитические решения. Наиболее часто такими моделями выступают сетевые графики проекта или его аналогичные формы (направленный ациклический граф, диаграмма Ганта, календарный план-график работ). Подобные методы предполагают наличие сетевого графика проекта (или его аналог), созданного для планирования и распределения всех работ по проекту. Работы, относящиеся к этой отрасли, достаточно активно используют строгую зафиксированную последовательность задач или действий, которые необходимо реализовать, а также все дополнительные ограничения (временные рамки, требуемые ресурсы и навыки исполнителей), чтобы создать более оптимальные планы [10, 11, 12, 13, 14] В частности, как подкласс используемых методов активно развиваются генетические алгоритмы, причем до такой степени, что начинают появляться работы, которые анализируют уже различные нюансы и частные

[1] Software development process (англ.)
[2] SDLC – Software development life cycle.

случаи эффективности в опубликованных работах [15]. Необходимо отметить, что исследования, которые пытаются построить более строгие с точки зрения алгебры модели программного обеспечения и проектов по их разработке, ведутся [16], но не получают активного развития в области прикладного применения.

2. Методы, применимость которых основана на выборках или данных, доступных для исследователя / аналитика. Один из наиболее распространенных методов – прикладная математическая статистика, работы по которой условно можно разделить на использующие методы, применимые к управлению проектами в общем [17], в рамках которых в последнее время активно развиваются методы, основанные на EVM[3] [18, 19, 20, 21], а также работы, выполненные с учетом особенностей программной инженерии [22, 23, 24]. К этой же категории методов относятся и алгоритмы машинного обучения [25] или приближенных вычислений, которые обычно фокусируются на отдельных задачах программной инженерии [26, 27].

Во-вторых, следует выделить работы, посвященные вопросам управления отдельными дисциплинами процесса разработки программного обеспечения. Поскольку исторически программная инженерия – дисциплина, созданная в первую очередь профессиональным сообществом, существует тенденция, что профессиональное сообщество создает различные новые практики, методы и подходы для все более качественного выполнения задач, а академическое сообщество начинает анализировать и сравнивать полученные результаты с целью выявления закономерностей, возможных путей улучшения, создания более формальных альтернатив. В качестве примера можно привести такие отрасли программной инженерии, как стандарты описания требований [28, 29, 30, 31], статические анализаторы исходного кода приложений [32, 33, 34, 35], обнаружение дефектов программного обеспечения [36, 37], процессы CI/CD[4], связанные с развертыванием приложений в средах разработки и эксплуатации [38, 39, 40].

В-третьих, прикладные исследования различных стандартов процессов разработки и их практического использования. Основное направление работ здесь связано с онтологическим анализом процессов разработки, который используется для высокоуровневого семантического описания основных сущностей процессов разработки, а также для определения связей между ними. Основные компоненты онтологии – словарь терминов и схема, описывающая связи между этими терминами. Одной из дополнительных возможностей такого способа описания процессов разработки является подход к сравнению различных процессов разработки, основанный на алгебре сопоставления терминов и на общих схемах в онтологической нотации [41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52].

[3] Earned Value Management – управление заработанной ценностью.
[4] Continuous Integration / Continuous Delivery

Отдельного внимания в онтологических подходах заслуживает Project Management Body of Knowledge[5] [42, 53, 54, 55, 56, 57, 58, 59, 60], который используют в либо в качестве основы онтологической модели, либо как один из сравниваемых при помощи онтологии стандартов.

Кроме этого, существует класс работ, посвященных взаимодействию систем для управления проектами (PM) и бизнес-процессов в организациях [42, 46, 57, 59, 61]. В этих работах авторы изучают проблемную ситуацию, связанную с тем, что процессы разработки рассматривают идеальные сценарии – есть одна команда, один проект, и все заняты только им. А в реальности организации, которые занимаются проектами по разработке программного обеспечения, одновременно заняты несколькими проектами и разработчики могут выполнять работы по всем проектам одновременно. Таким образом получается, что для разработчиков нет именно проектов – для них это бизнес-процесс, и для организации, которая занимается заказной проектной разработкой, это тоже процессная деятельность, а не проектная. В рамках описанной ситуации менеджер сталкивается с проблемой, как интегрировать проектные задачи, которые должны начать выполняться, как только они были поставлены, с процессами, в которых для разработчика задачи идут потоком с разных проектов.

Тем не менее до недавнего времени достаточно остро обсуждалась проблема «сравнимости» различных процессов разработки и методологий управления проектами. С одной стороны, даже наличие общепризнанного стандарта PMBOK не снимало вопросов, связанных со спецификой проектов в области программной инженерии. С другой стороны, различные процессы разработки не всегда однозначно сопоставляются друг с другом, причем в некоторых случаях даже разные версии процессов разработки в рамках одной и той же методологии получаются не сравнимыми друг с другом на уровне используемых сущностей, правил и принципов [62], не говоря уже о сопоставлении с общей теорией управления проектами.

С этой точки зрения отдельного внимания заслуживает инициатива SEMAT[6], которая была сформирована в 2009 г., чьей основной амбицией стало «воссоздание программной инженерии как дисциплины с четкой методологией». Основная идея нового подхода заключается не в поиске и фиксации новых универсальных процессов разработки, а в формализации устоявшихся способов организации тех или иных специализированных работ в рамках процесса разработки программного продукта, которые приводили бы команду проекта к относительно стабильному результату. Такие найденные, формализованные и проверенные в рамках действующих проектных команд способы организации специализированных работ получили название «практики». Определенная новизна,

[5] В текстах очень часто используется аббревиатура PMBOK.
[6] SEMAT – Software Engineering Method and Theory.

обнаруженная авторами инициативы, заключалась в том, что в абсолютном большинстве случаев команды или компании по разработке программного обеспечения не выбирали какой-либо процесс разработки полностью и явно ему следовали, а фактически брали для процесса те практики, которые их устраивали, и применяли их к своей деятельности, а если им не хватало текущего набора практик, то либо формировали собственные, либо брали их из других процессов разработки. Таким образом практически каждая команда постепенно формирует собственный подход к тому, как она занимается разработкой программного обеспечения. Комплексные подходы получили название «метод» в терминах, вводимых SEMAT.

В процессе обсуждения инициативы [63, 64] была поставлена цель разработать способ описания существующих и потенциальных практик организации работ в области программной инженерии и наиболее распространенных вариантов их компоновки – стандартных методов. В результате проделанной работы в 2013 г. появилось определение базового языка описания практик современной программной инженерии – языка Essence [65]. В форме этого языка инициатива SEMAT фактически получила одновременно и онтологическую модель различных процессов разработок, и графический язык для их описания.

Практическую значимость данного языка подтверждает тот факт, что в течение одного года он становится новым стандартом OMG[7] [66]. Появление нового стандарта вызвало обширную дискуссию в мире программной инженерии, в ходе которой обсуждались как основные понятия нового подхода – альфы, состояния, активности и т. д. [67, 68], так и его потенциальные возможности [69, 70, 71, 72, 73]. Тем не менее на текущий момент очень сложно оценить влияние нового стандарта на дисциплину программной инженерии. С одной стороны, в момент его подготовки многие известные исследователи и практики, исповедующие так называемые «гибкие методологии», высказывались об инициативе SEMAT достаточно критически [74, 75, 76]. С другой стороны, появилось достаточно много работ, показывающих применение стандарта [66] для решения различных задач в области программной инженерии.

Например, И. Якобсоном в идеологии Essence представлена практика применения вариантов использования [72]. В работе «Scrum Essential Cards» показано представление в рамках нового стандарта одной из самых известных Agile-методологий SCRUM [73]. В рамках образовательного трека инициативы SEMAT предпринимаются попытки разработки специальных курсов, которые будут знакомить будущих программных инженеров с новым стандартом [77]. Появляются работы,

[7] OMG – Object Management Group.

в рамках которых стандарт используется для решения практических задач [78]. Разрабатываются программные инструменты, использующие нотацию стандарта [79].

Учитывая специфику программной инженерии, распространение нового подхода должно сопровождаться разработкой средств визуального моделирования, позволяющих создавать модели и применять на практике полученные результаты [80, 81, 82, 83]. В настоящий момент наибольшей полнотой функциональности для построения моделей в рамках стандарта SEMAT обладают два CASE-средства от компании «Ivar Jacobson International»: «Practice Workbench» [83] – приложение, которое позволяет строить визуальные модели практик и методов, и «Essence Enterprise 365» – опубликованная библиотека практик и методов, реализованная в виде классического веб-приложения [80]. При этом необходимо отметить, что несмотря на то, что Essence и case-средства для его использования направлены на формальное описание практик и методов, к сожалению, существующие инструменты не решают достаточно важную задачу переноса их в реальную среду управления проектами, что ограничивает применение нового стандарта в существующих компаниях, занимающихся разработкой программного обеспечения. На практике даже при наличии стандартизированного и строгого описания процесса разработки вопрос его корректного переноса в среду управления проектами по-прежнему является субъективной задачей команды.

Таким образом, необходимо отметить, что на текущий день не существует подхода или модели решения, которая одновременно позволяла бы учитывать специфику процессов разработки при управлении проектами по разработке программного обеспечения и реализовывала бы прикладные математические и алгоритмические подходы для оптимизации управленческой деятельности, а поскольку программная инженерия относится к области со слабо формализуемой экспертной деятельностью, то вариантом подобного подхода может оказаться система поддержки принятия решений, которая в отличие от классических интеллектуальных систем не заменяет человеческую деятельность, а позволяет менеджеру проекта по разработке программного обеспечения учитывать, находить и демонстрировать дополнительную значимую информацию. Для эффективного практического применения подобная система также должна:

– становиться частью или интегрироваться со средой управления проектами;

– с одной стороны позволять учитывать специфику проектной деятельности конкретной команды, но с другой – не навязывать процессов разработки или правил проектной деятельности, поддержка которых была бы экономически невыгодна;

– иметь возможность перестраиваться в зависимости от используемых проектной командой практик;

– реализовывать прикладные математические методы, которые улучшали бы процесс управления проектом.

Создание подобной системы поддержки принятия решений является масштабной и комплексной научно-технической задачей. Поскольку, с одной стороны, необходимо определить процесс формальной трансляции процесса разработки и связанных с ним ограничений на организацию проектной деятельности в среде управления проектами, с другой стороны, эта трансляция должна привести к некой модели, в основе которой лежит транслируемый процесс разработки и для которой применимы прикладные алгоритмические или математические методы, обеспечивающие интеллектуальную поддержку в процессе управления проектами по разработке программного обеспечения.

Учитывая, что в совокупности число комбинаций процессов разработки и различных прикладных алгоритмических или математических очень большая, то полноценная реализация даже существенной части из из них в форме модулей системы поддержки принятия решений крайне трудозатратна, а также тот факт, что многие прикладные методы требуют заранее подготовленной и размеченной выборки данных (которых по данному подходу просто еще не существует), было принято решение начать с того, чтобы разработать эволюционный прототип, который с одной стороны доказывал бы принципиальную реализуемость подобной системы поддержки принятия решений в управлении проектами по разработки программных обеспечений, а с другой стороны мог бы обеспечить дальнейшее развитие функциональных возможностей. Более детально требования к каждой части системы поддержки принятия решений обсуждаются в главах представленной работы.

Цель и задачи исследования. Целью данной работы является разработка математического и программного обеспечения для эволюционного прототипа системы поддержки принятия решений в управлении проектами в области программной инженерии. Для достижения цели были поставлены следующие задачи:

1) анализ специфики языка OMG Essence как средства формального описания процессов разработки;

2) разработка процедуры трансляции описанного на формальном языке процесса разработки программного обеспечения в модель организации проектной деятельности в средах управления проектами;

3) поиск прикладной математической модели для формальной постановки задач поддержки в управлении проектами по разработке программного обеспечения;

4) проверка гипотезы о применимости найденной математической модели для решения прикладной задачи управлении проектами;

5) реализация эволюционного прототипа в форме расширения к существующей среде управления проектами.

Научная новизна результатов:

1) впервые предложена промежуточная модель представления процесса разработки программного обеспечения, формализованного средствами стандарта OMG Essence, для практического использования в процессе управления проектами в программной инженерии, которая позволяет реализовать алгоритм трансляции описанного в терминах Essence процесса разработки в среды управления проектами.

2) на основе предложенной промежуточной модели реализована процедура переноса практик и методов Essence в программные средства для управления проектами.

3) построена оригинальная семантическая модель зависимостей между различными элементами ядра стандарта OMG Essence, которая позволяет более точно учитывать отношения между различными элементами ядра OMG Essence.

4) впервые предложена математическая модель для решения прикладных задач в управлении проектами по разработке программного обеспечения на основе стандарта OMG Essence в виде динамической байесовской сети, которая даёт возможность аналитические методы теории байесовских сетей.

5) на основе предложенной математической модели решена задача поиска ложноположительных ошибок менеджмента проекта в рамках теоретического контура методов Essence.

Положения, выносимые на защиту

1. Промежуточная модель представления процесса разработки программного обеспечения, формализованного средствами стандарта OMG Essence, для практического использования в процессе управления проектами в программной инженерии, которая позволяет реализовать алгоритм трансляции описанного в терминах Essence процесса разработки в среды управления проектами.

2. Процедура переноса практик и методов Essence в программные средства для управления проектами.

3. Семантическая модель зависимостей между различными элементами ядра стандарта OMG Essence, которая позволяет более точно учитывать отношения между различными элементами ядра OMG Essence;

4. Математическая модель для решения прикладных задач в управлении проектами по разработке программного обеспечения на основе стандарта OMG Essence в виде динамической байесовской сети, которая даёт возможность аналитические методы теории байесовских сетей.

5. Решение задачи поиска ложноположительных ошибок менеджмента проекта в рамках теоретического контура методов Essence.

Методы исследования. Для решения поставленных задач использованы методы прикладного системного анализа, математического моделирования, теории вероятностей и математической статистики, объектно-ориентированного анализа и проектирования.

Теоретическая и практическая значимость диссертационной работы

Разработанный эволюционный прототип и процедура трансляции процессов разработки в среды управления проектами позволяют сконцентрировать потенциальные точки расширения и дальнейшего развития системы поддержки принятия решений, и в дальнейшем развивать прикладные методы в управлении проектами, опираясь на архитектурные механизмы эволюционного прототипа. Со стороны теоретической значимости полученную байесовскую сеть можно использовать и для других задач в управлении проектами, а также на ее основе применять другие классы методов, такие как имитационные модели, статистические подходы, модели, основанные на алгоритмах машинного обучения.

Со стороны практической значимости полученные результаты позволяют:

1) оптимизировать ресурсные затраты на конфигурирование среды управления проектами при использовании новых методов и практик, за счёт автоматической трансляции процессов разработки в среды управления проектами;

2) в процессе обучения менеджеров управления проектами по разработке программного обеспечения, наглядно показывать риски, связанные с ложнопозитивными ошибками;

3) в дальнейшем развивать вопросы удобства использования разработанного расширения для практического применения в деятельности IT компаний.

Достоверность и обоснованность полученных результатов подтверждается корректным использованием методов прикладного системного анализа, математического моделирования, прикладной теории вероятностей и математической статистики, объектно-ориентированного анализа и проектирования, а также непротиворечивостью полученных результатов с точки зрения управления проектами.

Личное участие автора в получении результатов

Общая концепция исследования принадлежит научному руководителю, выбор методов, их применение и решение связанных с ними задач – личный вклад автора. Несмотря на то, что большинство достигнутых результатов опубликованы в соавторстве, фактически за привлекаемыми соавторами стоит техническая реализация программных артефактов, а также помощь в обнаружении проблем, связанных с отдельными деталями разработанных программной и математической моделей. Исключение – публикации по § 3.1 в соавторстве с д.ф.-м.н. Дмитриевым Ю. Г., который

выполнял роль научного руководителя этой составляющей исследования, а автор диссертации – исполнителя.

Связь работы с крупными научными проектами

Некоторые положения диссертации обсуждались в рамках заседаний рабочей группы инициативы SEMAT.

Соответствие паспорту специальности. Диссертационная работа соответствует области исследования специальности 05.13.11 – «Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей» по п. 1 «Модели, методы и алгоритмы проектирования и анализа программ и программных систем, их эквивалентных преобразований, верификации и тестирования» (пп. 1, 2 научной новизны), п. 3 «Модели, методы, алгоритмы, языки и программные инструменты для организации взаимодействия программ и программных систем» (пп. 3–5 научной новизны), п. 10 «Оценка качества, стандартизация и сопровождение программных систем» (пп. 1, 2 научной новизны).

Апробация работы. Основные положения работы и отдельные ее вопросы докладывались и обсуждались на следующих конференциях и семинарах международного и всероссийского уровня: Информационные технологии и математическое моделирование (ИТММ–2011), 25–26 ноября 2011 г., г. Томск; Информационные технологии. МНСК–2018, 22–27 апреля 2018 г., г. Новосибирск; Математическое и программное обеспечение информационных, технических и экономических систем, 24–26 мая 2018 г., г. Томск; Eighth International Conference on Risk Analysis and Design of Experiments, 23–26 April 2019, Vienna; Информационные технологии и математическое моделирование (ИТММ–2019), 26−30 июня 2019 г., г. Томск; Actual Problems of Systems and Software Engineering (APSSE 2019), 12–14 November 2019, Moscow; Математическое и программное обеспечение информационных, технических и экономических систем, 28–30 мая 2020 г., г. Томск; 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T), 9–12 November 2020, Munich; Информационные технологии и математическое моделирование (ИТММ–2020), 2−5 декабря 2020 г., г. Томск.

Публикации. По тематике диссертации опубликовано 11 работ, из них две статьи в журналах, включенных в Перечень рецензируемых научных изданий, в которых должны быть опубликованы основные научные результаты диссертаций на соискание ученой степени кандидата наук, на соискание ученой степени доктора наук; две статьи в изданиях, индексируемых Web of Science и Scopus; одна статья в сборнике научных трудов и шесть публикаций в сборниках материалов международных и всероссийских научных и научно-практических конференций (из них одна в сборник материалов зарубежной конференции).

Структура работы. Диссертация состоит из введения, трех глав, заключения, списка литературы, приложений. Общий объем работы составляет 120 страниц, иллюстративный материал представлен 63 рисунками и 2 таблицами, список литературы содержит 115 наименований.

Благодарности. Автор выражает глубокую благодарность:

– профессору Ивару Якобсону[8] и Полу И Макмэхону[9] за включение в состав рабочей группы инициативы SEMAT, а также за возможность консультации по некоторым вопросам, связанным с Essence;

– Андрею Даниленко и Даниилу Тамазлыкарю за помощь в технической реализации расширений к средам управления проектами;

– Данилу Соколову, Акиму Глушкову, Лидии Ивановой, Алексею Цыганкову, Виктору Скибе за моральную поддержку в течение всего исследования и предоставление взгляда со стороны ведущих разработчиков на семантические связи между утверждениями Essence Alpha Cards;

– компании «Bitworks» в лице Ивана Кудрявцева и компании «DevGuild» в лице Михаила Неверова за предоставленные данные;

– доктору физ-мат. наук Юрию Глебовичу Дмитриеву за консультации и помощь в применении статистических методов;

– компании «Ivar Jacobson International» за предоставленную лицензию к приложению «Practice Workbench».

[8] Ivar Jacobson
[9] Paul E McMahon

1. ОСНОВНЫЕ ЭЛЕМЕНТЫ СТАНДАРТА OMG ESSENCE

В рамках настоящей работы активно используются логика и модель, разработанные в рамках стандарта OMG Essence – Kernel and Language for Software Engineering Methods [66]. Разработанный на основе формальной модели стандарта теоретический контур ядра Essence обеспечивает связь между теоретическими и практическими результатами диссертации. Стандарт вводит следующий набор принципиально новых элементов для описания и использования процессов разработки программного обеспечения.

Во-первых, графическую нотацию, которая позволяет строить полные модели, фиксирующие практики программной инженерии. Эта нотация определяет язык стандарта.

Во-вторых, авторами стандарта предложена высокоуровневая модель (ядро), фиксирующая общие концепции, применимые для любого метода разработки программного обеспечения. С одной стороны, именно наличие этой модели позволяет говорить об универсальном теоретическом контуре описания методов программной инженерии. С другой – предложенный подход не ограничивается только теоретическим контуром: в рамках ядра строится своеобразный каркас, на практике обеспечивающий прохождение любого метода разработки в соответствии с предложенной теорией.

В-третьих, стандарт предусматривает новый подход к формированию методов программной инженерии как конструктора из существующих практик.

Таким образом, обеспечивается бесконечное количество возможных методов разработки – достаточно часто уникальных для каждой компании разработчиков – и фиксация устоявшихся образцов (паттернов) организации проектов по разработке программного обеспечения (практик).

Подход, предложенный в рамках стандарта, схематично представлен на рисунке 1. Учитывая значимость перечисленных выше элементов стандарта для дальнейшего понимания представленной работы, в рамках настоящей главы кратко описаны элементы стандарта и специфика их применения в контексте данной работы.

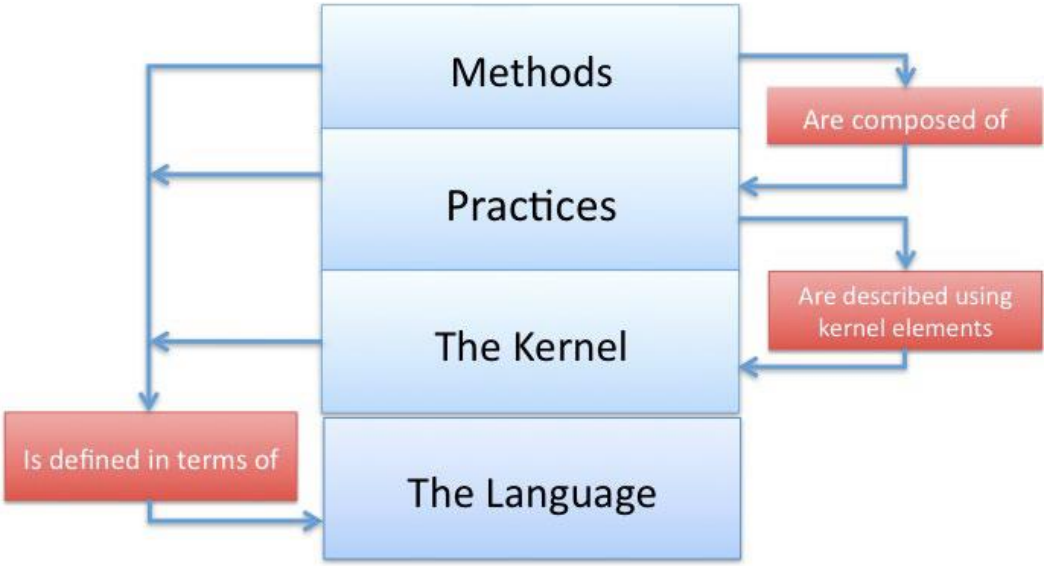


Рисунок 1 – Архитектура метода [66]

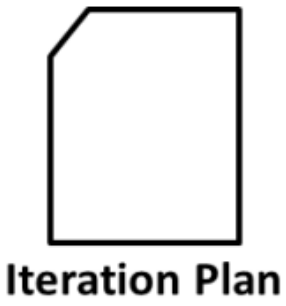
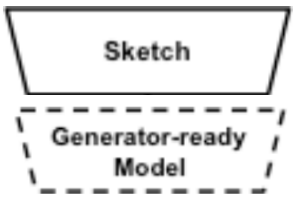
Описание основной графической нотации элементов Essence представлено в § 1.1. Более подробный разбор содержания этих элементов представлен в § 1.2. Особенности описания практик Essence разобраны в § 1.3.

Ну 1.1. Графическая нотация стандарта OMG Essence

Essence имеет двоякую природу: с одной стороны, он представлен как достаточно простой и наглядный язык графического представления практик процессов разработки. С другой стороны, на все концептуальные элементы, описывающие Essence в визуальной модели, вводится описание OMG, которое ограничивает весь стандарт строгими правилами и формальной нотацией моделирования Meta-Object Facility (MOF) [84].

В рамках графической нотации в стандарте вводится небольшой набор элементов. При этом необходимо отметить, что на практике этого набора элементов оказывается вполне достаточно для решения задач фиксации методов и практик программной инженерии. Графическая нотация стандарта, которая активно используется в рамках данной работы, с краткими определениями каждого элемента представлена в таблице 1.

Таблица 1 – Элементы графической нотации Essence

Пиктограмма	Название	Определение
	ALPHa (Альфа)	Изначально – акроним от английского Abstract Level Progress Health: семантически фиксирует некоторую бизнес-область метода или практики, состояние которой необходимо отслеживать в процессе реализации проекта
	Alpha State (Состояние Альфы)	Сущность, фиксирующая прогресс изменений ALPH (альфы) в ходе реализации метода или практики. Состоит из набора истинных / ложных утверждений, называемых Checkbox
	Activity Space (Пространство Активностей)	На высоком абстрактном уровне описывает виды деятельности, необходимые в процессе реализации метода или практики (с точки зрения ООП в некотором смысле являются абстрактными классами или интерфейсами)
	Activity (Активность)	Моделирует детальные и конкретные действия, которые необходимо совершить в процессе выполнения проекта, чтобы добиться прогресса, предписываемого практикой (с точки зрения ООП в некотором смысле являются аналогами объектов или классов, реализующих интерфейсы)
	Work Products (Рабочий Продукт)	Моделирует сущности, необходимые для описания результатов, которые создаются в ходе выполнения активностей или задач в тех или иных практиках
	Level of detail (Уровень детализации)	Сущность, фиксирующая прогресс, достигнутый рабочим продуктом. Обычно выделяются два подвида: обязательный – со сплошной рамкой – и необязательный – с пунктирной рамкой. Фактически выполняют роль, аналогичную состояниям Alpha

Кроме представленной выше графической нотации, стандарт определяет структуру данных, которая формально фиксирует все связи и сущности, необходимые для описания практик и методов Essence. На рисунке 2 представлен пример, описывающий часть структуры данных, необходимой

для описания Alpha[10], Alpha State и Work Product. На момент написания данной работы существующие программные инструменты, поддерживающие Essence и позволяющие описывать новые практики, имеют структуру данных, схожую с представленной в стандарте, но используют при этом более привычные для создания программных систем инструменты. Например, в Practices Workbench [85] для представления структур данных используется формат XML [87].

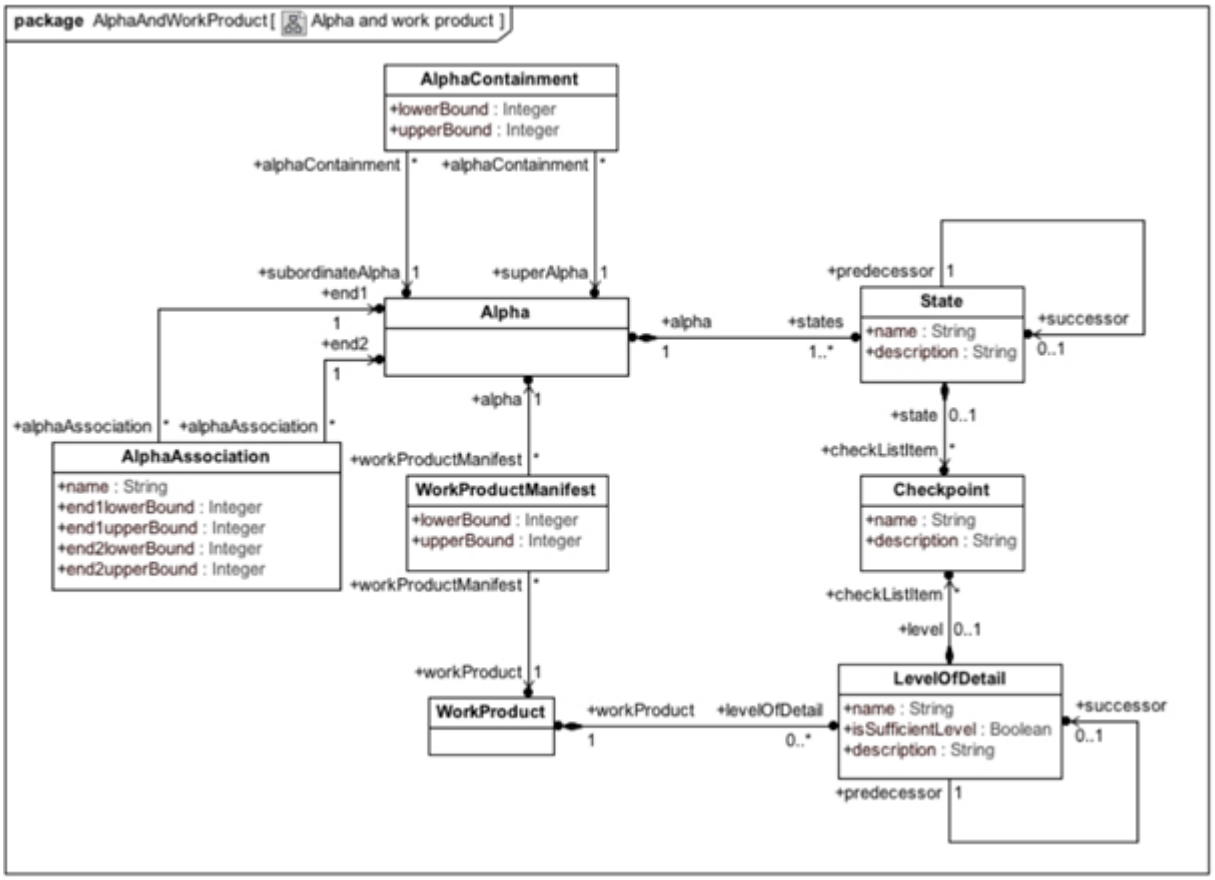


Рисунок 2 – Пример из стандарта MOF для фиксации структуры данных Alpha, Alpha State, Work product и Level of Detail [66]

При всех достоинствах формальной нотации формат MOF очень сложен для применения в решении практических задач, не связанных непосредственно со строгим описанием новых графических языков. Даже вводимые в стандарте концептуальные диаграммы Essence выполнены в демонстрационном, а не в формальном стиле взаимодействия сущностей MOF. Также важно отметить, что стандарт Essence сам по себе не связан напрямую с принятыми в профессиональной деятельности сущностями, даже такими общими, как проект, задачи, исполнители и т. п. Эти связи можно вычленить, только читая семантическую часть описания вводимых сущностей. Из представленных в стандарте описаний языка не следует, как именно применяются практики,

[10] В работе термины *ALPH, ALPHa, Альфа* и *альфа* используются как разнозначные по смыслу.

описанные в терминах стандарта. Фактически стандарт вводит язык и некоторые общие правила взаимодействия между элементами языка, но при этом явно не решает вопросы применимости описанных им же элементов. В результате для прикладного применения языка Essence необходимо проанализировать не только стандарт и вводимые им правила, а также детально изучить существующие и опубликованные авторами Essence примеры описания процессов разработки (методов разработки, согласно языку Essence) и обратить внимание именно на особенности использования языка Essence, которые негласно вводятся языком и его авторами.

1.2. Модель Kernel OMG Essence

Рассмотрим более подробно универсальную теоретическую модель, которая предлагается в стандарте в виде Kernel OMG Essence. Естественно, что в качестве начальных элементов этой модели необходимо рассмотреть набор ALPH, который введен в качестве основных элементов. При этом следует учитывать тот факт, что сущность ALPH[11] введена в язык Essence как способ отслеживания и проверки статуса некоторого формального процесса, фактически позволяя измерять прогресс процесса не в количестве написанных строк кода или в виде реализованных функциональных возможностей системы, а определяя, есть ли у команды проекта уверенность по некоторым критическим вопросам, способным повлиять на общий успех проекта.

В Kernel Essence определен набор из семи ALPH, которые в рамках стандарта считаются основными для любого проекта по разработке программного обеспечения. Этот набор ALPH разделен на три зоны, необходимость которых достаточно очевидна в рамках контекста, связанного с процессами разработки программного обеспечения. Распределение основных ALPH по тематическим зонам и связи между ними приведены на рисунке 3.

[11] Abstract Level Progress Health – абстрактный уровень здоровья прогресса.

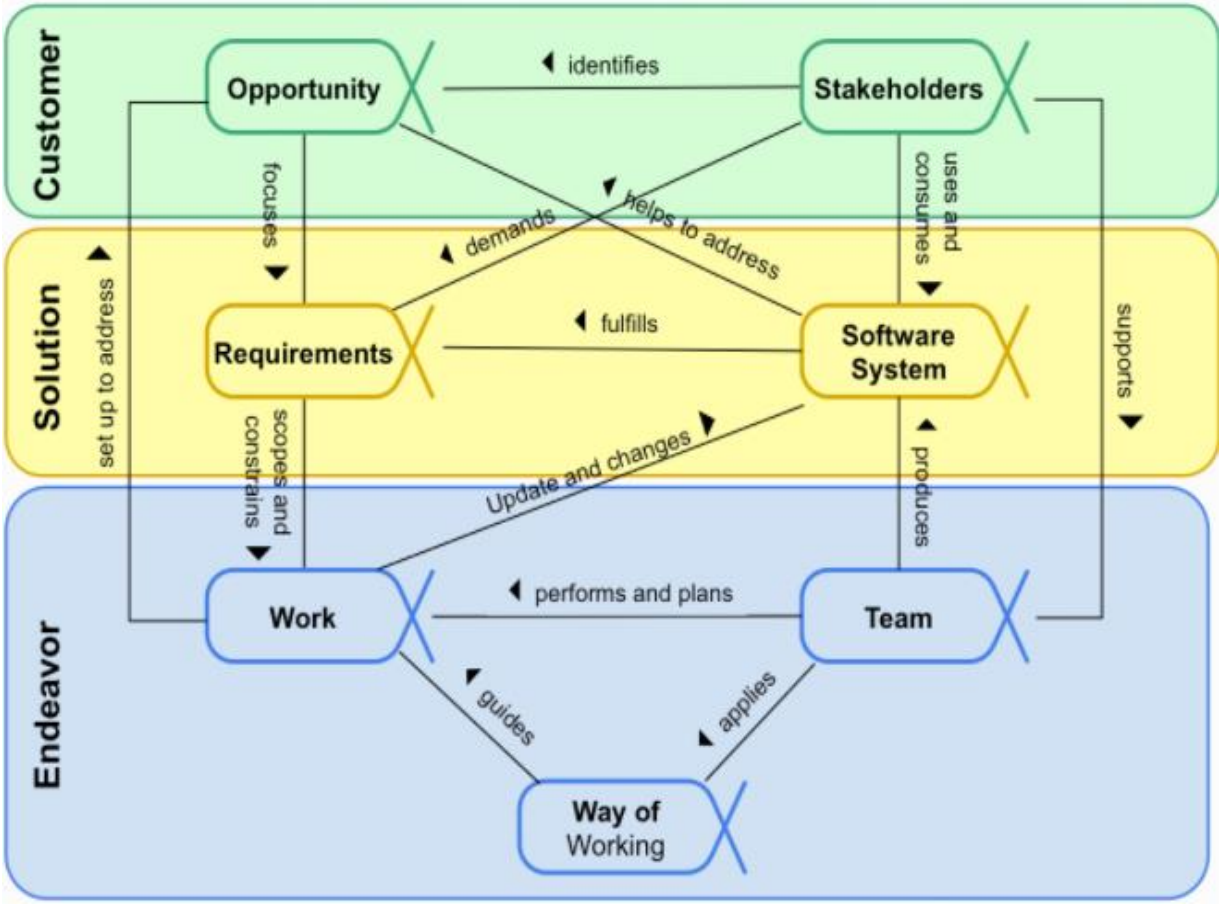


Рисунок 3 – Группировка основных ALPH и связей между ними

Далее зафиксируем семантический смысл тематических зон.

1. Зона Customer (Клиент) содержит альфы, которые отвечают за взаимодействие с клиентами и стейкхолдерами. В эту зону входят альфы Stakeholders (Стейкхолдеры) и Opportunity (Возможности). Семантика этой зоны напрямую связана с таким уровнем управления проектами, как понимание бизнес-требований. Фактически множество состояний альф зоны Customer фиксируют прогресс в понимании **кто** (альфа Стейкхолдеры) так или иначе финансирует разработку и **за что** (альфа Возможности) команда проекта получает это финансирование.

2. Зона Solution (Решение) содержит альфы, множество состояний которых фиксирует непосредственный прогресс процесса разработки целевой программной системы. Непосредственно в эту зону входят альфа Requirements (Требования) и альфа Software System (Программная система). Семантика этой зоны – непосредственно разрабатываемая программная система и требования, которые ее описывают. Как отмечают сами авторы Essence, обычно с этой зоной и с пониманием ее содержания у большинства проектных команд не возникает никаких проблем, т. к. основные

компетенции программных инженеров обычно связаны с задачами, протекающими в зоне разработки программной системы согласно зафиксированным требованиям.

3. Зона Endeavor (Поведение), к которой относятся альфы, фиксирующие: 1) то, как проектная команда организована – альфа Team (Команда); 2) насколько определены и стабильны принципы работы команды – альфа Way of Working (Способ работы); и 3) как оценивается прогресс текущего проекта – альфа Work (Работа). Хочется заметить, что именно это зона отвечает за то, что в обычной практике называется организацией и управлением процессом разработки. С этой точки зрения альфы в рассматриваемой зоне в большей степени фиксируют не уровень здоровья самого проекта, а уровень зрелости используемого процесса и уровень развития команды. Далее возможны два типичных варианта развития событий. С одной стороны, компания или команда может иметь устоявшийся метод разработки программного обеспечения. В этом случае набор состояний альф Team и Way of Working зоны Endeavor фиксирует процесс развертывания этого метода в контексте реального проекта. С другой стороны, если у команды нет устоявшегося метода, либо сильно изменился состав участников, эти альфы фиксируют известный факт, что разные команды разработчиков выполнят один и тот же проект по-разному. В этом смысле исполнитель проекта или инженер программной системы становится очень существенной и критической частью проекта, а следовательно, и программной системы, т. е. стираются границы между объектом разработки и субъектом, ее осуществляющим.

Согласно стандарту, любая ALPHa имеет набор уникальных для себя состояний, через которые она последовательно проходит в процессе реализации проекта. С этой точки зрения альфы Kernel OMG Essence, с одной стороны, на уровне структурной модели определяют инвариантное множество состояний, которое применимо для любого проекта по разработке программных систем, а, с другой стороны, на динамическом уровне фиксируют текущий прогресс, достигнутый командой разработчиков. На рисунке 4 приведено множество состояний, общих для альф Kernel OMG Essence.

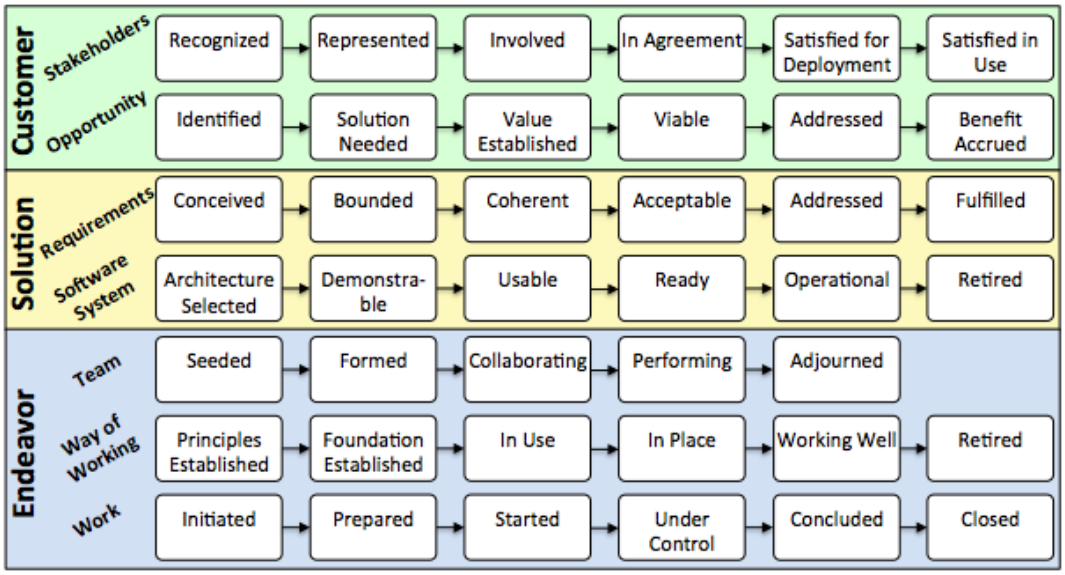


Рисунок 4 – Множество состояний альф Kernel OMG Essence

Таким образом, Alpha State (Состояние Альфы) – это характеристика, позволяющая измерять прогресс реализации ALPHa в процессе исполнения проекта. Факт достижения альфой определенного состояния из набора Alpha State обеспечивается множеством бинарных утверждений Checkbox (например, для ALPHa «программная система» есть состояние «выбрана архитектура», одно из утверждений которого «технологии выбраны»). Общее правило достигнутого состояния альфы заключается в том, что если все утверждения состояния альфы истинны в данном проекте, то считается, что альфа достигла этого состояния. Фактически изменение достигнутого прогресса проекта происходит за счет того, что проектная команда (или ее менеджмент) принимает решения об истинности или ложности тех или иных утверждений, представленных на карточках Alpha State, пример которых приведен на рисунке 5.

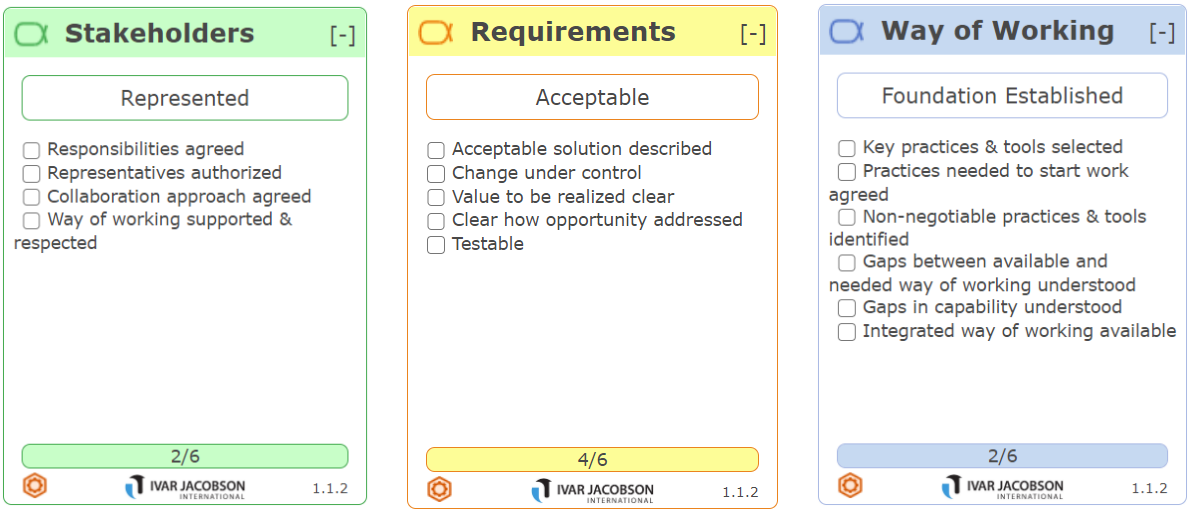


Рисунок 5 – Пример карточки Alpha State

Утверждается, что изменение состояний Альф есть линейная последовательность, также последовательна и семантическая логика представленных утверждений: в начале проекта активными утверждениями (т. е. теми, над закрытием которых трудится команда) являются утверждения начальных состояний альф, в конце проекта основными утверждениями являются утверждения конечных состояний. При этом необходимо отметить, что в языке Essence нет явного запрета возможности команды работать над утверждениями из нескольких состояний Alpha одновременно. И Alpha State, и альфы являются способами оценивать прогресс проекта, однако любая проектная деятельность сосредоточена вокруг работы (создания и модификации) различных проектных артефактов, которые в Essence получили термин Work Product, пример которого приведен на рисунке 6.

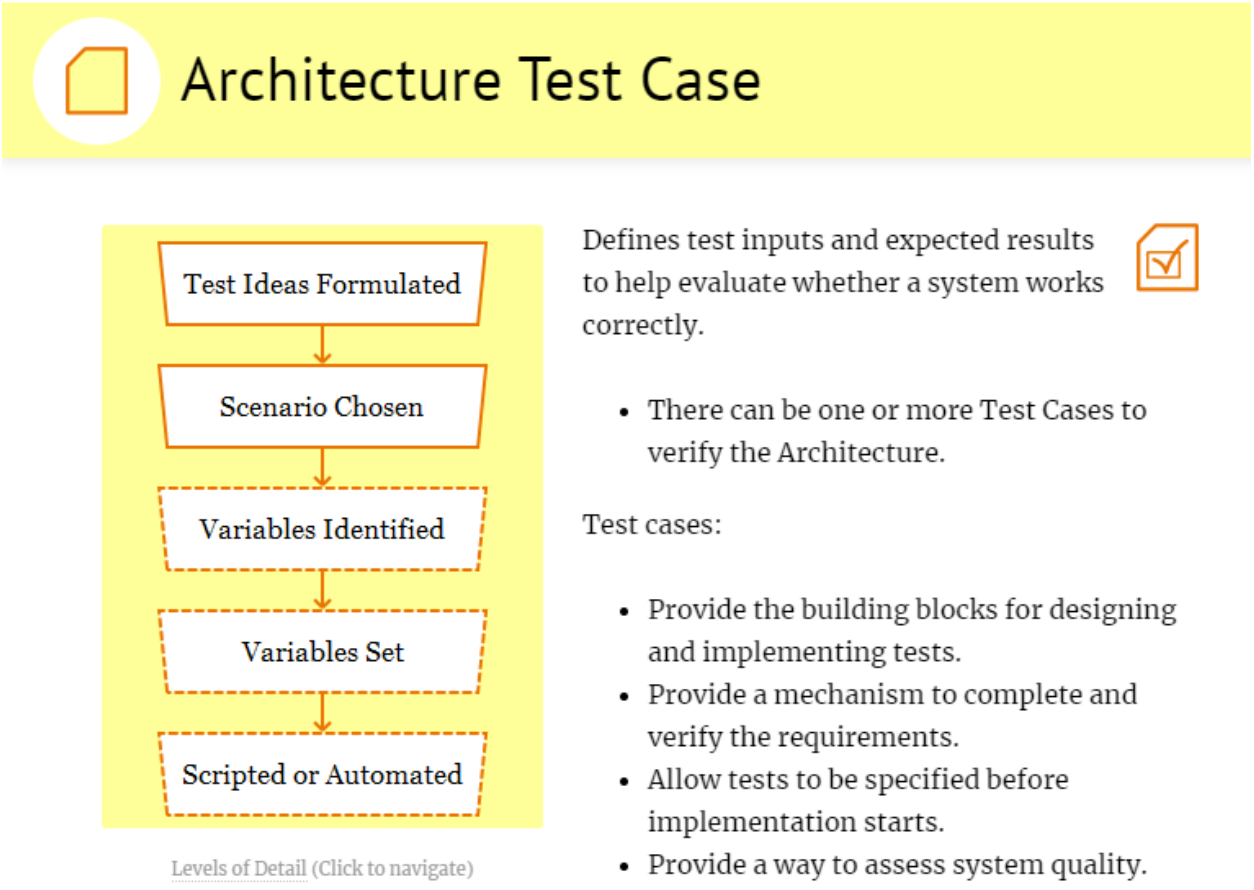


Рисунок 6 – Work Product "Architecture Test Case"

Work Product (Рабочий продукт) является отражением более общего термина «артефакт», достаточно широко применяемого в программной инженерии. Work Product используется для того, чтобы описывать создаваемые, модифицируемые и используемые результаты выполненных проектных активностей или задач в тех или иных практиках. Work Product по аналогии с Alpha также

обладает собственной структурой прогресса (которая не показана на концептуальной диаграмме): прогресс для Work Product измеряется в Level of Details (Уровне детализации), который аналогично обладает набором утверждений, семантически описывающих, что Work Product находится в том или ином состоянии. Однако есть существенное отличие от механизма Alpha State: согласно логике языка, для того чтобы считалось, что Alpha достигла определенного Alpha State, все утверждения этого Alpha State должны быть истинными, но для того, чтобы считалось, что Work Product достиг определенного Level of Details, достаточно, чтобы хотя бы одно утверждение было истинным. Второе, не столь существенное, отличие заключается в том, что авторы Essence вводят два подтипа для Level of Details, один из которых не имеет явного названия, а другой называется Sufficient Level (Достаточный уровень) и обозначается графически пунктирной линией. Это означает, что рабочий продукт, достигший этого уровня, считается достаточно детализированным, однако существуют практики, в рабочих продуктах которых есть несколько последовательных достаточных уровней (пример такого рабочего продукта на рисунке 6).

Помимо перечисленных Alpha, Alpha State и Work Product, в Essence существует достаточно важный элемент, который часто используется для описания практик, а именно Sub-Alpha (суб-Альфа), во многом схожий с ALPHa. Из опубликованных примеров методов можно сделать вывод, что Sub-Alpha описывает какое-то более конкретное явление, напрямую связанное с практикой. Например, достаточно известный и распространенный подход к работе с функциональными требованиями – Use Case (Варианты или Прецеденты использования) авторами Essence называется практикой, где каждый конкретный вариант использования является Sub-Alpha, относящейся к требованиям, и фиксирует состояние конкретного варианта использования, у которого есть рабочие продукты в форме описанных сценариев варианта использования.

Описанные выше сущности относятся к категории Things to work with (Вещи, с которыми работать) и фактически описывают структурные элементы ядра или практики, которые, с одной стороны, фиксируют прогресс проекта, с другой стороны, все так или иначе обладают механизмом состояний (Alpha state для Alpha и Sub-Alpha и Level of Details для Work Product). Далее следует описание сущностей, введенных в Essence для фиксации динамических аспектов процессов разработки, а именно Activity Space и Activity, примеры которых приведены на рисунках 7 и 8 соответственно.

Activity Space (Пространство активностей) задает пространство действий, которые необходимо выполнить для реализации проекта по разработке программного обеспечения. Это достаточно высокий уровень абстракции; можно допустить аналогию, что задается набор абстрактных классов (интерфейсов), необходимый для реализации проекта.

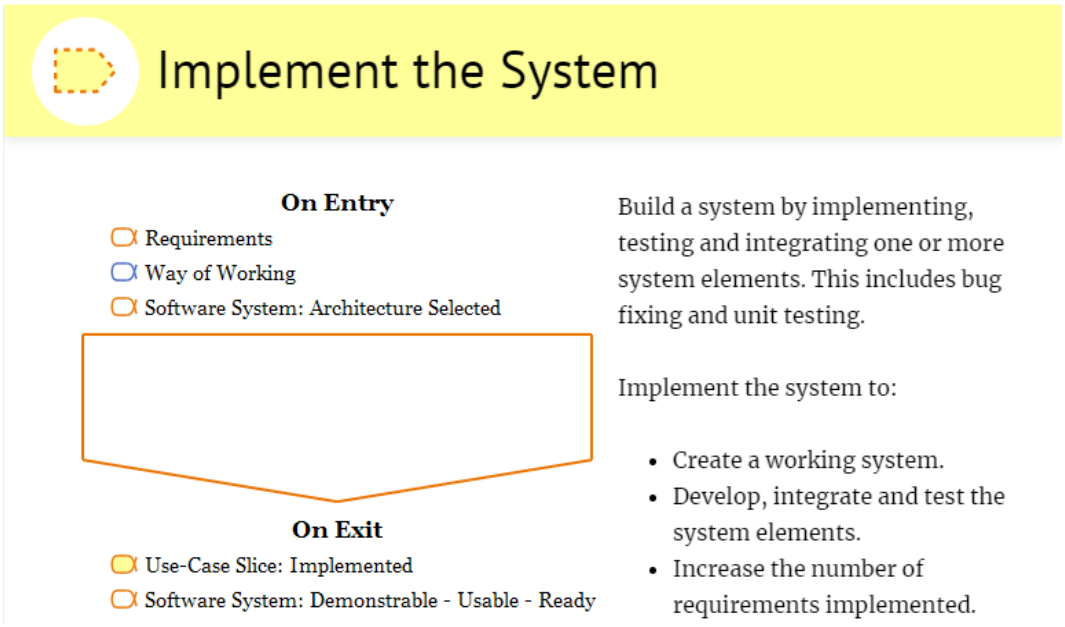


Рисунок 7 – Essence Activity Space "Implement The System"

По аналогии для реализации этих абстрактных интерфейсов введены элементы Activity, которые определяются в конкретных практиках и принимают на себя часть обязательств, зафиксированных их предками из Activity Space.

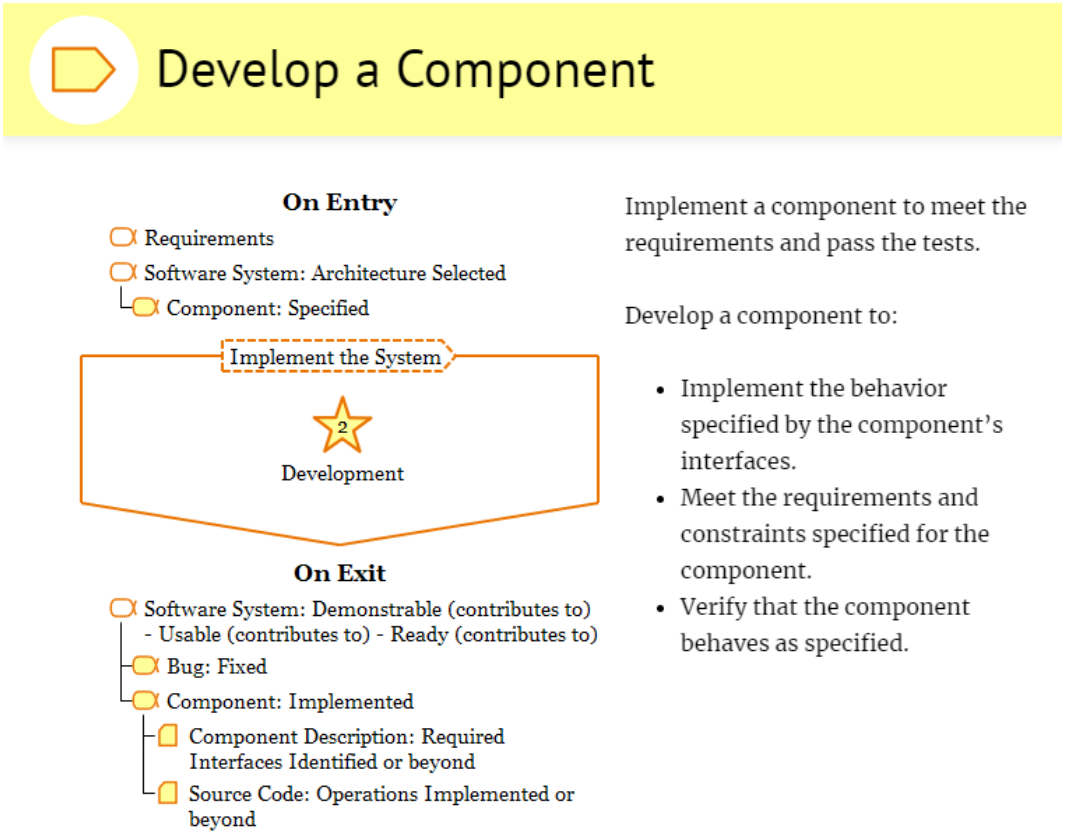


Рисунок 8 – Work Product Develop a Component

Аналогично Activity Space указываются входные и выходные множества структурных элементов, которые играют абсолютно схожую роль. Фактически основная разница заключается в том, что Activity Space фиксирует уровень описания всего вида деятельности, а Activity конкретного типа деятельности. Необходимо отметить, что несмотря на семантически логичную вложенность, и семантически строгую принадлежность Activity к Activity Space, и даже приведенную аналогию с абстрактными классами и наследниками, согласно стандарту и существующим опубликованным примерам практик это не совсем так. При описании практик Activity может вовсе не принадлежать или принадлежать одновременно нескольким Activity Space (этот парадоксальный факт также был явно подтвержден в переписке с одними из авторов Essence – Иваром Якобсоном и Полом Э. Мак-Мэхоном).

Формально процесс представления метода или практики с использованием нотации Essence представлен на рисунке 9. Можно заметить, что такой подход явно выделяет два уровня описания метода или практики: теоретический (левая часть рисунка) и прикладной (правая часть рисунка). При этом элементы ALPHa, а точнее ее состояний, связывают теоретическое описание с его практической реализацией в виде конкретных активностей и рабочих продуктов, которые разрабатываются в процессе реализации проекта.

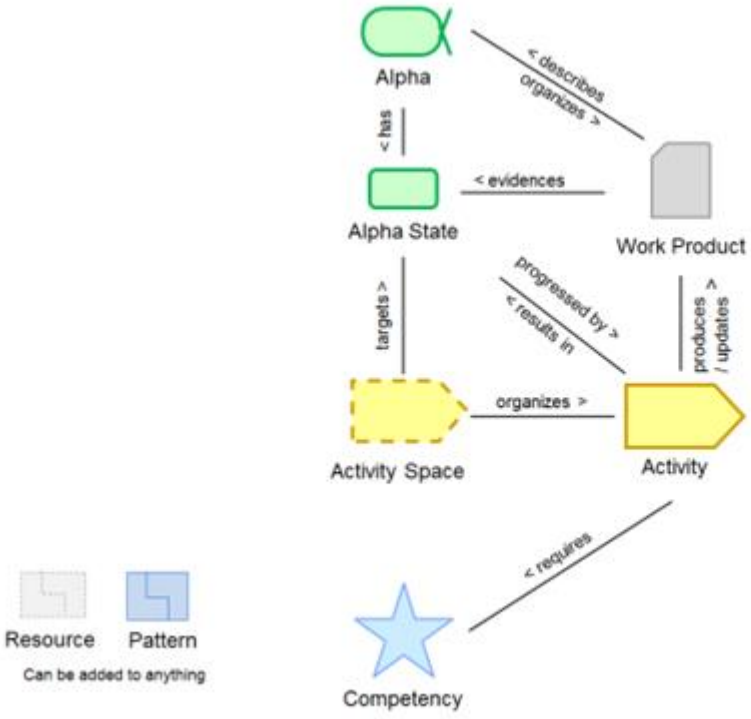


Рисунок 9 – Концептуальная схема языка Essence

Таким образом, можно отметить тот факт, что стандарт OMG Essence теоретически придерживается характерного для объектно-ориентированных парадигм подхода, основанного на понятии двойственности [86]. В рамках этого подхода выделяется два уровня: классы, которые фиксируют структурную составляющую модели, и объекты, которые предназначены для реализации модели в динамике во время работы программной системы. Основываясь на этом подходе, выстраивается общая теоретическая структурная модель, которую предлагается считать универсальной для описания любой практики разработки программного обеспечения.

В рамках этой модели на общем абстрактном уровне концептуальное (теоретическое) описание метода программной инженерии – теоретический контур Essence – выглядит следующим образом. Для каждой из трех основных зон стандарта (Customer, Solution и Endeavor) определен набор элементов Alpha и Alpha State. Переходы между состояниями различных Alpha зафиксированы с помощью соответствующих параметров Activity Space: в совокупности этот набор обеспечивает теоретическую полноту элементов, представленных в ядре OMG Essence. Схематически эти отношения представлены на рисунке 10.

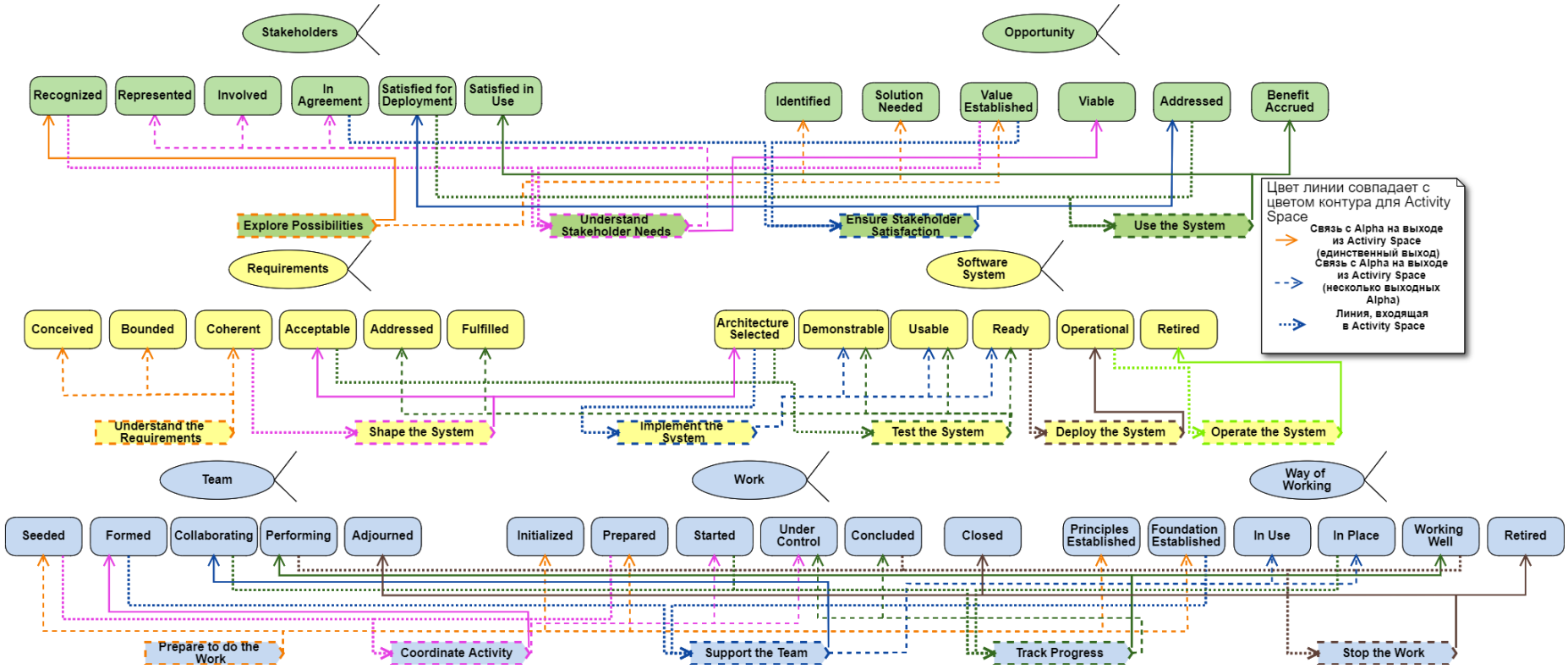


Рисунок 10 – Теоретический контур Essence

Нетрудно заметить, что в теоретическом контуре основной поток взаимодействия сущностей, определенных стандартом Essence, осуществляется между элементами Activity Space, которые принадлежат конкретной зоне. Связь с Alpha обеспечивается на уровне параметров входа для каждого элемента Activity Space двумя разными вариантами (пример на рисунке 11).

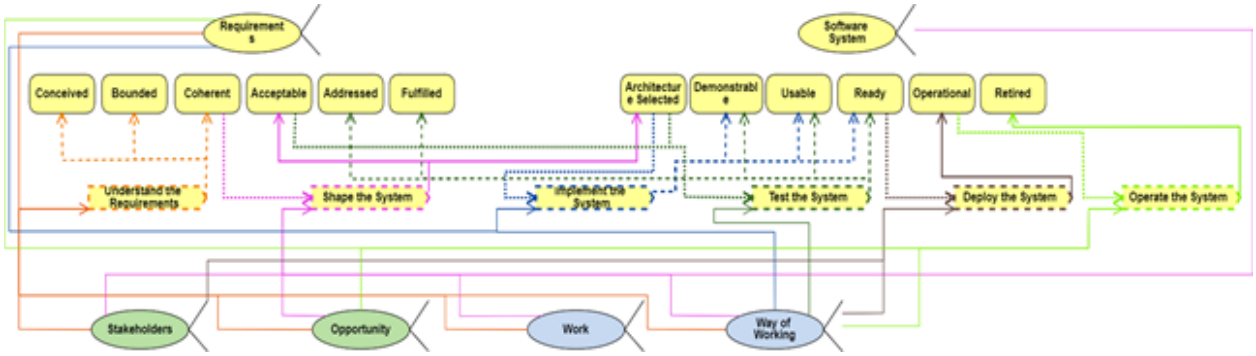


Рисунок 11 – Детализация теоретического контура для зоны Solution

Первый вариант – это ассоциация следующего содержания: выполняя данную Activity Space, необходимо учитывать текущий прогресс связанных Alpha (например, Alpha «Opportunity» – входная Alpha для Activity Space «Shape the System»). Второй – это ограничение, суть которого в следующем: чтобы можно было реализовать данную Activity Space, определенная Alpha должна достигнуть конкретное Alpha State (например, для Activity Space «Shape the System» Alpha «Requirements» должна быть в Alpha State «Coherent»).

Фактически механизмы описания общего теоретического контура этим ограничиваются, и в результате получается, что практически любой проект по разработке программного обеспечения на абстрактном уровне выглядит одинаково. Различие же будет основываться на том, как именно реализовать те или иные виды деятельности в ходе реализации проекта, т. е. какие практики будут использоваться в процессе формирования конкретного метода. Этот переход от общего уровня к частному (или практическому для актуальных проектов) осуществляется за счет описания отдельных практик различных процессов разработки в терминах Essence, но необходимо отметить, что сам процесс описания практик на текущий момент еще не имеет стандарта или руководства, в связи с чем возникают некоторые особенности при попытках прикладной разработки с использованием практик.

1.3. Особенности описания практик Essence

Рассмотрим процесс описания практики с помощью языка Essence на примере описания Product Backlog Essentials [87]. Фактически для описания практики с помощью стандарта необходимо определить набор соответствующих элементов Activity и Work Product, выделить набор Sub-Alpha для фиксации процессов, за состоянием которых команда должна следить, и связать это множество элементов с базовыми элементами ядра OMG Essence. Пример практики «Product Backlog Essentials» приведен на рисунке 12.

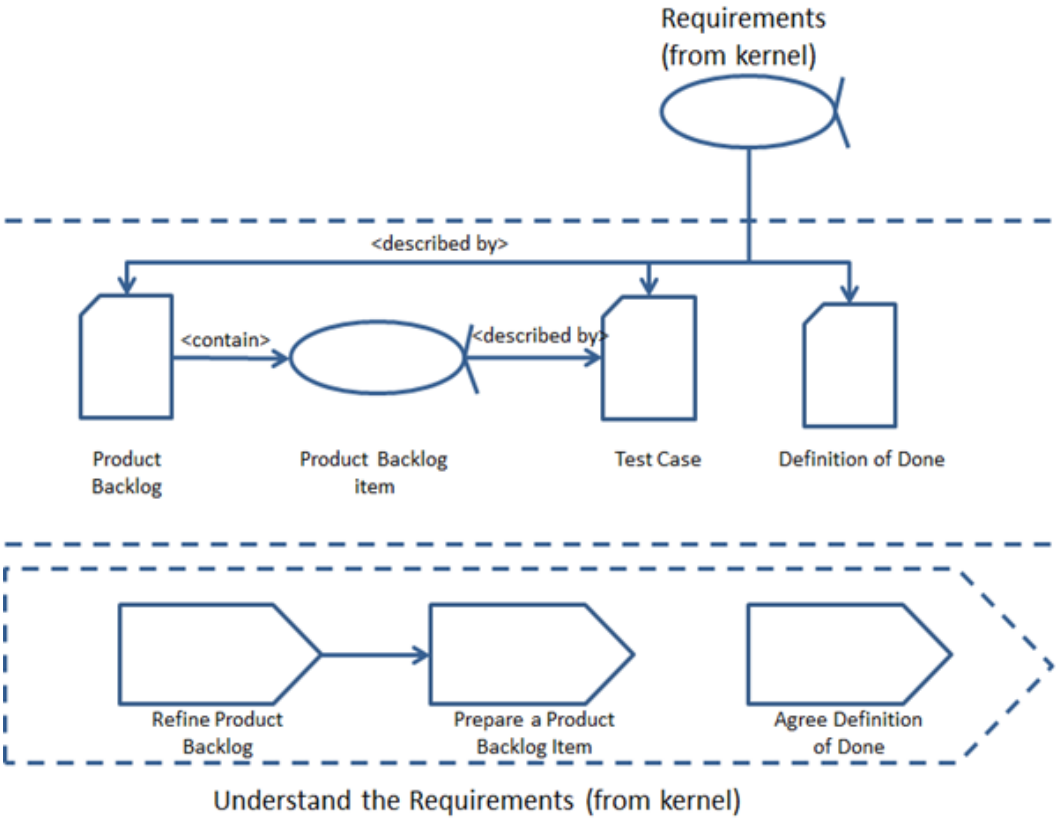


Рисунок 12 – Практика Product Backlog Essentials [87]

В этом примере исполнитель, описывающий практику, вводит конкретные объекты Activity (Refine Product Backlog, Prepare a Product Backlog Item, Agree Definition of Done) и связывает их с одним или несколькими элементами Activity Space. Наличие этой связи переносит ограничения теоретического контура в плоскость возможности их применения в практической деятельности. Далее определяются связи между Activity и Work Product (Product Backlog, Test Case, Definition of Done), а для контроля над прогрессом выполнения реальных работ вводятся элементы Sub-Alpha. Таким образом, именно процесс погружения практики в язык стандарта обеспечивает связь

теоретического и практического контура. Заметим, что именно наличие такого описания делает набор устоявшихся действий практикой в терминах стандарта. Примеры такого рода описаний можно найти в работах исследователей [72, 73].

Необходимо отметить, что ни в описании теоретического контура Essence с указанием общих Alpha и Activity Space, ни при описании практик, для Activity не указывается, на выполнения каких именно Checkbox эта Activity направлена. Например, если для практик, которые детализируют Sub-Alpha, фиксируется, что Activity на выходе имеет Sub-Alpha в нужном состоянии, тогда можно только предположить, что в таком случае в процессе выполнения Activity все условия для выполнения всех Checkbox указанного состояния Sub-Alpha должны быть выполнены. Но при этом для общих Alpha, которые описывают общее состояние проекта, вопрос об отметке Checkbox и, следовательно, об определении текущего достигнутого состояния, никак не решается используемой практикой.

В результате вопрос о выполнении Checkbox и достижении Alpha нужных Alpha State остается на уровне принятия экспертного решения командой и менеджментом проекта. При этом прикладное применение практики на представленных диаграммах ограничивается элементами Work Product и Activity и, как следствие, нуждается в более детальном описании с точки зрения использования Essence в рамках реализации проекта. В качестве примера на рисунке 13 детализируется представленная выше практика Product Backlog Essentials (рисунок 12).

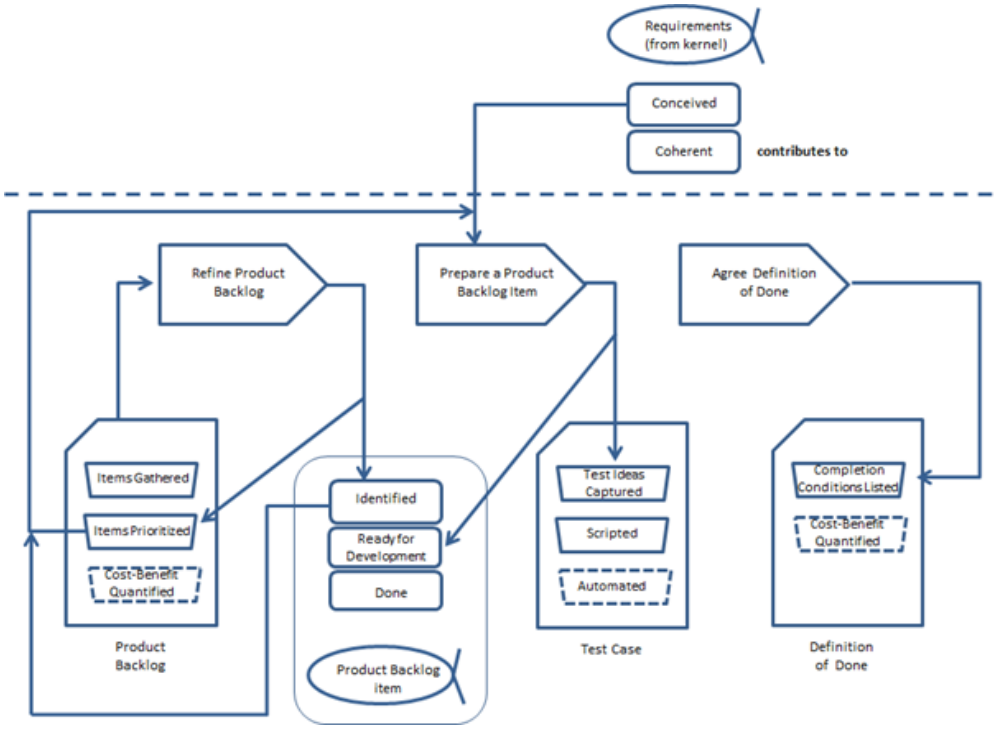


Рисунок 13 – Детализация практики Product Backlog Essentials [87]

Легко заметить, что ассоциации детализируются изменениями состояний Sub-Alpha и уровней детализации Work Product. При этом даже для такой достаточно несложной практики очевидно, что диаграмма не дает полного представления о взаимосвязях элементов.

Таким образом, можно провести аналогию, что практики Essence фиксируют конечные автоматы, в которых Sub-Alpha, Work Products – фиксируют возможные состояния и их последовательность, а Activity определяют правила перехода между возможными состояниями. Описывая конечный автомат в контексте управления проектами, можно получить следующий высокоуровневый алгоритм.

1. Создается экземпляр Activity в виде назначения конкретных задач конкретным исполнителям, играющим соответствующие Team Role.

2. Исполнение этих задач изменяет состояния связанных экземпляров класса Work Product или создает новые Work Product.

3. Назначенная задача признается выполненной, тем самым экземпляр Activity считается завершенным.

4. На основании текущего состояния экземпляра класса Work Product принимается решение о полноте реализации соответствующих элементов экземпляра класса Sub-Alpha (Checkbox, State) или более глобальных основных Alpha.

5. Прогресс, достигнутый при выполнении нескольких итераций шагов 1–4 на уровне экземпляра класса Sub-Alpha, позволяет принять решение об изменении текущего состояния экземпляра родительской Alpha (Checkbox, State).

Приведенный выше алгоритм позволяет связать общие элементы управления проектами по разработке программного обеспечения с практиками и методами, описанными в языке Essence, фактически адаптируя описание практик и методов в дополнительные описания к задачам, указывающие то, как они должны выполняться и с какими артефактами работать.

Выводы главы

Необходимо отметить, что OMG Essence, с одной стороны, действительно является онтологической моделью для описания процессов разработки разных типов, с другой – наличие в нем нескольких разных уровней абстракции элементов позволяет как формализовать высокоуровневый прогресс в разработке программного обеспечения, так и описывать и фиксировать отдельные элементы практик, используемых в различных процессах разработки.

Однако важно учесть несколько возможных практических трудностей, связанных с потенциалом активного использования Essence в прикладных задачах программной инженерии.

Во-первых, сам по себе Essence является одновременно и языком, и подходом к описанию и анализу процессов разработки, но, к сожалению, компании не готовы выделять достаточно ресурсов на решение вопроса эффективности выбора применяемых процессов разработки, в связи с чем в профессиональной деятельности программных инженеров изучение и практическое применение данного языка не становится повсеместной практикой. Иными словами, существует достаточно высокая вероятность, что потенциальная выгода от использования Essence в большинстве компаний будет недостаточной для принятия решения об обучении новому подходу руководителей компании.

Во-вторых, даже в случае, если проектный менеджер приобретет навык работы с Essence, это не освободит его от необходимости изучать другие процессы разработки и их практики, поскольку сам Essence является только языком описания, который упрощает процессы трансформации и перехода к другим практикам, но не исключает их полностью.

В-третьих, на момент написания данной работы развитие Essence и подходы к описанию методов с помощью этого стандарта демонстрируют, что практики и их элементы слишком обособлены: по сути, кроме как семантически, они никак не взаимосвязаны с другими практиками и их элементами на уровне связей самого Essence. Как следствие, даже если менеджер, используя язык Essence, будет изучать другие методы, он зачастую будет использовать набор отдельных практик, которые не складываются в единую и строгую картину управления проектами. Например, Essence вводит семь основных Alpha на любой проект, однако нет никаких ограничений, на то, чтобы метод был обязан описывать практики для работы со всеми семью альфами, в связи с чем можно получить следующую картину: автор какого-либо метода в его описании никак не раскрывает способы работы с требованиями, и Essence этому никак не противоречит.

Как вывод из вышесказанного, можно сформулировать следующую задачу: для более формального и эффективного практического применения языка, который «должен вернуть программной инженерии инженерную составляющую» [63], необходимо в целом упростить его, а в частности позволить использовать формальный язык описания процессов разработки, чтобы автоматизировать работу по конфигурированию среды управления проектами под требования и правила конкретного процесса разработки.

В текущих же условиях из независимости практик друг от друга следует также дополнительная особенность: для описания процедуры переноса метода Essence в практически

используемые системы управления проектами необходимо и достаточно получить алгоритм переноса отдельной практики и при необходимости повторить эту процедуру для остальных практик, входящих в выбранный метод Essence, учитывая общие ограничения языка (например, уникальность основных Alpha). Решению этой задачи посвящена вторая глава этой диссертации.

Основные результаты, изложенные в данной главе, опубликованы автором диссертации в составе исследовательских коллективов в научной периодике [89, 90].

2. ИМПОРТ МЕТОДОВ, ОПИСАННЫХ НА ЯЗЫКЕ ESSENCE, В СРЕДУ УПРАВЛЕНИЯ ПРОЕКТАМИ

В целом вопросы, связанные с процессами разработки, достаточно существенны и сами по себе, поскольку фиксируют некоторые абстрактные формы деятельности и их виды, осуществляемые в течение выполнения заказной разработки программного обеспечения. Однако необходимо понимать, что на текущем уровне развития технологий общепринятой практикой является использование большого количества вспомогательных инструментов, которые позволяют проектным командам частично автоматизировать рутинные задачи. К таким инструментам можно отнести следующие системы: контроля версий программного кода, автоматизированного тестирования, автоматического развертывания программных систем, автоматического создания резервных копий, балансировщики нагрузки и др.

При этом интересно, что развитие таких систем направлено, в первую очередь, на задачи, стоящие непосредственно перед разработчиками и исполнителями, а системы, которые упрощали бы работы более высокого уровня абстракции, развиваются не слишком активно и редко изменяются или принципиально улучшаются, хотя большинство из них существует в этой профессиональной деятельности достаточно давно. К подобным можно отнести системы контроля требований, системы, поддерживающие процедуры прямого проектирования и model-driven-development подходы [91], и, наконец, системы управления проектами. Анализ особенностей процессов разработки показывает, что они зачастую отображаются именно в системах управления проектами. Это схематично представлено на рисунке 14.

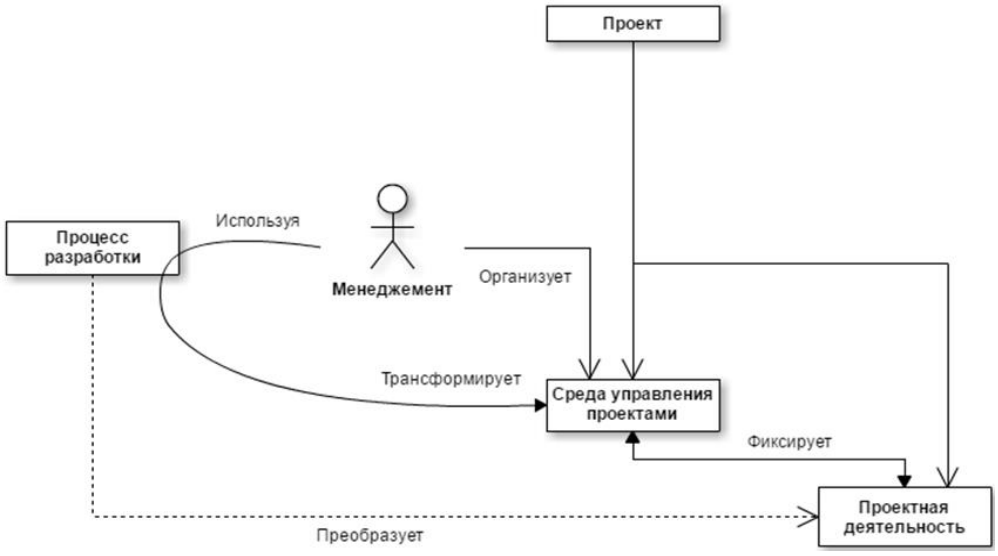


Рисунок 14 – Место процесса разработки в структуре реализации проекта

Изначальная трансформация[12] и поддержка выбранного процесса разработки является экспертной (неавтоматизируемой) компетенцией менеджмента проекта, что влечет за собой следующие важные особенности:

– практически любой новый процесс разработки или его модификация потребует от менеджмента ресурсов на изучение, понимание и дальнейшую реализацию изменений в среде управления проектами;

– сам процесс трансформации основан исключительно на человеческом факторе и его крайне сложно формализовать: как следствие, качество переноса процесса разработки и то, насколько подходят внесенные в среду управления проектами изменения, не измеримо объективными показателями;

– в ходе анализа результатов проекта выводы относительно конкретного процесса разработки или его реализации в выбранной среде управления проектами будут субъективны, что препятствует накоплению общего полезного для всей отрасли опыта;

– менеджмент проекта может оказаться недостаточно компетентен в выбранном им процессе разработки, в результате чего некоторые особенности процесса разработки могут быть транслированы в среду управления проектами нерационально – наиболее часто в концепции «следуя букве закона» вместо анализа принципов и адаптации их к конкретному проекту.

В результате складывается противоречивая ситуация: с одной стороны, именно на менеджмент возлагается достаточно сложная задача, качественное выполнение которой требует серьезных временных затрат на изучение специфики процесса разработки и его реализацию; с другой стороны, эти временные ресурсы обычно не выделяются в достаточном количестве, поскольку неочевидно, до какой степени необходимо модифицировать среду управления проектами или каких убытков это явно поможет избежать.

Таким образом, задача автоматического импорта процесса разработки и адаптации среды управления проектами довольно значима, поскольку, во-первых, успешное решение позволит снизить временные издержки для настройки среды управления проектами, во-вторых, использование созданных специалистами моделей процессов разработки на языке Essence позволят снизить требования к уровню компетентности менеджмента в этих процессах, а в-третьих, появление формальных правил переноса процесса разработки позволит накапливать объективные данные используемых практик для дальнейшего сравнения и анализа.

На момент написания данной работы единственным полнофункциональным источником данных, поддерживающим описание методов и практик Essence, является приложение «Practice

[12] Под трансформацией понимается адаптация процесса разработки к особенностям выполнения конкретного проекта.

Workbench» [52], созданное компаний «Ivar Jacobson International». Особенности представления данных и подход к построению полнофункционального case-средства, представленные этой компанией, рассмотрены в п. 2.1. В п. 2.2 описана разработанная в рамках настоящей работы промежуточная модель данных, предназначенная для решения задачи импорта практик и методов Essence в реальные среды управления проектами, а также представлена общая процедура переноса практик Essence в среды управления проектами. В п. 2.3 приведен пример имплементации процедуры в среду управления проектами «Redmine».

2.1. Особенности реализации модели данных в «Practice Workbench»

В открытом доступе в структурированном и формальном виде практики и методы Essence доступны через веб-приложение «Practice Library» [90], созданное компанией «Ivar Jacobson International» для популяризации стандарта и демонстрации его возможностей. В библиотеке доступны три метода и двадцать восемь практик. Доступ к материалам библиотеки является свободным (бесплатным), но требует регистрации. Пример описания Activity приведен на рисунке 15.

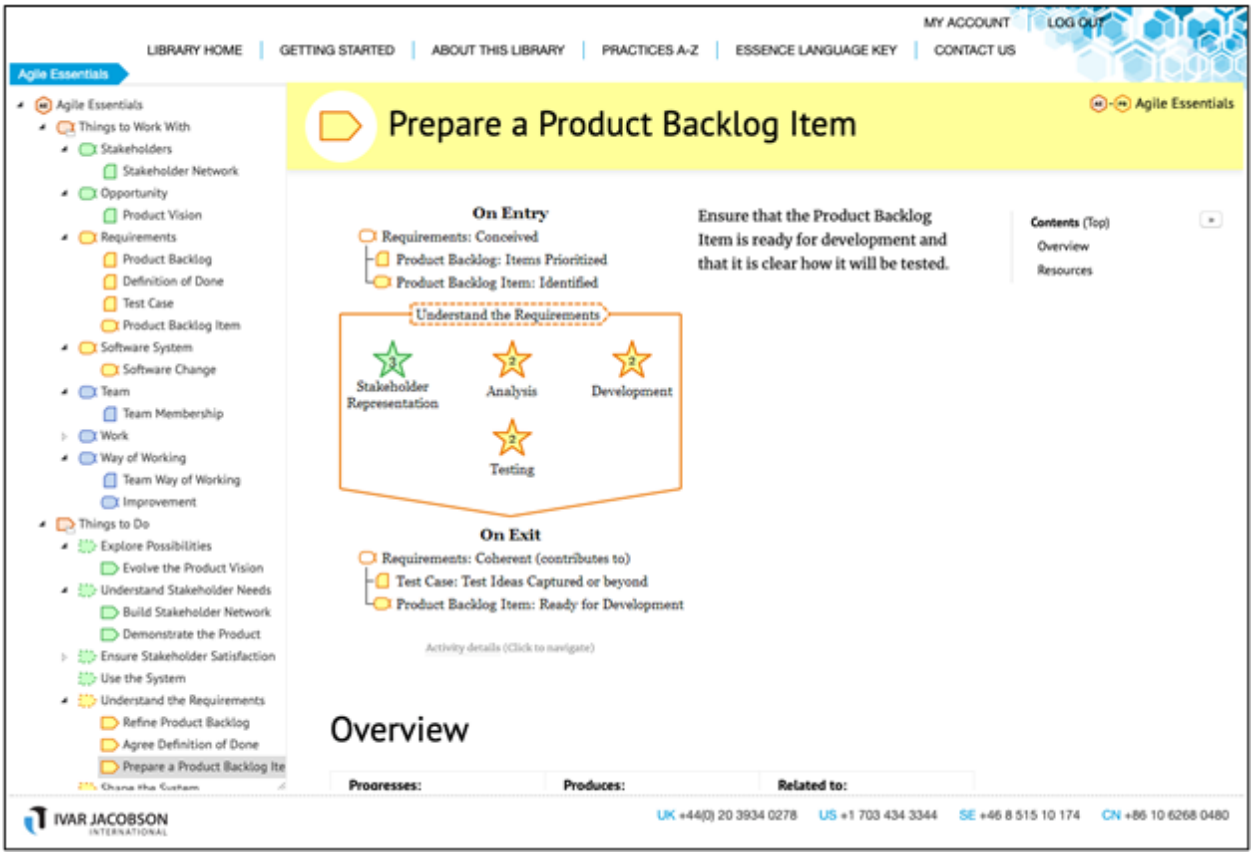


Рисунок 15 – Скриншот приложения «Practice Library»

Описания практик и методов в библиотеке создаются на основе model-driven-development подхода при помощи среды разработки практик и методов «Practice Workbench» [83], которая поддерживает импорт и экспорт готовых методов. Данное приложение распространяется уже в основном на коммерческой основе в виде лицензий на использование. Для выполнения данной работы была получена академическая лицензия на продукт и доступ к одному готовому методу – «Agile Essentials», в связи с чем разработка решения интеграции методов, описанных на языке Essence, будет рассмотрена на примере интеграции метода «Agile Essentials». Скриншот приложения приведен на рисунке 16.

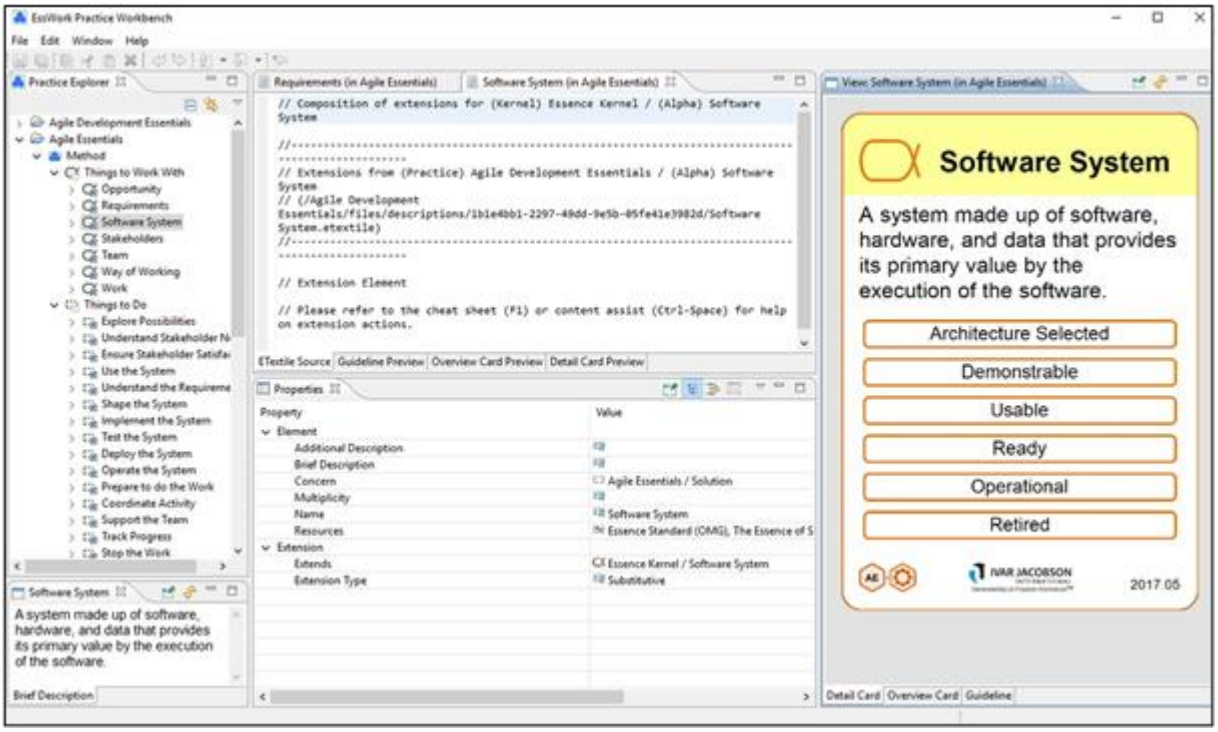


Рисунок 16 – Скриншот приложения «Practice Workbench»

Практики и методы в «Practice Workbench» экспортируются в файлы формата *.xml*, на рисунке 17 продемонстрирована часть кода, описывающего альфу Stakeholders из практики «Essence Kernel». Как можно заметить, структура файла выражена коллекцией вложенных элементов, каждый из которых имеет название, уникальный id, а также определенный набор параметров в зависимости от типа объекта, который этот элемент описывает.

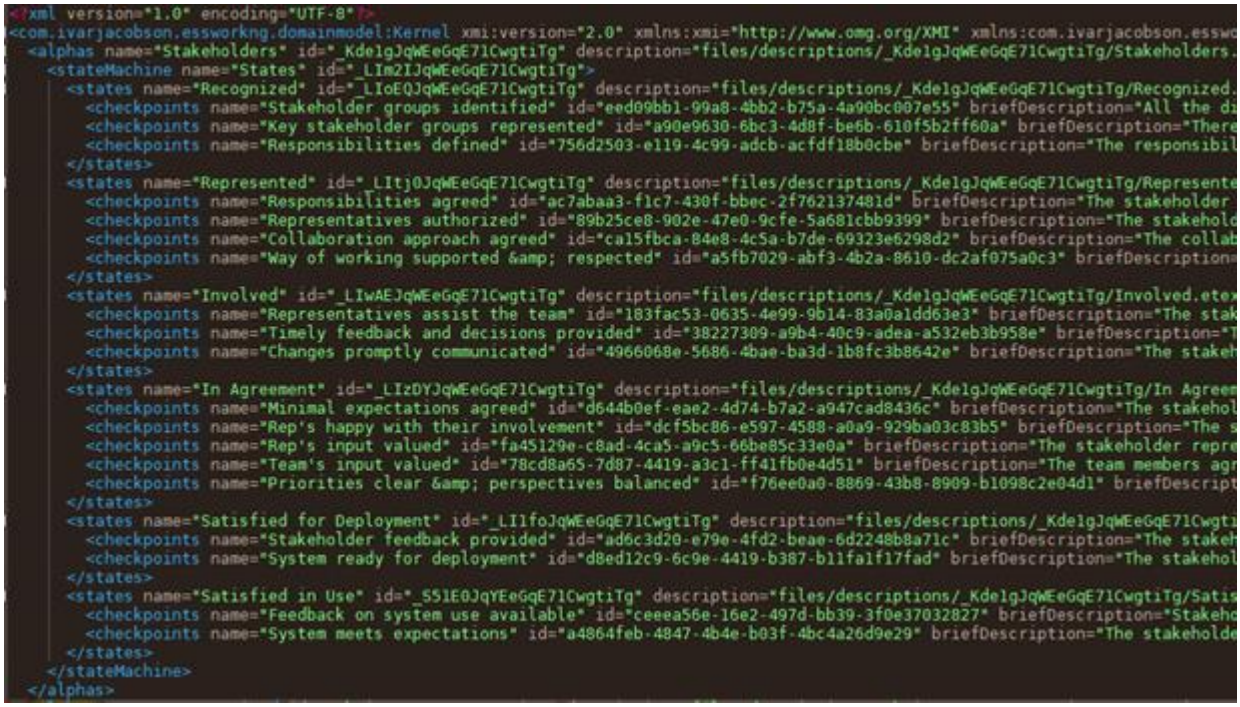


Рисунок 17 – Файл практики из приложения «Practice Workbench»

При этом, если какая-то практика использует объект, который был объявлен в другой практике или ядре, будет указано, что этот объект является расширением уже существующего, и дана ссылка на практику или ядро, где был изначально описан расширенный объект (элемент extends). Вложенность в данном случае необходима для описания связей между объектами, однако имеется существенный недостаток: если мы захотим получить, например, все рабочие продукты, с которыми взаимодействует практика, то придется для начала пройтись по всем альфам для того, чтобы их собрать, или найти конкретный рабочий продукт. Это сложно назвать эффективным решением, поскольку у каждого элемента существует уникальный id, который используется, в том числе, для того чтобы отобразить связь между какими-либо элементами.

Помимо файлов со структурой объектов практик, существуют файлы ресурсов, описаний, и прочего, необходимые для создания семантически полного описания практики по образу, аналогичному «Practice Library». Эти файлы представляют собой документальную базу, текстовые описания, иллюстрации и не влияют на структуру практик, но расширяют ее семантически или более полно описывают значение и роль отдельных элементов в практике.

Описываемый инструмент поддерживает валидацию практик и методов. В частности, невозможно создать внутри элементов того, чего семантически не может там быть, – например, альфу внутри состояния. Кроме того, в «Practice Workbench» можно создавать главенствующие элементы разных типов для уровней метода и практики. Например, для описания ядра указывается тип Kernel, для описания методов и практик

– Method и Practice соответственно. Причем в ядре невозможно сослаться, например, на элемент из практики: ядро может содержать только оригиналы объектов, на которые, в свою очередь, могут сослаться практики. Метод же не может содержать в себе какие-либо объекты, поскольку по определению состоит из практик: он может содержать только ссылки на элементы других практик или ядра.

Важно отметить, что возможность экспорта и импорта практик создана специально для экспорта из «Practice Workbench» и импорта в это же приложение. Специальных структур для импорта в другие инструменты не предусмотрено, как и не существует альтернативных инструментов, описывающих методы и практики, в связи с чем возникла необходимость изучить, как импортировать практики и методы в систему управления проектами, используя только файлы «Practice Workbench». Для этого потребовалось зафиксировать модель данных «Practice Workbench». На рисунке 18 в виде диаграммы UML [92] изображена модель элементов «Practice Workbench» так, как они представлены в *.xml* файлах.

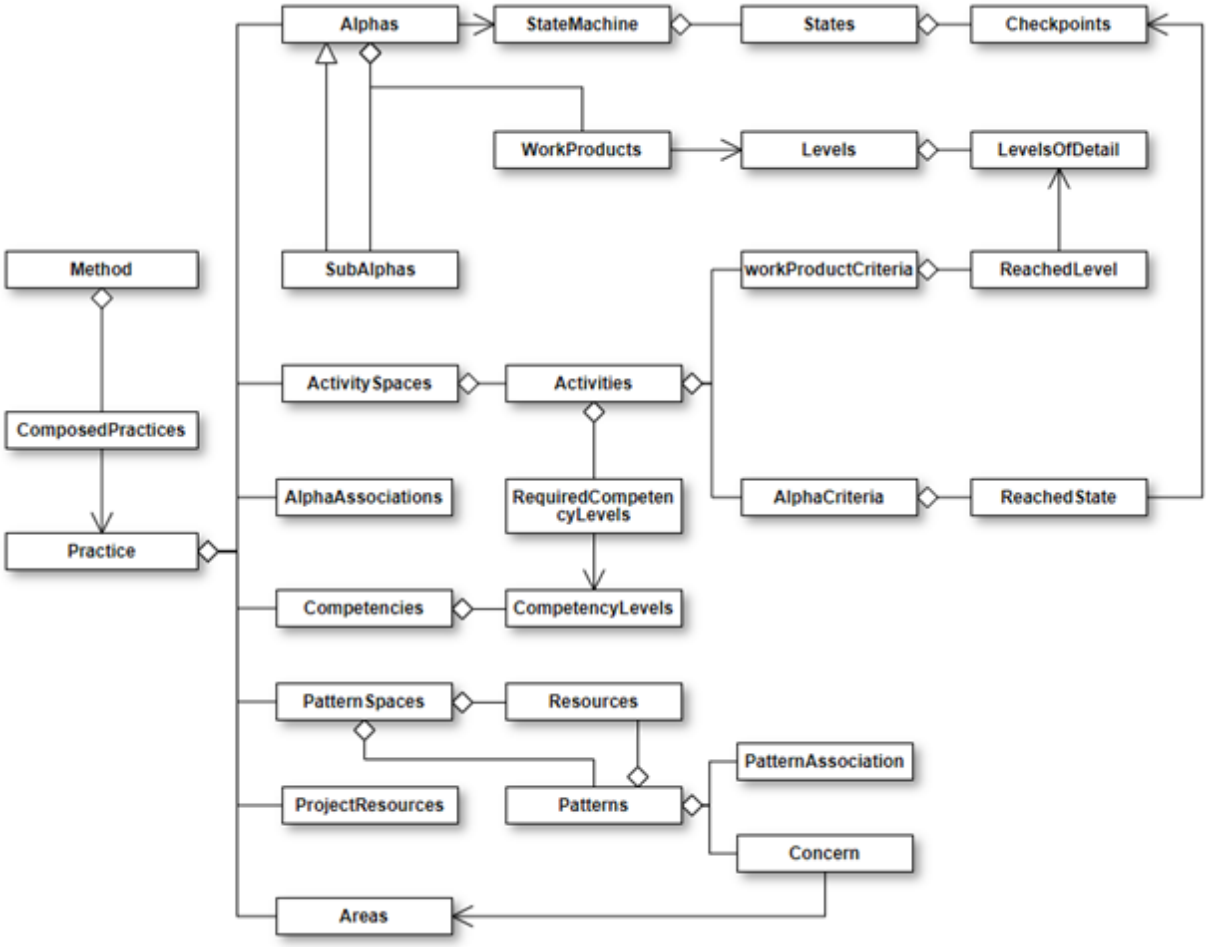


Рисунок 18 – Модель элементов «Practice Workbench»

Помимо общей схемы структуры объектов в «Practice Workbench», было решено представить структуру одной практики на диаграмме объектов. На рисунке 19 представлена диаграмма объектов для практики Product Backlog Agile Essentials в «Practice Workbench».

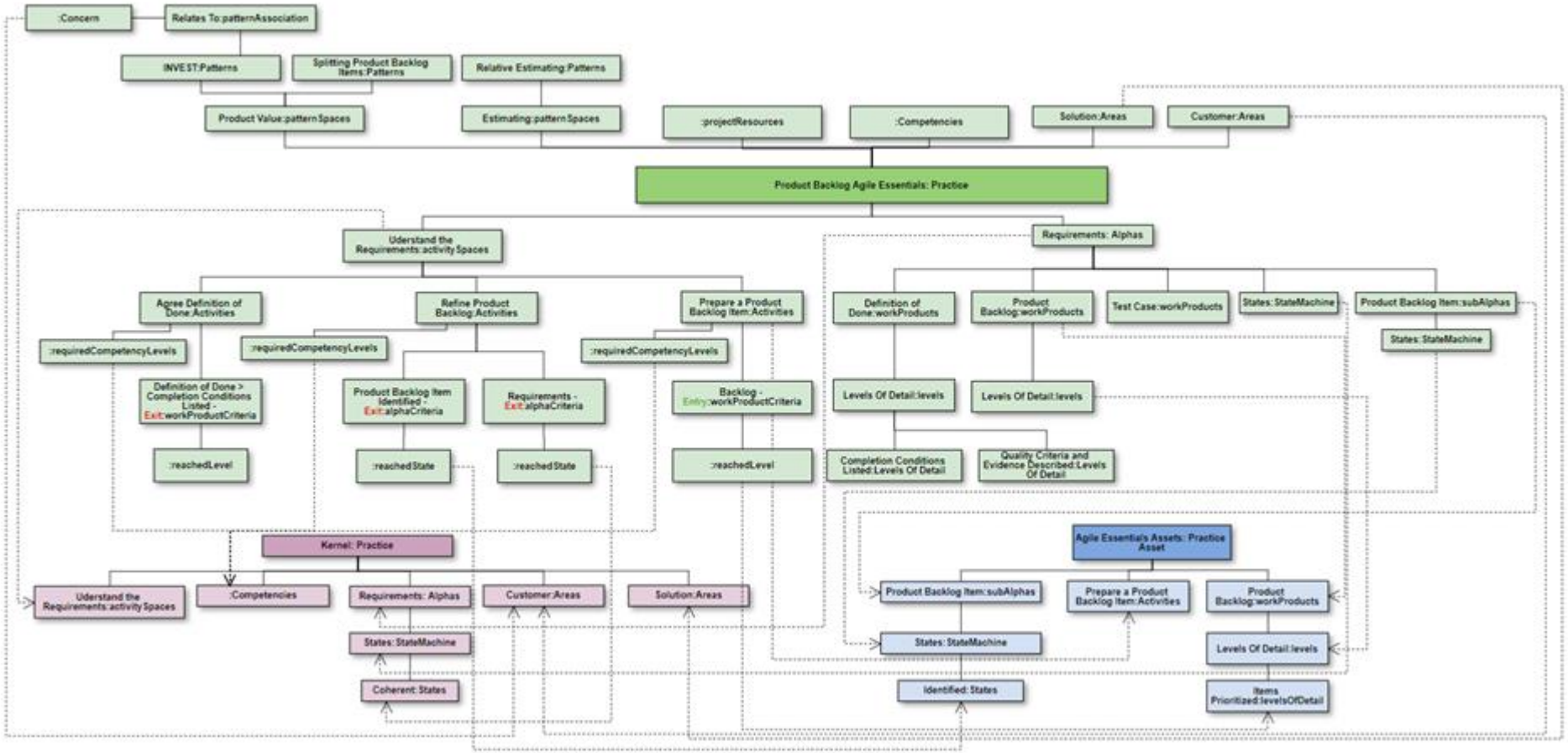


Рисунок 19 – Диаграмма объектов для практики Product Backlog Agile Essentials *в «Practice Workbench»*

На этой диаграмме наглядно отображаются связи между практиками, в частности, то, как выглядит расширение объектов одной практики какой-либо другой: некоторые элементы практики Product Backlog[13] расширяют здесь элементы ядра[14] и другой практики – Agile Essentials Assets[15].

На рисунке 20 представлен фрагмент диаграммы объектов, демонстрирующий, как именно выглядит связь между двумя практиками: практика Product Backlog расширяет действие из практики Agile Essentials Assets, добавляя в него новый входной критерий.

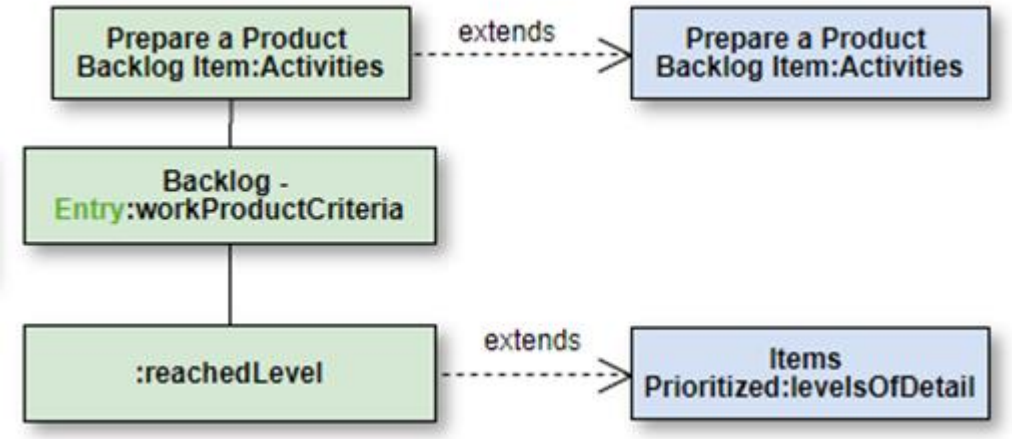


Рисунок 20 – Демонстрация связи между практиками в «Practice Workbench»

В ходе изучения модели языка Essence Language и исследования практики «Use Case 2.0 Essentials» [72] были обнаружены несоответствия между стандартом Essence и описанием практик в «Practice Library» и «Practice Workbench». При попытке представить описание практики из библиотеки, используя классы модели Essence Language, было обнаружено, что модель описания практики имеет дополнительные семантические элементы, которые невозможно выразить средствами языка Essence стандарта 1.2.

В качестве примера на рисунке 21 приведен скриншот действия «Prepare a Product Backlog Item» из «Practice Library», на котором изображены выходные критерии с дополнительным поведением, описанным как «contributes to» и «or beyond».

[13] Зеленый цвет на диаграмме.
[14] Красный цвет.
[15] Синий цвет.

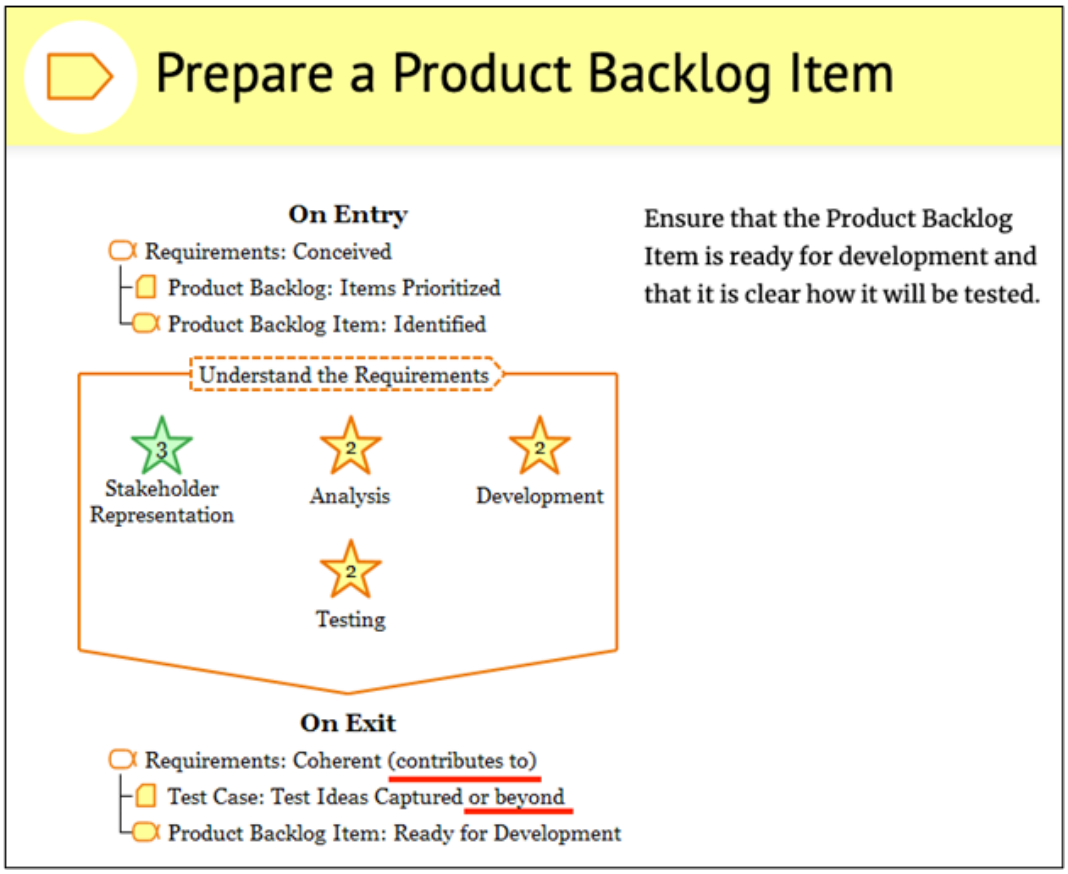


Рисунок 21 – Действие из «Practice Library» с нестандартным поведением

Поведение, описанное «contributes to», означает, что данное действие при своем завершении может как перевести альфу из состояния в состояние, так и способствовать ее переводу, т. е. выполнить некоторые утверждения, но не переводить альфу в указанное состояние. Таким образом, выполнение действия «Prepare a Product Backlog Item», скорее всего, не переведет альфу Requirements в состояние «Coherent», но выполнит некоторые элементы его контрольного списка.

Следующее поведение описывается фразой «or beyond», которая специфицирует, что требуемым состоянием или уровнем детализации некоторого критерия является не только явно указанное состояние или уровень детализации, но и последующие за ним. Например, при выполнении рассматриваемого действия рабочий продукт «Test Case» переходит в уровень детализации «Test Ideas Captured» или в следующие за ним «Scripted» и «Automated».

Необходимо понять, как данное поведение может быть описано при помощи модели языка стандарта Essence. Для этого рассмотрим рисунок 22, на котором изображена диаграмма классов пакета «Activity Space and Activity» из стандарта Essence 1.2.

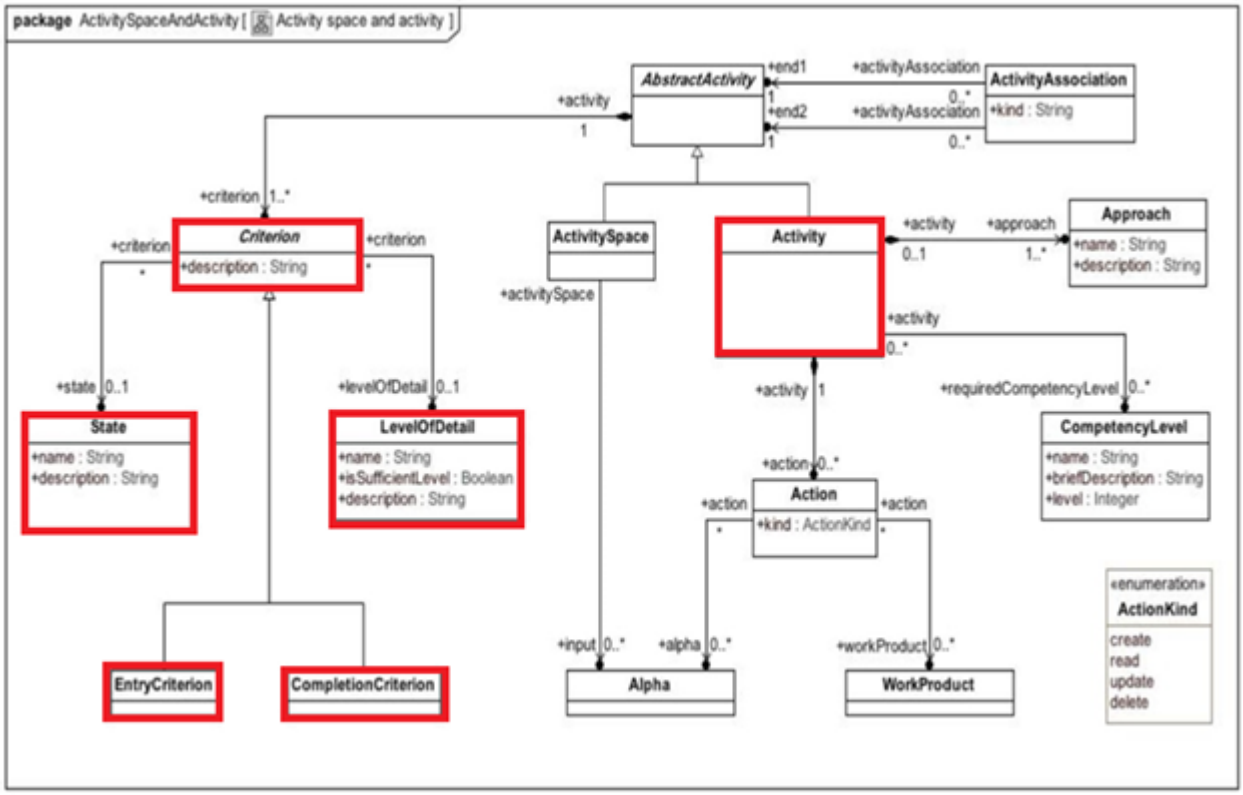


Рисунок 22 – Пакет «Activity Space and Activity» из стандарта Essence 1.2

На данной диаграмме представлены классы, описывающие модель действия с точки зрения стандарта. Красным цветом обозначены основные классы, относящиеся к действию и его критериям. Используя выделенные классы, невозможно описать поведение, представленное в «Practice Library». Рассматриваемое поведение можно получить при помощи расширения модели языка – например, добавлением в класс «Criterion» атрибутов-флагов, при установке которых соответствующий критерий описывал бы необходимое поведение. Таким образом, можно сделать вывод, что, используя модель, описанную в стандарте, невозможно описать поведение критерия из «Practice Library».

Тем не менее для решения задачи импорта методов было решено использовать приложение «Practice Workbench», поскольку оно обладает необходимым функционалом для работы с языком Essence. Кроме того, приложение поддерживает импорт методов и практик. Недостатком данного приложения является то, что файлы импорта методов и практик не предназначены для использования в системах управления проектами, поэтому необходимо было разработать способ переноса методов и практик из «Practice Workbench» во внешнюю среду управления проектами.

2.2. Промежуточная модель данных практик Essence для переноса во внешнюю среду управления проектами

Для решения задачи переноса практик и методов, описанных на языке Essence, в расширение среды управления проектами, они должны быть преобразованы специальным средством и пройти через несколько уровней моделей. Общая схема такого преобразования представлена на рисунке 23.

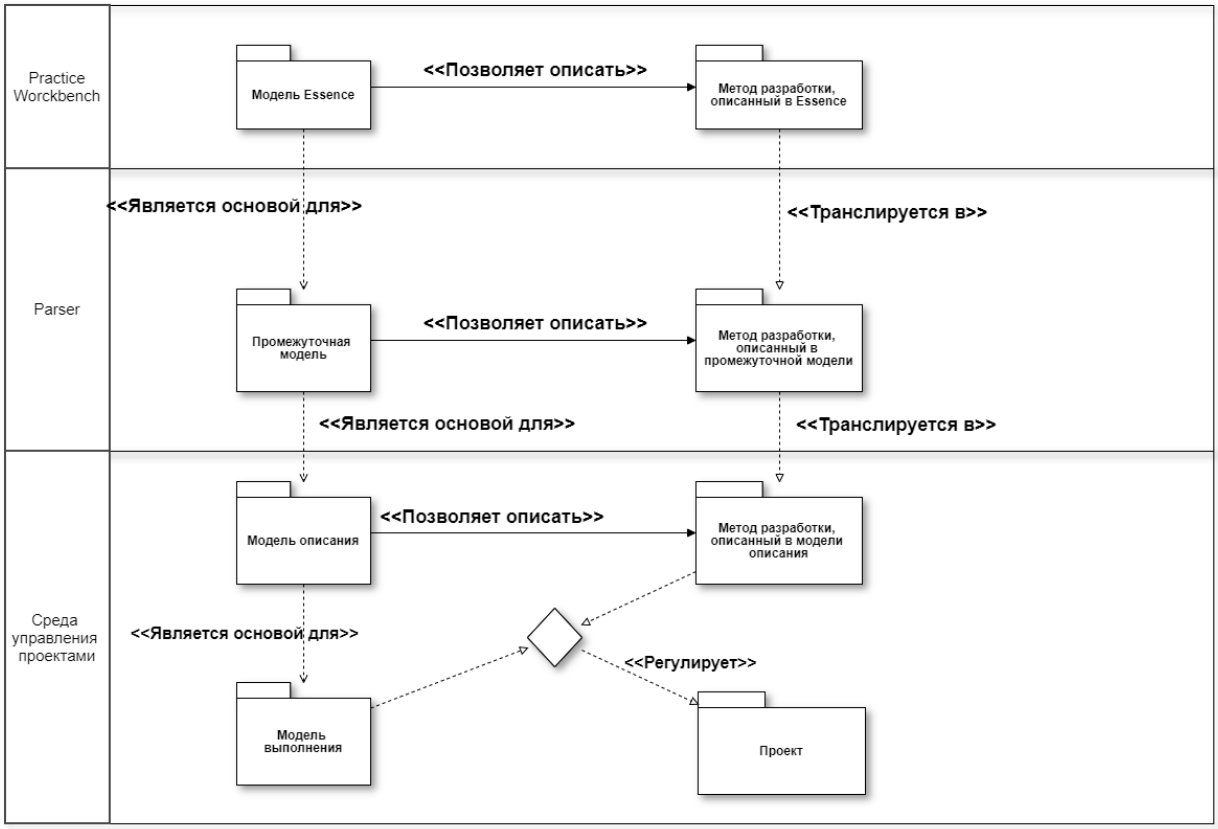


Рисунок 23 – Схема преобразования метода из «Practice Workbench» в среду управления проектами

При выполнении данного преобразования сначала рационально представить практики и методы из «Practice Workbench» в формат промежуточной модели, которую уже можно будет использовать для того, чтобы импортировать методы и практики в систему управления проектами. Разработка промежуточной модели, и преобразование практик в неё, позволит достичь следующих преимуществ:

– убрать прямую зависимость между «Practice Workbench» и расширениями для систем управления проектами;

– упростить структуру методов и практик для расширений систем управления проектами;

– уменьшить количество передаваемых объектов, таких как ресурсы, паттерны и прочие объекты, необходимые только для описаний.

На основании описанных выше причин была создана специальная промежуточная модель, которая определяет набор основных элементов языка Essence, необходимых для использования методов в системе управления проектами. По этой причине, в нее не вошли такие абстрактные элементы языка, как пространства действий, компетенции и др. Важно отметить, что при составлении промежуточной модели (рисунок 24) были учтены найденные несоответствия между стандартом и моделью описания «Practice Workbench».

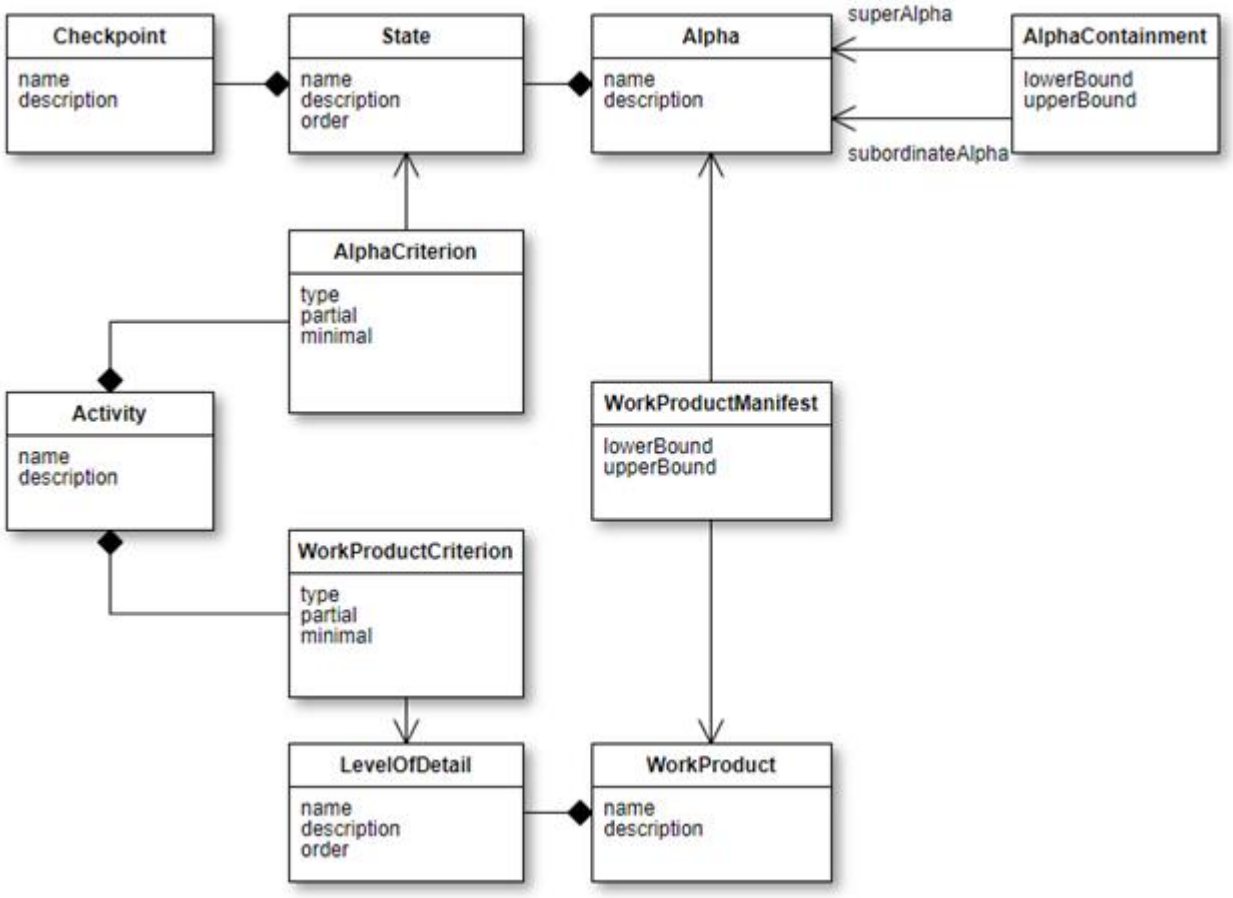


Рисунок 24 – Структура классов промежуточной модели

Рассмотрим значения элементов промежуточной модели.

Alpha (Альфа) – представляет описание альфы, используемой в методе. Каждая альфа имеет набор состояний (**State**). Состояния альфы располагаются в определенной последовательности, определяемой значениями атрибута «order». Текущее состояние альфы определяется при помощи набора контрольных пунктов (**Checkpoint**), относящихся к данному состоянию. Альфа считается

достигшей определенного состояния только в том случае, если выполнены все контрольные пункты текущего состояния и всех ему предшествующих.

AlphaContainment (Содержание альфы) – представляет описание вложенности альф друг в друга[16]. Каждая альфа может быть родительской альфой и суб-Альфой одновременно. Каждая родительская альфа может иметь несколько экземпляров одной и той же суб-Альфы, поэтому класс AlphaContainment хранит ограничения по количеству суб-Альф.

WorkProduct (Рабочий продукт) – представляет описание рабочего продукта, используемого в методе. Каждый рабочий продукт содержит набор своих возможных состояний, называемых уровнями детализации (**LevelOfDetail**). Уровни детализации располагаются в определенной последовательности, определяемой значениями атрибута «order».

WorkProductManifest (Манифест рабочего продукта) – представляет описание отношения рабочих продуктов к альфе. Объекты данного класса хранят ограничения по количеству элементов множества рабочих продуктов, относящихся к определенной альфе.

Activity (Действие) – представляет описание действия и имеет набор критериев - альф (**AlphaCriterion**) и набор критериев - рабочих продуктов (**WorkProductCriterion**). AlphaCriterion описывает критерий, при котором альфа должна быть в определенном состоянии. WorkProductCriterion описывает критерий, при котором рабочий продукт должен быть в определенном уровне детализации (состоянии рабочего продукта). Каждый из критериев имеет следующие атрибуты:

«type» – определяет, является ли критерий выходным или критерием завершения;

«partial» – определяет, должен ли данный критерий переводить объект в указанное состояние или должен способствовать его переводу[17];

«minimal» – определяет, могут ли рассматриваться последующие состояния требуемого объекта как возможные[18].

Для преобразования файлов методов и практик в файлы промежуточной модели было разработано приложение-преобразователь на языке C++. На вход приложение получает файлы метода и практик, используемых в этом методе. На выходе приложение генерирует файл формата *.json*, в котором содержатся элементы метода в промежуточной модели [93]. На рисунке 25 представлена работа преобразователя в форме диаграммы последовательности UML.

---

[16] Альфы могут иметь вложенные альфы, называемые суб-Альфами.
[17] Реализует описанное в § 2.1 поведение «contributes to».
[18] Реализует поведение «or beyond».

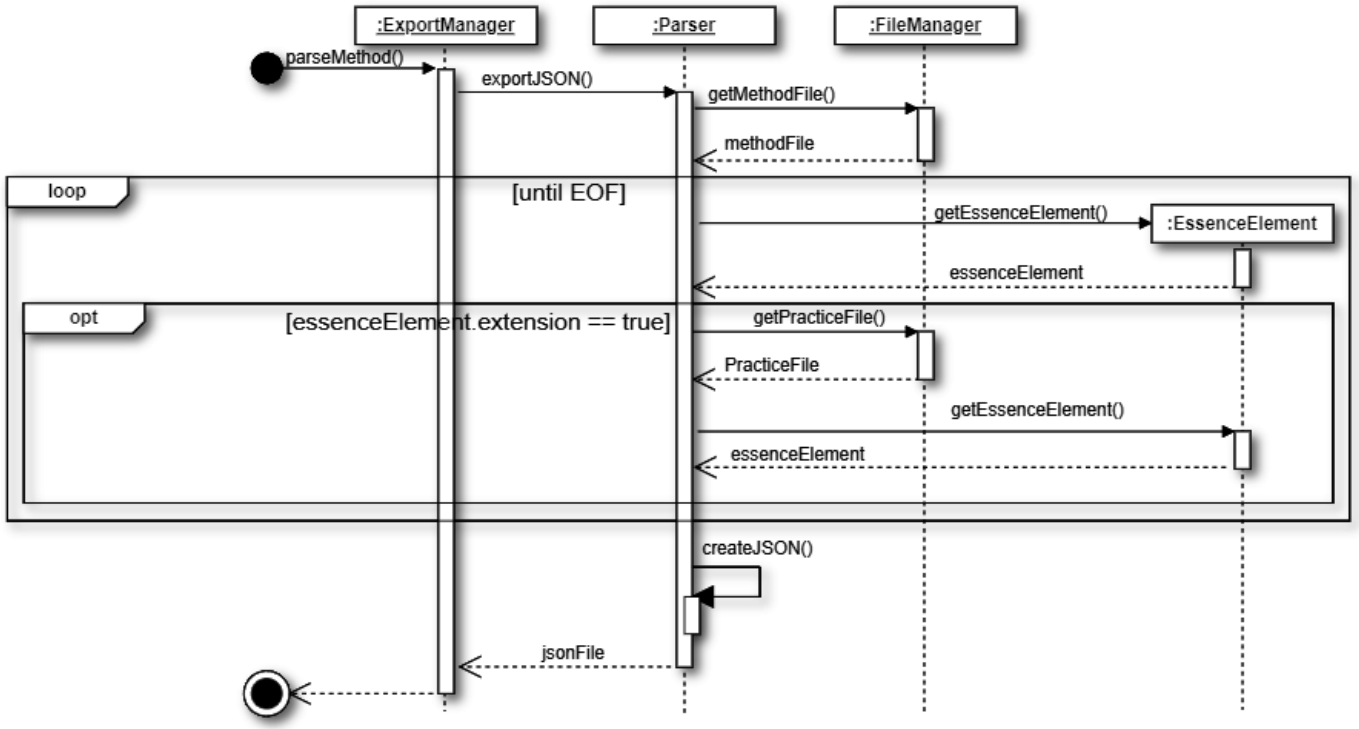


Рисунок 25 – Преобразователь промежуточной модели

Методы, которые планируется интегрировать в реальную среду управления проектами, описаны в промежуточной модели Такое описание метода будет являться уровнем M1 стандарта Meta-Object Facility, что в рамках MOF называется «моделью» (в рамках этой работы далее мы будем использовать термин «модель описания», чтобы отличать эту модель). Для работы с самим методом необходимо создавать конкретные экземпляры элементов (уровень M0), описанных уровнем M1 (для этой модели по аналогии, введем понятие «модель выполнения»). Например, для метода Agile Essentials модель описания определяет, что у альфы Требования (Requirements) может существовать от нуля до бесконечности рабочих продуктов Тестовый случай (Test Case). Таким образом, можно зафиксировать следующее требование к реальной системе управления проектами: необходимо, чтобы система управления проектами могла создавать объекты модели выполнения на основе элементов модели описания.

В итоге получаем достаточно универсальную процедуру переноса методов и практик на языке Essence Language в реальную среду управления проектами.

Шаг 1. Преобразуем метод или практику в формат промежуточной модели. В рамках данной работы для этого используются метод и практики из «Practice Workbench», но вполне возможно реализовать приложение, которое способно изначально работать с форматом промежуточной модели без преобразования файлов методов и практик в файлы промежуточной модели.

Шаг 2. Фиксируем механизмы расширения конкретной среды управления проектами и на основе этих механизмов реализуем необходимое множество элементов, поддерживающее модель описания. Связываем модель выполнения с представлением проекта на стороне конкретной среды управления проектами.

Шаг 3. Реализуем приложение-преобразователь, которое переводит элементы методов и практик на языке Essence Language в формате промежуточной модели в элементы модели описания в среде управления проектами.

Шаг 4. Осуществляем импорт метода и практик в среду управления проектами.

Шаг 5. В среде управления проектами можно начинать новый проект, в котором на основе модели выполнения автоматически (не изменяя привычной логики использования приложения) рабочие задачи исполнителям будут дополнятся дополнительными связями с элементами практик Essence (таким образом будет реализовываться алгоритм, описанный в § 1.3). Выполнение этих задач позволит собирать размеченные согласно правилам стандарта OMG Essence данные о достигнутом прогрессе, выраженном в изменениях состояний Альф, Суб-Альф и Рабочих продуктов.

В результате реализации, с одной стороны, получаем метод OMG Essence, имплементированный в целевую среду управления проектами: во-первых, в виде модели описания, которая фиксирует набор входящих в метод практик и отношения между ними; во-вторых, в виде модели выполнения, позволяющей команде использовать реализацию метода в реальном проекте при том, что все теоретические ограничения поддерживаются автоматически, а команда продолжает использовать привычное для нее окружение, посредством чего при трансляции процесса разработки в среды управления проектами снижается влияние человеческого фактора, о котором шла речь во вступлении к этой главе. С другой стороны, в процессе реализации методов и практик в реальных проектах формируются наборы размеченных данных, на основании которых можно разрабатывать различные математические методы[19] для систем поддержки принятия решений для менеджера проекта. Далее рассмотрим два примера реализации предложенного универсального метода в различных средах управления проектами.

[19] Созданию такого рода математической модели посвящена третья глава данной диссертации.

2.3 Пример механизма импорта практики в систему управления проектами Redmine

2.3.1 Краткая характеристика Redmine

В качестве первого примера реализации универсального метода переноса и интеграции методов, описанных на языке Essence Language, в реальную систему управления проектами рассмотрим Redmine [94]. Выбор Redmine обусловлен тем, что это открытое популярное серверное веб-приложение для управления проектами и задачами, которое имеет большие возможности расширения за счет механизма плагинов. Рассмотрение данной системы начнем с определения стека ее технологий и механизмов ее расширения.

Redmine реализована на основе фреймворка Ruby on Rails (RoR), написанного на языке программирования Ruby[20] [97]. Фреймворк реализует архитектурный шаблон Model-View-Controller для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером баз данных [98]. Фреймворк подразумевает использование модели предметной области[21] [98] как способ организации слоя бизнес-логики, источник данных которой по умолчанию реализован при помощи активной записи[22] [98], что накладывает определенные ограничения.

2.3.2 Архитектурные механизмы расширения Redmine

Redmine является гибкой системой, предоставляющей богатые возможности по расширению исходной функциональности на основе плагинов[23]. Для добавления плагина в систему необходимо скопировать его в специальную папку «plugin», осуществить миграцию базы данных, если это необходимо для устанавливаемого плагина, и перезапустить Redmine.

Благодаря особенностям языка Ruby и фреймворка Ruby on Rails можно с легкостью расширять ядро Redmine. Существуют возможности добавления новых методов к модели или контроллеру, а также оборачивания уже существующих. Также, Redmine реализует интерфейс для переопределения любых представлений системы.

---

[20] Ruby – динамический интерпретируемый объектно-ориентированный язык программирования, который обладает строгой динамической типизацией, сборщиком мусора и многими другими возможностями.

[21] Domain Model

[22] Active Record

[23] Плагин – дополнительный программный модуль, подключаемый к системе для расширения или использования ее возможностей.

Еще одном механизмом расширения Redmine являются хуки[24]. В Redmine хуки реализованы через механизмы, которые выполняют дополнительный код на фиксированный набор событий. Например, для создания элемента может существовать два хука: первый может быть определен до сохранения созданного элемента в базе данных, а второй – после. Redmine предоставляет конечный набор хуков для представлений, контроллеров и моделей.

2.3.3 Модель представления проектов в Redmine

Важным принципом этой работы является сохранение привычного для пользователя целевой системы управления проектами представления бизнес-процессов. Интеграция методов и практик, описанных на языке Essence, должна происходить максимально незаметно для пользователя системы. Таким образом, необходимо зафиксировать ключевые сущности модели Redmine, которые связаны с представлением проекта. Они рассмотрены ниже на диаграмме классов (рисунок 26).

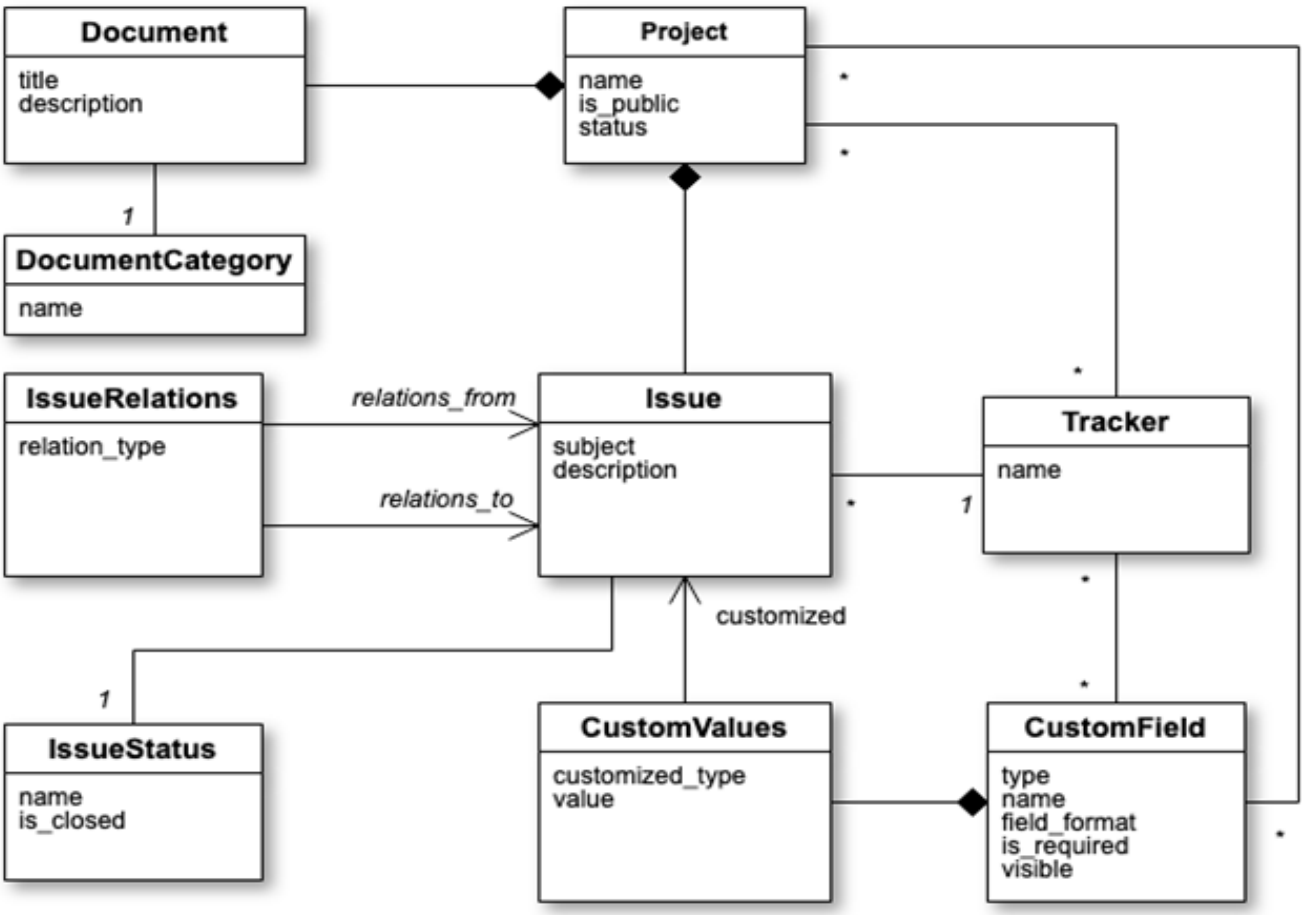


Рисунок 26 – Диаграмма классов ключевых элементов модели Redmine

[24] Хук (перехват) – технология, позволяющая изменить стандартное поведение тех или иных компонентов информационной системы.

Project (Проект) является основной сущностью системы. В рамках проектов ведется вся разработка программного обеспечения, хранятся задачи и рабочие артефакты.

Issue (Задача) представляет описание конкретного задания, которое необходимо выполнить в рамках проекта. Каждая задача относится к какому-либо трекеру (Tracker), определяющему тип задачи: дефект, улучшение, добавление нового функционала и т. п. Кроме того, каждая задача обязательно имеет текущий статус – например, «в работе» или «закрыта», выраженный при помощи класса IssueStatus.

CustomField (Дополнительное поле) представляет механизм расширения конфигурации задач, который привязан к конкретным трекерам и проектам. Механизм дополнительных полей позволяет добавить к задаче различные поля, которые можно заполнять при создании или обновлении задачи, а также хранить в них какую-либо метаинформацию. Механизм хранит указанные значения полей в CustomValues.

Document (Документ) – представляет некоторые документы и артефакты проекта. Позволяет загружать файлы или описывать их при помощи встроенного текстового редактора. Каждый документ относится к какой-либо категории[25] – например, «пользовательская документация» или «техническая документация».

2.3.4 Сохранение модели описания

Необходимо хранить модель описания в Redmine, поскольку она определяет связи между элементами и ограничения, которые необходимо поддерживать при использовании метода. Также модель описания позволяет иметь несколько методов и применять различные методы к разным проектам. Используем для этой цели структуру промежуточной модели, добавив в нее дополнительный класс MethodDefinition и указание «Definition» к большинству ее классов. Диаграмма классов хранения методов в Redmine изображена на рисунке 27.

[25] DocumentCategory

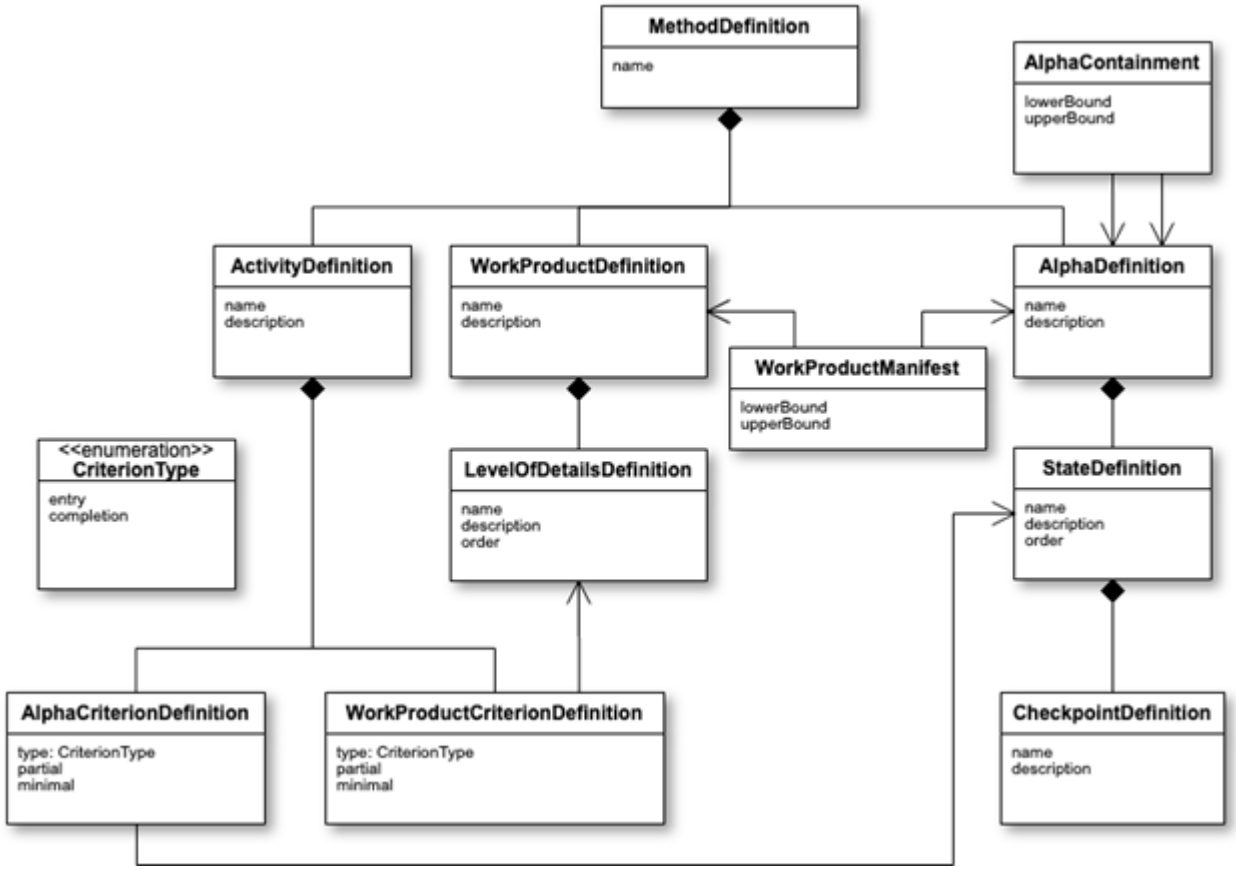


Рисунок 27 – Диаграмма классов хранения метода

После того, как хранение методов в Redmine описано, выделим элементы модели описания, для которых необходимо создавать элементы модели выполнения. Все элементы можно условно разбить на следующие группы в зависимости от взаимоотношения элементов:

1) «Метод» – включает MethodDefinition;

2) «Альфа» – включает AlphaDefinition, StateDefinition, CheckpointDefinition;

3) «Рабочий продукт» – включает WorkProductDefinition, LevelOfDetailsDefinition;

4) «Действие» – включает ActivityDefinition, AlphaCriterionDefinition, WorkProductCriterionDefinition;

5) «Прочие элементы» – включают AlphaContainment, WorkProductManifest.

Из выделенных групп необходимость в создании элементов модели выполнения существует для групп «Метод», «Альфа», «Рабочий продукт», «Действие». Поскольку оставшиеся AlphaContainment и WorkProductManifest используются в стандарте для фиксации структурированных сущностей, эти элементы не предполагаются в практической реализации и переносе на другой уровень. Далее рассмотрим создание модели выполнения.

2.3.5 Перенос метода в модель выполнения

Элемент MethodDefinition из модели описания представляет агрегирующую сущность для AlphaDefinition, WorkProductDefinition и ActivityDefinition. Поскольку в Redmine вся работа ведется в рамках проектов, выраженных классом Project, таким образом проекты в Redmine могут быть представлены в виде элементов модели выполнения для класса MethodDefinition. На рисунке 28 изображена диаграмма классов модели выполнения метода.

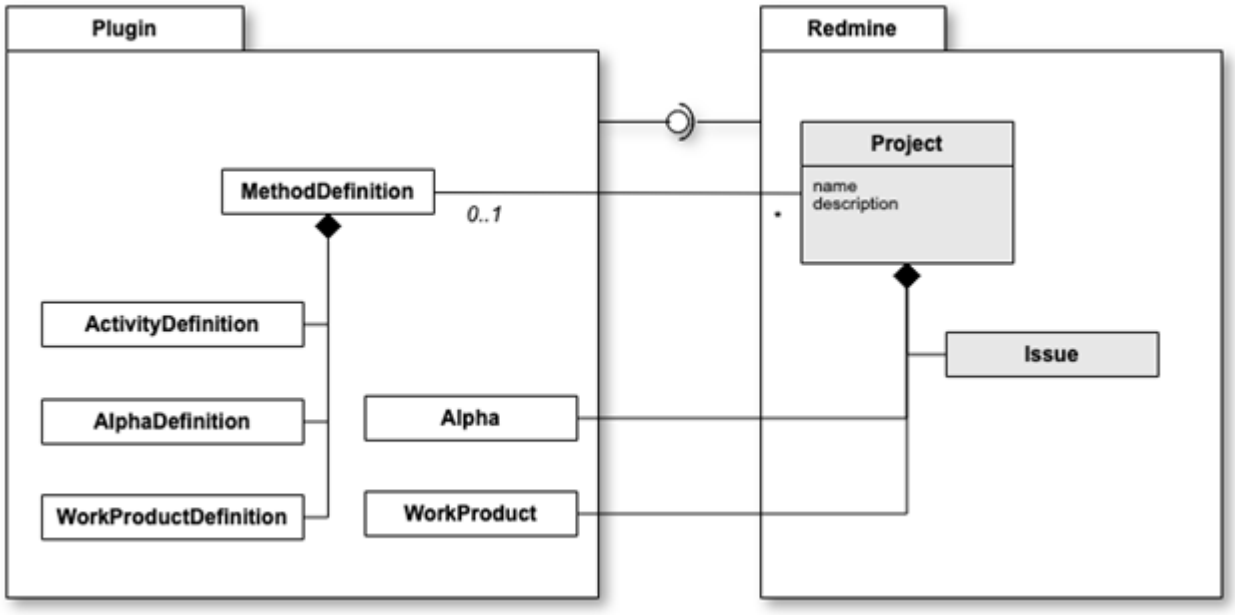


Рисунок 28 – Диаграмма классов модели выполнения для метода

С точки зрения реализации модели выполнения, каждый ее элемент обязательно должен иметь ссылку на свою модель описания для определения собственного типа и поддержания ограничений целостности, поэтому существует связь между классами MethodDefinition и Project. Поскольку класс Project является частью Redmine, для создания связи необходимо использовать механизм расширения модели Redmine.

Подобные связи между классом, представляющим элемент модели описания, и классом, представляющим элемент модели выполнения, будут существовать и для других элементов, рассматриваемых ниже.

2.3.6 Перенос альфы в модель выполнения

В контексте систем управления проектами альфа должна реализоваться как некая сущность, которая хранит свое текущее состояние и агрегирует связанные с ней рабочие продукты и суб-Альфы. Эта сущность должна иметь возможность демонстрировать все свои возможные состояния

с их набором контрольных пунктов, выполнение которых переводит альфу в это состояние при условии, что все предыдущие состояния достигнуты. Также необходимо учитывать ограничения, которые накладывают на любой метод Essence семь уникальных альф, определенных в рамках ядра. Эти ограничения необходимо поддерживать в модели выполнения. Итоговая структура данных, используемая для реализации переноса логики работы Alpha, приведена на рисунке 29.

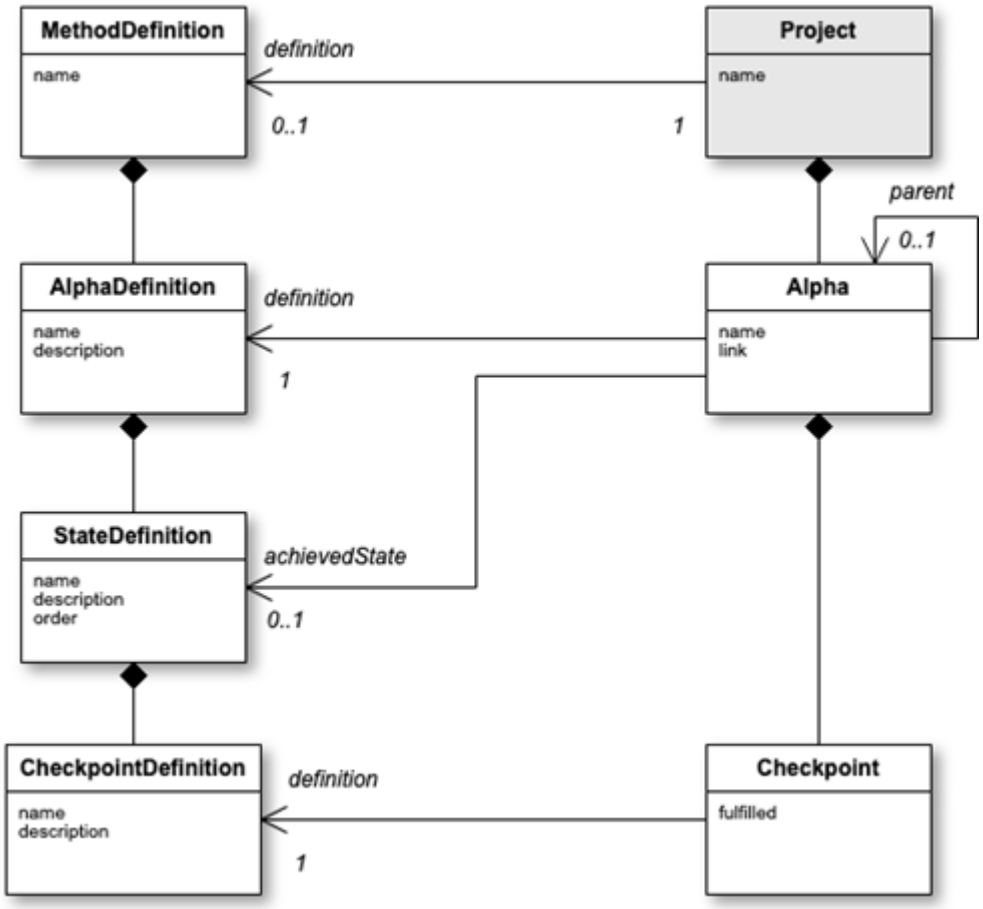


Рисунок 29 – Диаграмма классов модели выполнения для альфы

В модели Redmine не удалось выделить сущность, на основе которой можно реализовать все особенности альф, поэтому представление альф необходимо было добавить в текущую модель. Модель описания альфы состоит из трех классов: AlphaDefinition, StateDefinition, CheckpointDefinition. Для реализации модели выполнения достаточно будет двух классов: Alpha и Checkpoint. Класс StateDefinition не реализуется в модели выполнения, поскольку представляет абстрактную группирующую сущность для контрольных пунктов, а для описания состояния альфы достаточно хранить ссылку на объект класса StateDefinition. Однако, для определения необходимого объекта класса StateDefinition следует хранить состояние элементов контрольного списка. Это обстоятельство привело к созданию класса Checkpoint.

Класс Alpha содержит поле для хранения своего имени в рамках проекта и поле, позволяющее указать ссылку на какой-либо документ, если это необходимо. Для упрощения поиска родительской альфы и суб-Альф, классу Альфа добавлена связь со своим родителем, что создает возможность отделять основные альфы от суб-Альф: основные альфы не имеют связи с родительской альфой.

2.3.7 Перенос рабочего продукта в модель выполнения

Рабочим продуктом может быть некий документ или часть программного кода, которые зачастую хранятся в разных местах. Например, описание требований может храниться в облачном хранилище, а программный код – в системе контроля версий. Поэтому в рамках Redmine рабочий продукт будет объект, для которого необходимо указать ссылку на конкретный файл или набор файлов, представляющих указанный рабочий продукт. При этом данный элемент должен иметь возможность хранить ссылку на текущий уровень описания (LevelOfDetails) рабочего продукта.

Подобное описание имеет класс Document из модели Redmine. Но, к сожалению, данный элемент не подходит по следующей причине: элемент модели выполнения рабочего продукта обязательно должен иметь связь с элементом модели описания, т. е. при создании элемента класса Document пользователь должен выбрать, к какому рабочему продукту из метода он принадлежит. Это можно было бы сделать на основе категорий документа, если бы они относились к проекту, а не к системе управления в целом, но существует проблема в том, что необходимо иметь отношение между DocumentCategory и WorkProductDefinition. Эту связь можно было бы реализовать по имени, т. к. в рамках метода все имена рабочих продуктов уникальны, но, поскольку категории документов создаются для всех проектов в Redmine, при импорте двух методов, содержащих рабочие продукты с одинаковыми названиями, получится конфликт.

В связи с данной проблемой было решено реализовать рабочие продукты с помощью добавления новых сущностей в Redmine. Структура классов для этого приведена на рисунке 30.

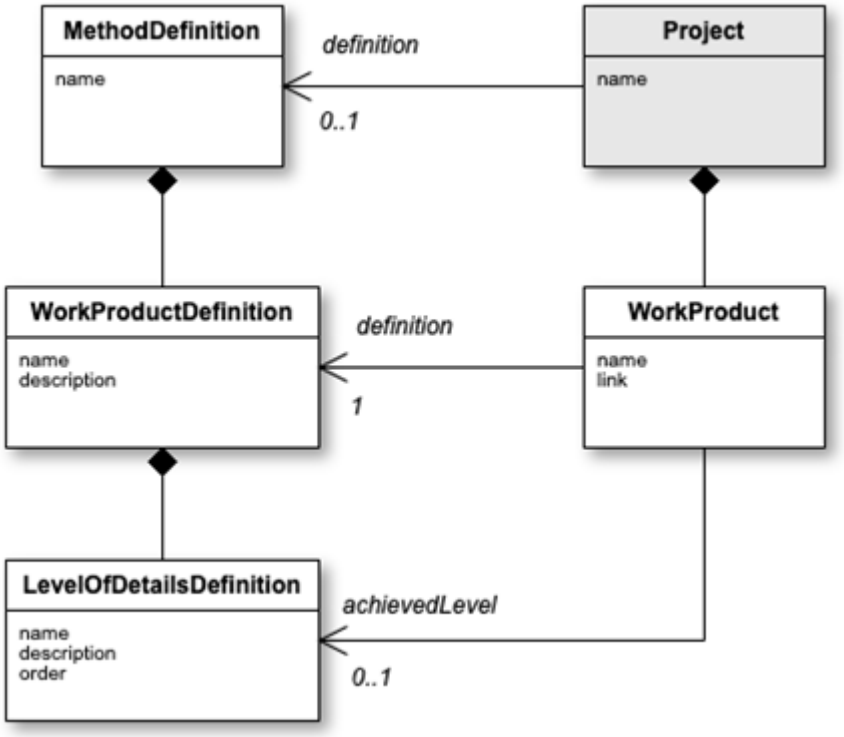


Рисунок 30 – Диаграмма классов модели выполнения для рабочего продукта

Описание рабочего продукта состоит из двух классов: WorkProductDefinition и LevelOfDetailsDefinition, но поскольку рабочему продукту достаточно хранить ссылку на достигнутый уровень детализации, LevelOfDetailsDefinition не реализуется в модели выполнения.

2.3.8 Перенос активности в модель выполнения

На практическом уровне объект класса Активность[26] – это обычная задача, которая ставится исполнителю в рамках проекта по разработке программного обеспечения, с добавлением специфических входных и выходных параметров. В Redmine задачи представляют собой объекты класса Issue, к которым необходимо добавить возможность добавлять входные и выходные данные – элементы классов Alpha и WorkProduct из модели выполнения.

Для задач в Redmine существует собственный встроенный механизм расширения на основе дополнительных полей. Но данный механизм нам не подходит, поскольку у него есть существенное ограничение: элементы списка дополнительного поля должны быть указаны при создании. В нашем же случае необходимо сгенерировать список на основании элементов, хранящихся в базе данных. При этом было бы необходимо связывать дополнительные поля с объектами классов

[26] Активность – класс задач, для выполнения элементов которого необходимы определенные входные данные, а завершение задачи приводит к изменению некоторого набора выходных данных.

AlphaCriterionDefinition или WorkProductCriterionDefinition, что затруднительно, поскольку к дополнительному полю нельзя добавить скрытые метаданные для осуществления этой связи. Следовательно, добавление выходных и выходных данных должно реализоваться с использованием дополнительных сущностей.

Кроме того, каждая задача должна быть связана со своей моделью описания ActivityDefinition. Для реализации данной связи был рассмотрен механизм трекеров: на основе действий метода можно было бы создать набор трекеров и подключать их для необходимого проекта, а ссылка на нужный элемент ActivityDefinition хранилась бы в дополнительном скрытом числовом поле со значением, равным по умолчанию идентификатору элемента ActivityDefinition. Однако в Redmine есть требование к уникальности имен трекеров, поэтому разные методы не могут содержать элемент модели описания действия с одинаковым названием. В связи с чем было принято решение воспользоваться возможностью расширения ядра Redmine и однозначно определить связь между классом Issue и ActivityDefinition.

В результате в Redmine были добавлены два дополнительных класса: WorkProductCriterion и AlphaCriterion, которые определяют тернарную связь между задачей, критерием и входным элементом[27]. Итоговая диаграмма добавления модели выполнения для Активности представлена на рисунке 31.

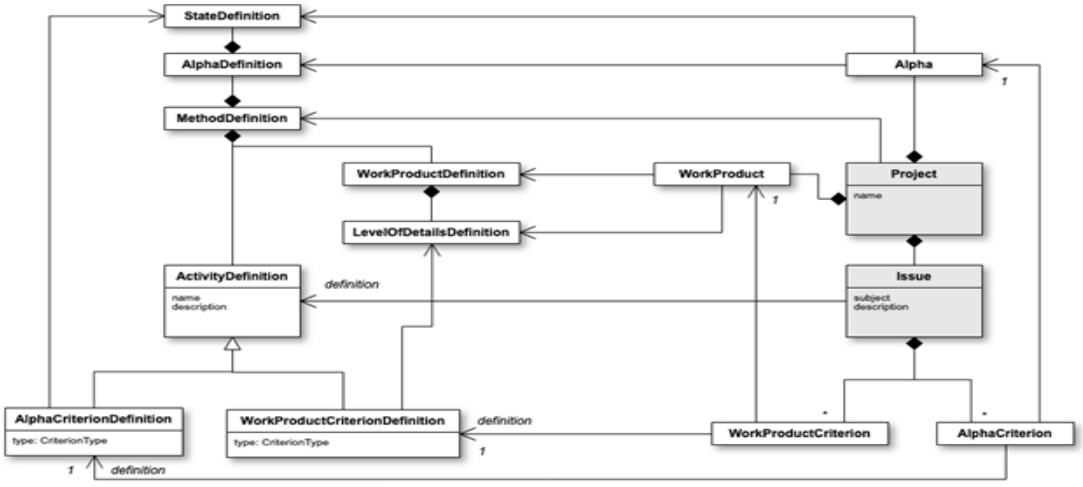


Рисунок 31 – Диаграмма классов модели выполнения для действия

[27] Этим элементом может быть Alpha или WorkProduct.

2.3.9 Реализация

Добавление метода в Redmine происходит с использованием файла формата *.json*. Данный файл должен содержать описание метода в структурированном виде при помощи элементов промежуточной модели. При добавлении нового метода в систему, необходимо указать его название, которое должно быть уникальным среди уже существующих методов Redmine. Пример реализации приведен на рисунке 32.

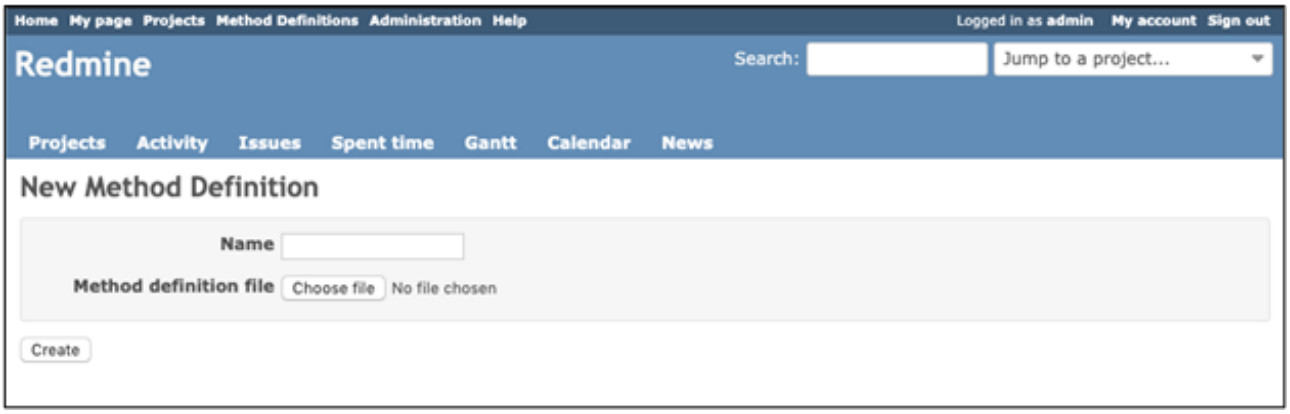


Рисунок 32 – Скриншот формы добавления метода

Алгоритм сохранения нового метода в Redmine основан на организации хранения данных в файле с описанием метода. В связи с тем, что все данные хранятся в структурированном виде, для сохранения нового метода необходимо получить список, описывающий элементы некоторого класса промежуточной модели, и в цикле создавать элементы модели описания на основе элементов полученного списка. При этом необходимо сохранять идентификаторы созданных элементов для восстановления связей, описанных в файле с методом. Кроме того, имеет значение порядок, в котором создаются элементы модели описания, потому что при сохранении элементов в базу данных, которые ссылаются на другие элементы, должно выполняться ограничение на внешние ключи. Данный подход к добавлению нового метода изображен на рисунке 33 в виде диаграммы деятельности, а на рисунке 34 приведен скриншот его реализации.

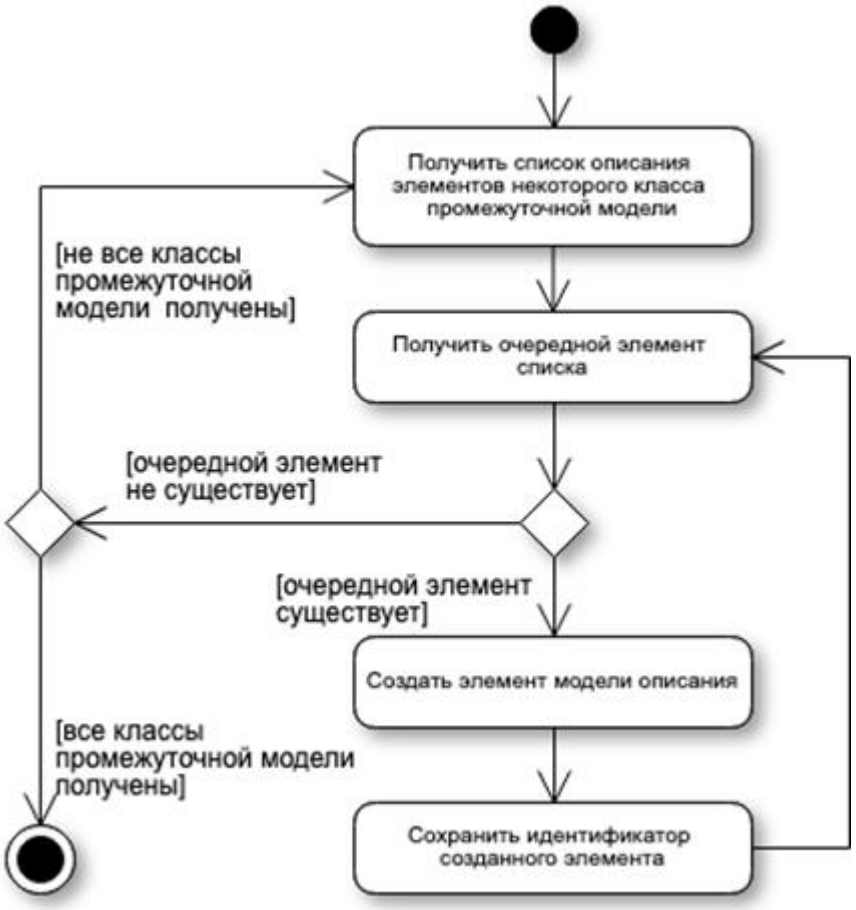


Рисунок 33 – Диаграмма деятельности добавления нового метода

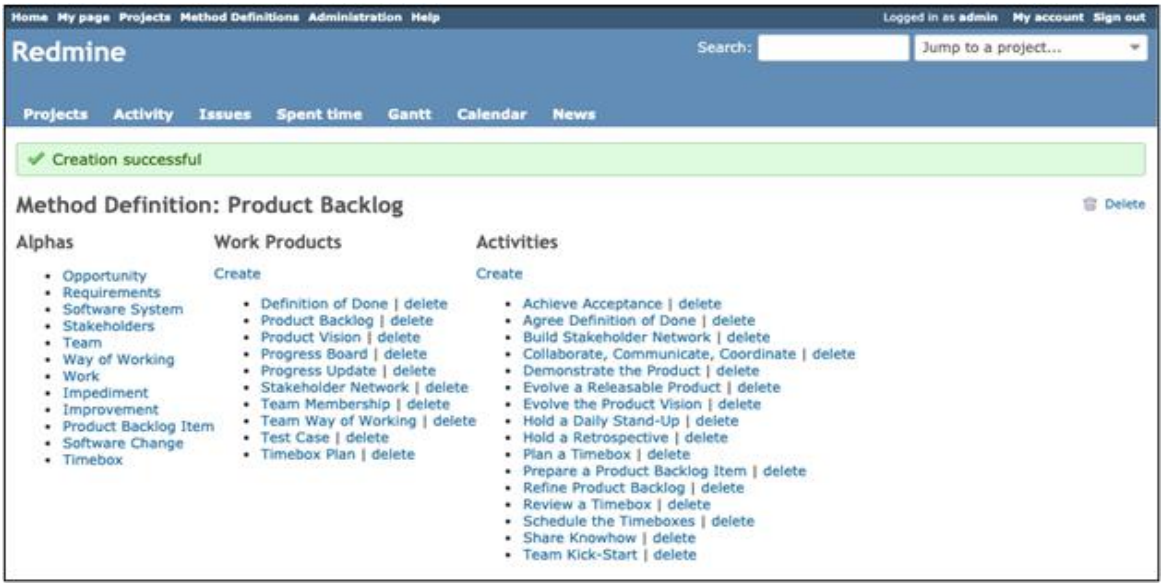


Рисунок 34 – Скриншот добавленного метода

Когда в Redmine существует добавленный в систему метод, его можно подключить к существующему проекту. На этом этапе необходимо создавать элементы модели выполнения

на основе описания подключаемого метода. Метод считается подключенным к проекту после выполнения следующих шагов:

1) добавить связь между проектом[28] и описанием метода[29];

2) создать основные альфы;

3) создать связанные с основными альфами рабочие продукты в минимальном количестве, указанном в WorkProductManifest;

4) создать суб-Альфы в минимальном количестве, указанном в AlphaContainment.

При этом необходимо иметь в виду, что альфа – это рекурсивная структура, которая содержит принадлежащие ей рабочие продукты, суб-Альфы и список чекпоинтов. Например, на рисунке 35 представлена диаграмма объектов, отображающая основную альфу «Работа»[30] метода «Agile Essentials».

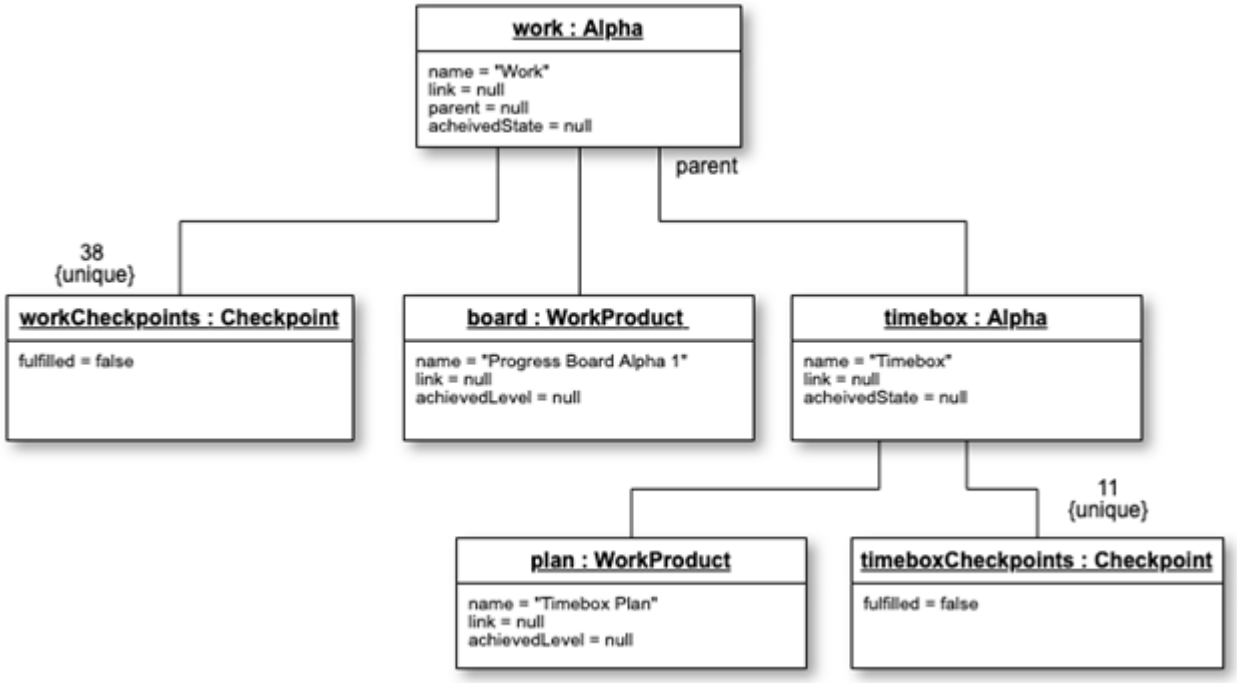


Рисунок 35 – Диаграмма объектов основной альфы «Работа»

Таким образом, при создании альфы должны создаваться принадлежащие ей элементы. Следовательно, для подключения метода необходимо вызвать только создание основных альф и добавить связь между проектом и описанием метода.

[28] Класс Project.
[29] Класс MethodDefinition.
[30] Work

Процесс создания альфы представлен на диаграмме последовательности, изображенной на рисунке 36. Пример подключения основных альф приведен на рисунке 37, детальное описание основной альфы – на рисунке 38.

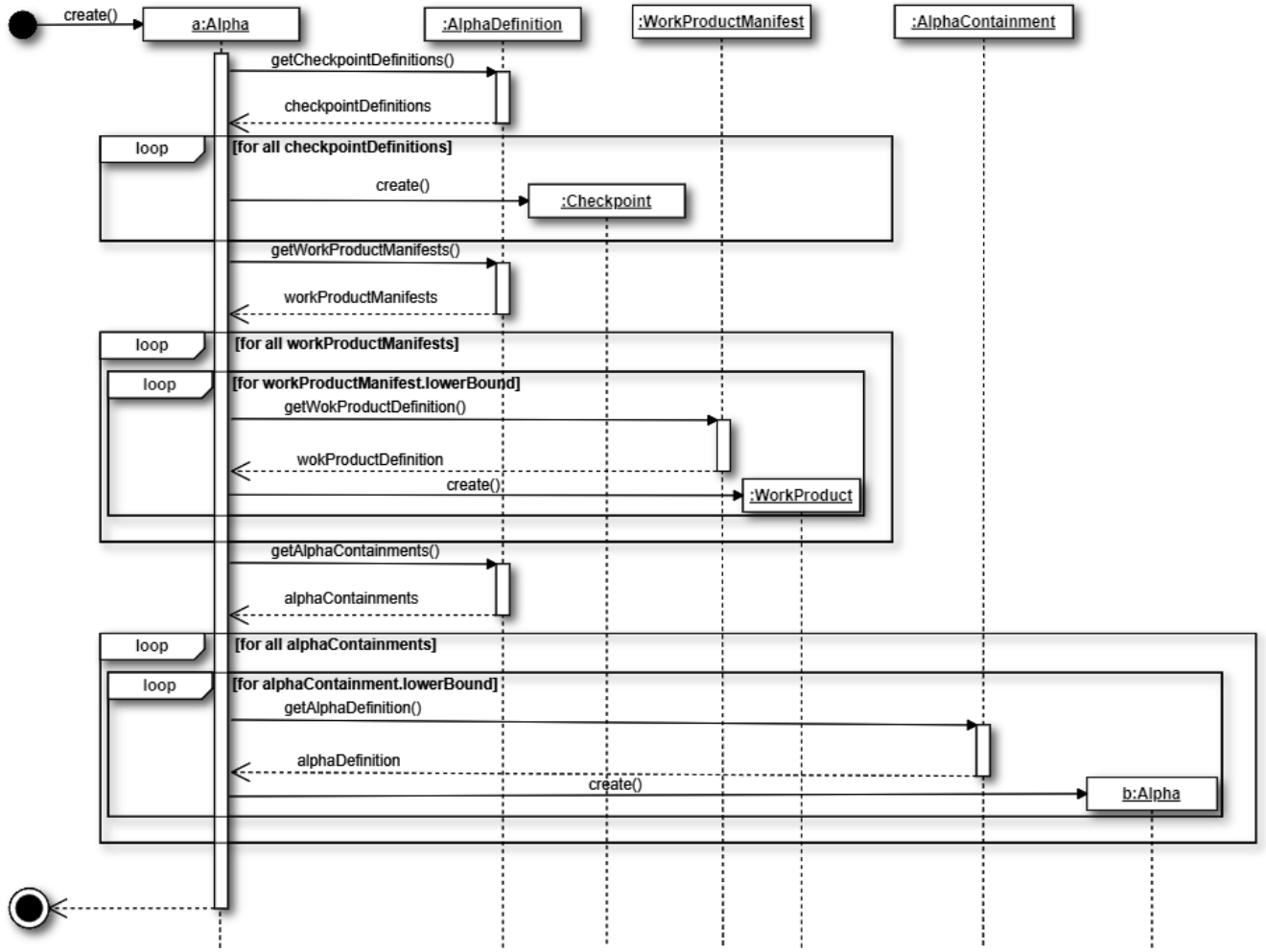


Рисунок 36 – Диаграмма последовательности создания альфы

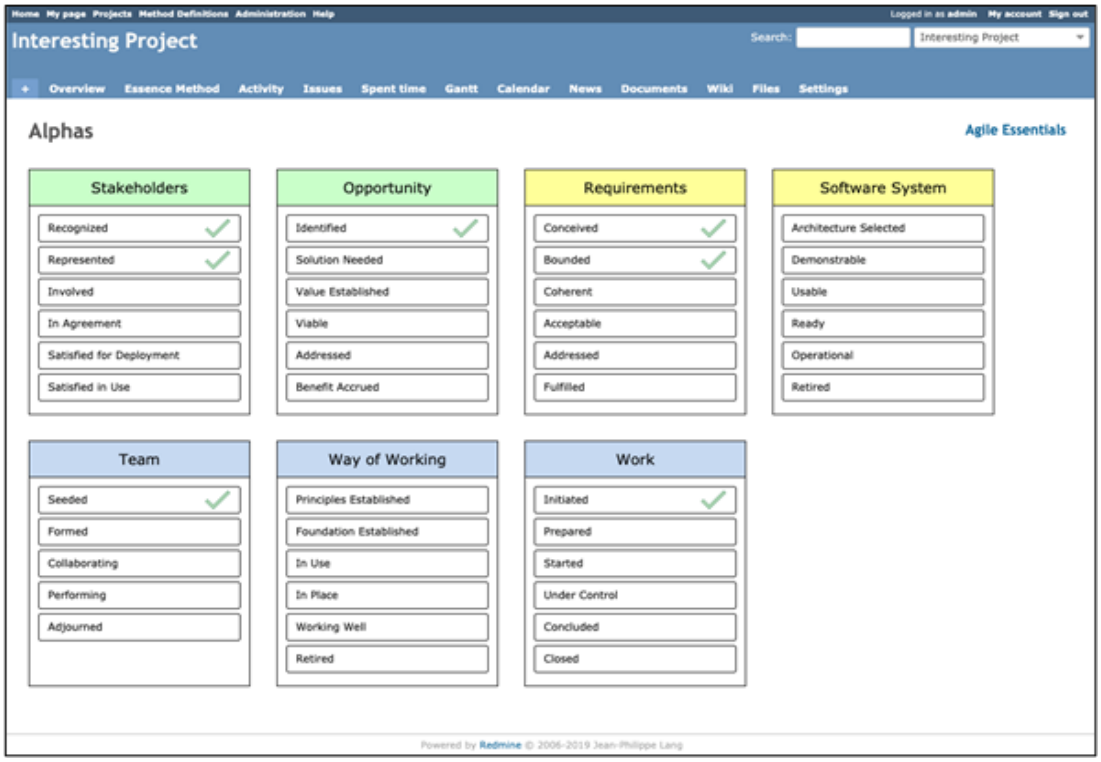


Рисунок 37 – Скриншот основных альф подключенного метода

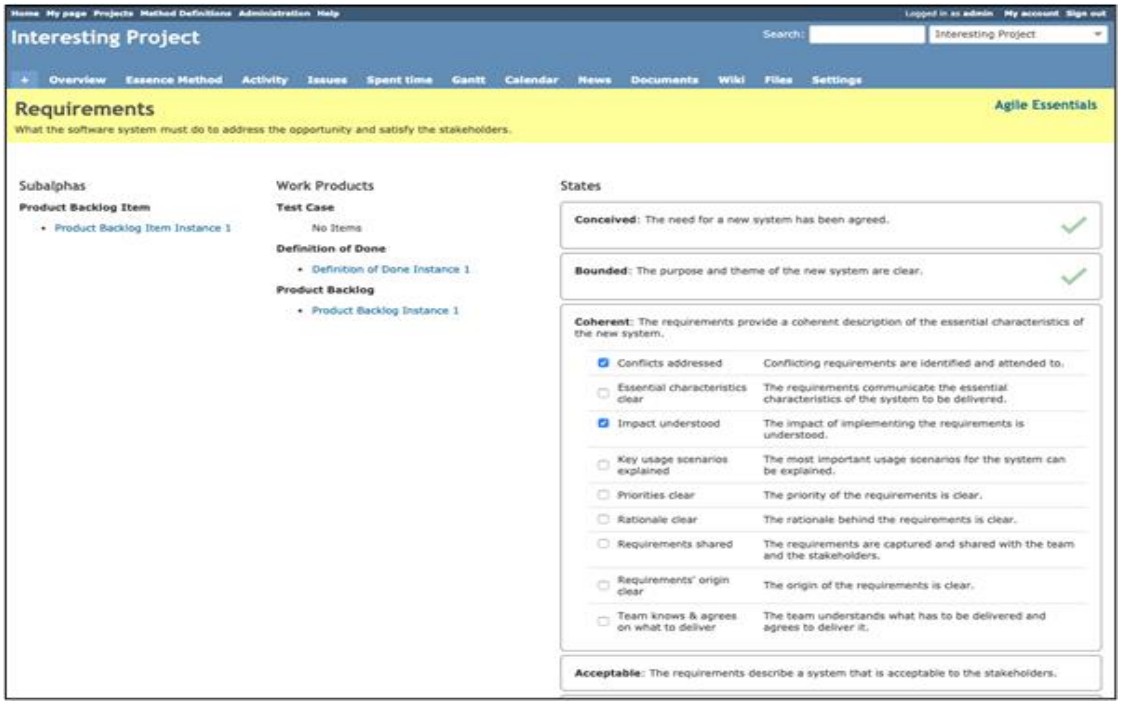


Рисунок 38 – Скриншот альфы «Требования» в подключенном методе

Создание действия основано на механизме задач Redmine. Следовательно, необходимо расширить форму создания задачи, в которой можно будет выбрать создаваемое действие и указать входные данные для него. Для реализации этого воспользуемся таким механизмом расширения

представлений Redmine, как хук. Система управления проектами содержит хук под названием «view_issues_form_details_bottom», который позволяет добавить часть представления в конец формы и получить элементы ее контекста, такие как объект формы и модель создаваемой задачи. При помощи данного контекста можно встроить элемент выпадающего списка доступных действий метода, выбранное значение которого автоматически будет привязано к объекту создаваемой задачи.

Добавляемая часть представления должна быть динамической[31] формой, которая содержит выпадающие списки для указания действия и его выходных критериев. Другими словами, необходимо генерировать выпадающие списки, соответствующие критериям конкретного действия при его изменении. Для реализации описанного поведения необходимо использовать технику AJAX[32] [97]. Следовательно, при изменении пользователем действия будет посылаться AJAX-запрос к серверу, который будет возвращать представление, содержащее выпадающие списки для необходимых критериев. На рисунке 39 изображена диаграмма последовательности, отображающая данное поведение, а на рисунке 40 представлен скришнот страницы создания задачи с реализованным механизмом.

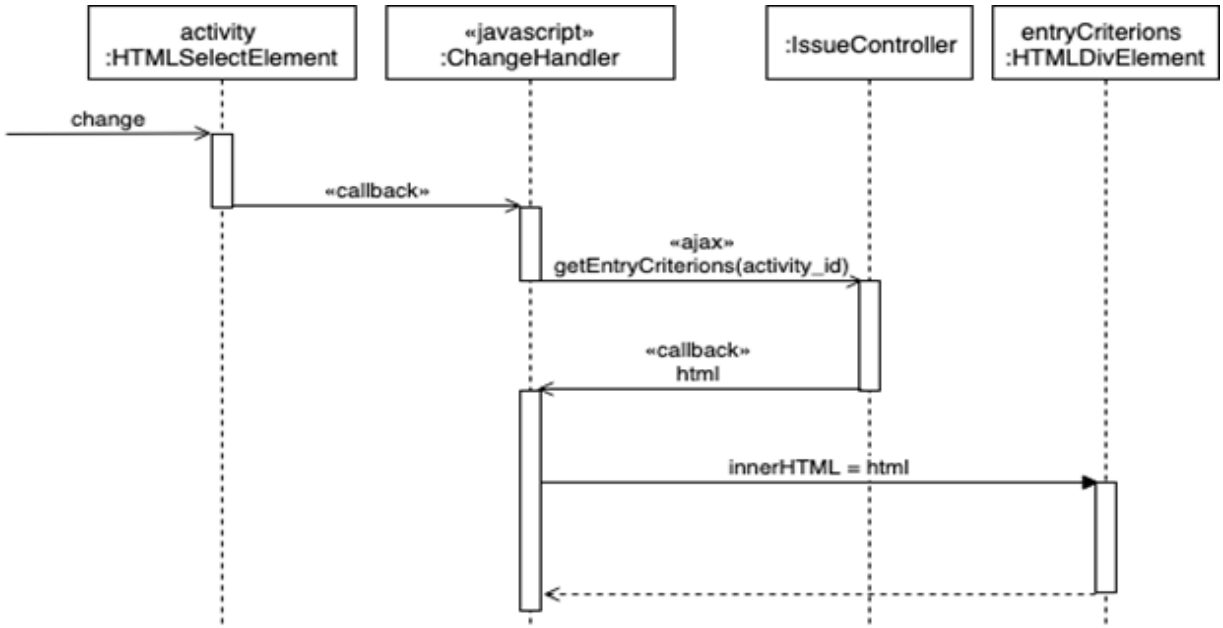


Рисунок 39 – Диаграмма последовательности изменения списка входных критериев

[31] Форма должна быть динамической, поскольку количество и тип критериев зависят от выбранного действия.

[32] AJAX (Asynchronous Javascript And Xml) – подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером.

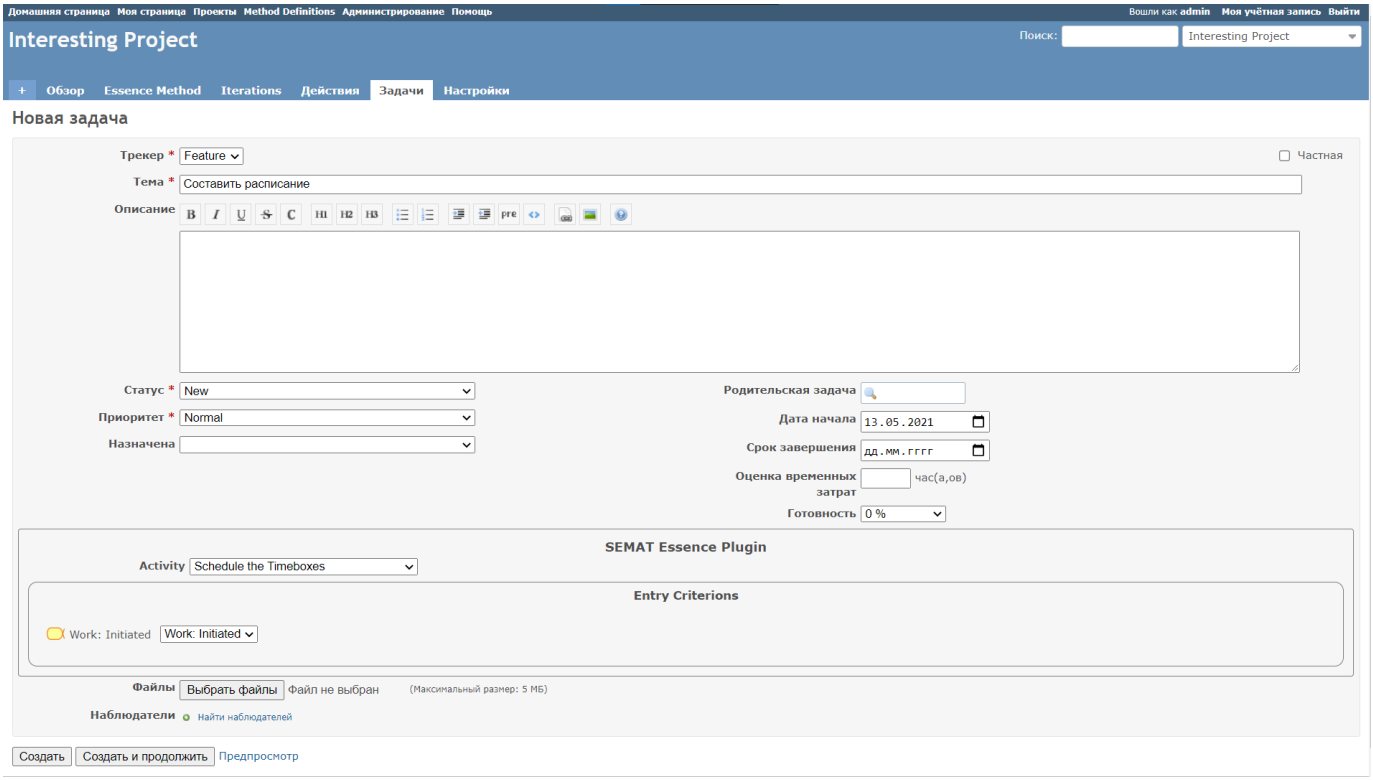


Рисунок 40 – Скриншот создания действия в виде задачи

Отслеживание закрытия действия основано на изменении обязательного поля любой задачи – статус (Status). Администратор может создавать любое количество различных статусов к задачам и определять, при каких статусах задача считается закрытой. Данная информация хранится в базе данных, и необходимо определять, выбрал ли пользователь статус, который считается закрывающим или нет. Если пользователь выбрал закрывающий статус, то необходимо получить с сервера критерии завершения действия и предоставить пользователю поля для их заполнения.

Для реализации необходимого поведения используется язык сценариев браузера – JavaScript. С его помощью добавлена функция обратного вызова (callback), которая выполняется при изменении значения состояния задачи. Следующим шагом после получения уведомления об изменении состояния идет проверка того, является ли выбранное состояние закрывающим. Для получения данной информации необходимо сделать запрос на сервер. В случае, если состояние является закрывающим, в ответе необходимо «отправить представление, содержащее списки критериев завершения с их возможными значениями». Данный подход изображен в виде диаграммы деятельности на рисунке 41, а на рисунке 42 представлен скриншот, демонстрирующий его реализацию.

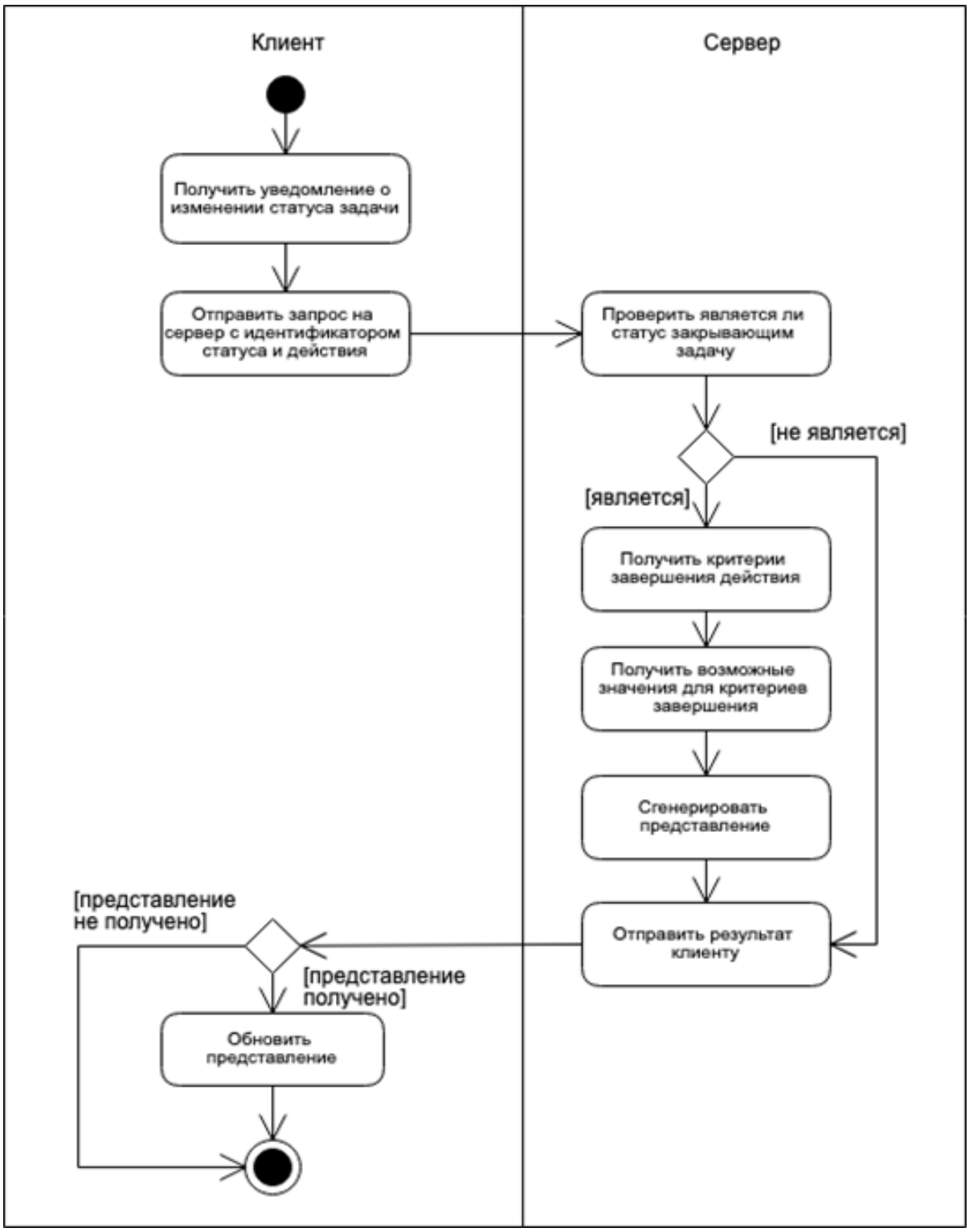


Рисунок 41 – Диаграмма деятельности получения представления критериев завершения

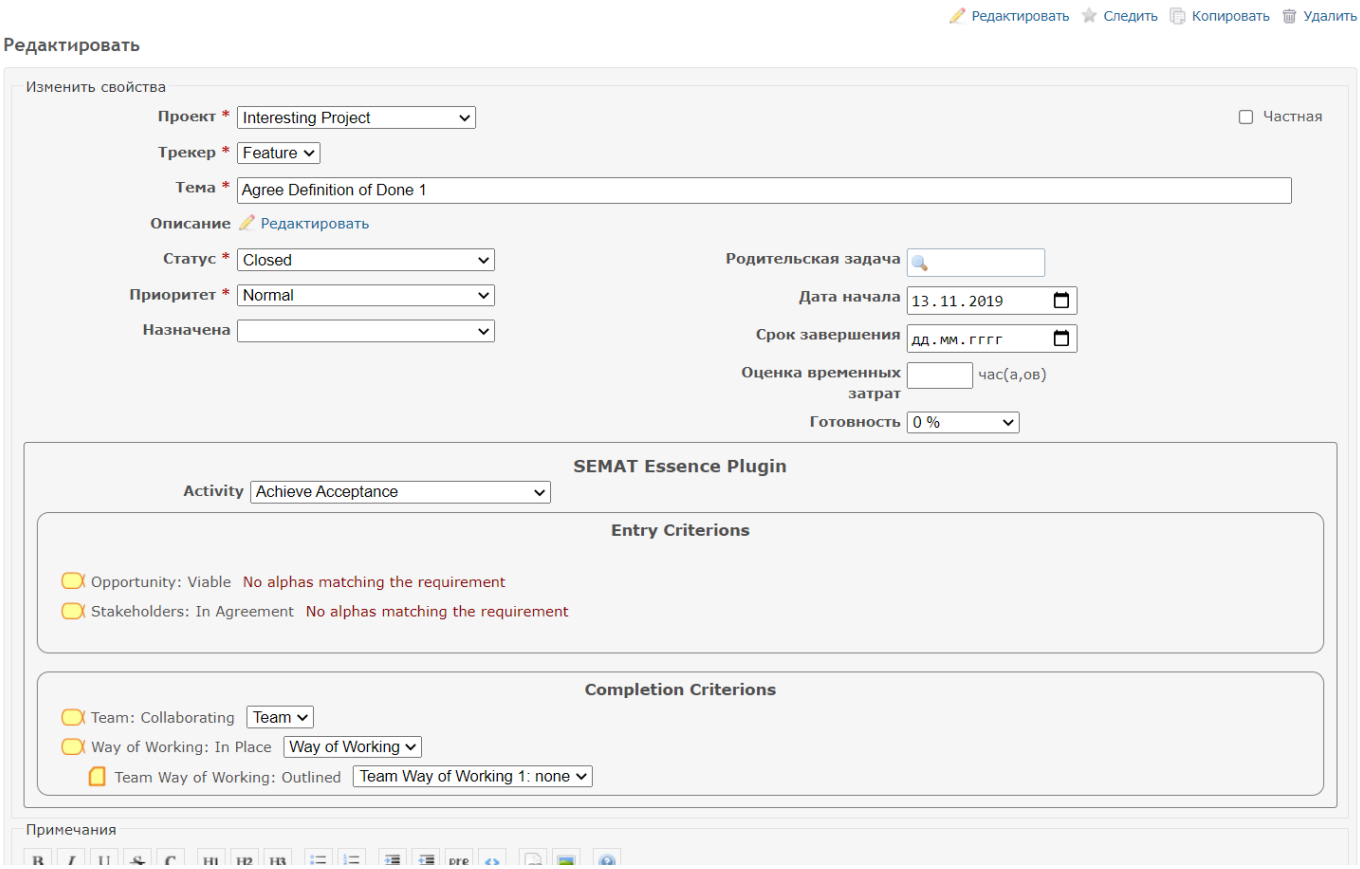


Рисунок 42 – Скриншот формы закрытия действия

После того, как пользователь нажмет кнопку сохранения изменений в браузере, необходимо обработать отправленные данные на сервере. Для этого расширим в процесс обновления задачи при помощи хука «controller_issues_edit_after_save». Данный хук выполняется после сохранения изменений атрибутов задачи в базе данных . Это означает, что модель задачи прошла валидацию, и можно создавать элементы модели выполнения для критериев завершения на основании предоставленных формой данных, если модель задачи содержит состояние, определяющее ее как закрытую.

Таким образом, задача инсталляции моделей описания и выполнения OMG Essence в реальную среду управления проектами Redmine полностью решена. Другой пример переноса в среду Azure DevOps Server подробно рассмотрен в соответствующей публикации [89], который не приводится в данной работе.

Выводы главы

Описанный механизм импорта практик Essence и состоящих из них методов в целом позволяет решить задачу автоматизации конфигурирования среды управления проектами под набор практик, которые собирается использовать проектная команда, однако даже наличие подобного решения не исключает ряд потенциальных трудностей и проблем, связанных с полноценным использованием предложенного подхода.

Во-первых, описанный метод и подход основаны на экспорте данных из проприетарной системы «Practice Workbench», и, к сожалению, на момент написания данной работы не существует альтернативных решений, которые позволили бы описывать практики Essence в машиночитабельном виде. Другими словами, фактически для прикладного развития данной работы необходимо чтобы появился или альтернативный инструмент со свободным доступом для описания практик на языке Essence, или хотя бы общий репозиторий со всеми практиками, которые могут описывать узконаправленные специалисты.

Во-вторых, в работе приведена одна среда управления проектами, однако, подобных систем достаточном много, и некоторые особенности принципов их функционирования различаются. В результате несмотря на то, что если соблюдать структуру промежуточной модели для сбора данных и анализа, теоретически можно разработать некоторые общие паттерны подобных расширений, задача разработки расширений систем управления проектами для импорта практик Essence пока все равно должна решаться индивидуально для каждой системы, причем не только на уровне имплементации решения, но также и на уровне проектирования архитектуры такого решения, хотя некоторые системы управления проектами и имеют схожие принципы организации бизнес-логики.

В-третьих, несмотря на существенный положительный эффект от накопления структурированных данных по использованию различных практик в проектах по разработке программного обеспечения, в краткосрочной перспективе пользу от этого будет, в первую очередь, получать академическое сообщество и только в долгосрочной – профессиональное, но при этом даже уменьшенные ресурсные затраты будет нести профессиональное сообщество. Иными словами, для профессионального сообщества просто автоматизированная конфигурация среды управления проектами – важная задача только в тех случаях, когда есть необходимость / желание опробовать новую практику, уменьшить затраты на изучение и конфигурирование, что само по себе достаточно

специфичный прецедент, который маловероятно приведет к широкому распространению подобных инструментов.[33]

Исходя из вышесказанного, можно сделать неутешительный вывод, что несмотря на весь прогресс и признание Essence сообществом – с одной стороны, и появление работ наподобие данной, решающих связанные практико-ориентированные задачи, – с другой, пока выгода для профессионального сообщества неочевидна и недостаточна, чтобы использование формальных подходов к описанию и анализу собственных практик разработки стало общепринятой нормой. По сути, для того чтобы компании по разработке программного обеспечения стали активнее внедрять Essence как правило работы в свои собственные процессы, необходимо продемонстрировать им, какую выгоду можно получить, используя даже отдельные части Essence, что рассмотрено в следующей главе.

Основные результаты, изложенные в данной главе, опубликованы в научной периодике [89, 90]. Рассматриваемый метод представлен автором работы в виде файла формата *.json* [93].

[33] Здесь можно отметить, что такой инструмент очень удачно применим в образовательном процессе по подготовке программных инженеров.

3. ПРИКЛАДНЫЕ МАТЕМАТИЧЕСКИЕ МЕТОДЫ ДЛЯ УПРАВЛЕНЧЕСКИХ РЕШЕНИЙ В ПРОЕКТАХ ПО РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В контексте данной работы необходимо указать сборник, вышедший в 1996 г., в котором детально разобраны различные задачи, примеры и варианты применения классических прикладных статистических методов к вопросам программной инженерии, в частности принятию решений в процессе выполнения проектов по разработке программного обеспечения [98]. В сборнике приводятся варианты использования различных оценок и статистических подходов на разных этапах разработки, в компаниях различного уровня и масштаба. Важно отметить, что результаты, представленных в этом сборнике работ, достаточно пессимистичны для дальнейшего развития прикладных методов. Фактически сделаны выводы о том, что распространенные и общепринятые методы прикладной статистики плохо применимы к отрасли программной инженерии. Напомним основные предположения авторов сборника, объясняющие полученные результаты.

Во-первых, несмотря на то, что существуют общепринятые классы программных систем, которые широко распространены в профессиональном сообществе для краткой характеристики основных особенностей системы, непосредственно проекты по разработке таких систем далеко не всегда совпадают по своей структуре и последовательности выполняемых работ. На практике это означает, что две компании, разрабатывающие системы из одного и того же класса – например, ERP системы – могут совершенно по-разному планировать и распределение обязанностей в рамках своих команд. В результате методы, для которых обучающей выборкой будут данные одной команды, могут давать противоречивые результаты для другой. При этом даже одна и та же команда может разрабатывать схожие по своему функциональному наполнению системы по-разному. Таким образом, обучающая выборка не всегда применима даже внутри одной компании. Эта проблема имеет достаточно глубокий характер, и ее состояние практически не изменилось за последние 20 лет.

Во-вторых, если пытаться оценивать характеристики или спецификации программных систем для дальнейшего анализа прогресса их разработки, то очень быстро выясняется, что для большинства современных программных систем требования или спецификации не пишутся на строгом и формальном языке, который позволял бы построить корректную математическую модель требований. Несмотря на то, что с момента выхода работы 1996 г. появилось несколько различных стандартов и подходов к описанию требований, на уровне практики ситуация по этому вопросу

значительно не изменилась, поскольку даже самые формальные из них допускают необходимость использования нестрогих высказываний для описания требований к программным продуктам. При этом даже использование самых формализованных стандартов описания характеристик не приносит значительного эффекта, т. к. они описывают ожидаемый от программной системы результат, а основная задача в ходе разработки – понять, как его достичь при реальных ограничениях.

В-третьих, один из альтернативных вариантов, предложенный в рассматриваемой работе, предполагал измерение и анализ производительности программистов в количестве написанных ими строк исходного кода программы. Однако в этой же части сборника обосновывалось, что данный подход практически бесполезен как для оценки эффективности разработчиков, так и при попытках прогнозирования производительности или оценки временных затрат на разработку программного продукта. Сейчас, как и тогда, разработка сводится к написанию исходного кода программного продукта только на самом элементарном уровне, основная же сложность – прийти к пониманию того, какой именно программный код необходимо написать, что не всегда очевидно.

Наконец, обсуждалась проблема сбора корректных данных из проектов по разработке программного обеспечения. В рассматриваемой работе проводилась параллель с классическими инженерными дисциплинами: зачастую прикладные статистические методы применяются на этапе производства или создания продукта или изделия. При этом помимо оценки человеческого фактора можно рассчитать более объективные показатели – достигнутый измеряемый прогресс, время простоя конвейера, расход материала и проч. В этой аналогии программная инженерия сопоставляется к конструкторским бюро: фактически работам, связанным с первичным проектированием и прототипированием будущего изделия, где как раз прикладные статистические методы не применяются ввиду нестабильности модели – значительной доли творчества в подобной деятельности. В сущности, для получения каких-либо характеристик или данных в программной инженерии всех исполнителей необходимо помещать в жестко контролируемые условия, что зачастую вызывает негативную реакцию разработчиков и влияет на получаемые данные. Стоит отметить, что за прошедшие 20 лет средства контроля за работой разработчика сильно продвинулись вперед, и вызывают уже не такой большой негатив, однако большинство компаний их не используют, поскольку достижимая выгода не совсем ясна.

Опубликованная в 1996 г. работа наглядно демонстрирует основные причины некоторого застоя в вопросах развития прикладных математических методов в области программной инженерии в целом и управления проектами по разработке программных продуктов, в частности.

Третья глава посвящена построению альтернативной математической модели для решения реальных практических задач на основе формального описания метода разработки на языке OMG Essence. В § 3.1 кратко описаны результаты, полученные при попытке использовать общепринятые методы на основе другой выборки экспериментальных данных. К сожалению, выводы практически повторяют результаты 1996 г. В § 3.2. Даётся описание, какими характеристиками должен обладать практически применимый прикладной математический метод для управления проектами по разработке программного обеспечения. В § 3.3 предлагается использовать байесовскую сеть и решается вопрос о построении сети на основе общего описания теоретического контура стандарта OMG Essence. В § 3.4 приведен один из вариантов применения полученной байесовской сети для помощи в управлении проектами по разработке программного обеспечения. Для построения байесовских сетей использовалось программное обеспечение GeNIe, для которого была получена академическая лицензия.

3.1. Методы статистической программной инженерии

В ходе выполнения данной работы на начальных этапах исследования была рассмотрена возможность использования общепринятых методов [99], в т. ч. методов с небольшой модификацией [100], для принятия управленческих решений в проектах по разработке программного обеспечения, но вместо используемых в статистической программной инженерии формальных характеристик программного продукта или оценки количества строк кода предлагается использовать самооценку разработчиков – фактически, соотношение того, как разработчик оценивает сложность получаемой им задачи в человеко-часах, и того, сколько человеко-часов затрачено этим разработчиком на ее исполнение в итоге. Для апробирования такого подхода были использованы данные, предоставленные компаниями «BitWorks» и «DevGuild». Кратко сформулируем выводы, полученные из проведенных работ.

1. Из ряда экспериментов наиболее перспективный результат обнаружен у регрессии оценок разработчиком полученных им задач. Однако положительный результат оказался достаточно специфичным: общая оценка набора задач становилась точнее, при этом оценка отдельных задач ухудшилась (разница по модулю между фактическим итоговым значением и значением, полученным по непараметрической регрессии, становилась хуже). На практике данный подход применим, только если у разработчика есть детальный план по задачам, рассчитанный на несколько

полных человеко-недель по общему объему оценок времени, которые благодаря полученной регрессии можно улучшить.

2. Другим интересным результатом оказалось следствие из проверки однородности – получение сдвига[34] при оценке разработчиком задач, которые он планирует взять себе на исполнение. Однако практическое применение полученной характеристики очень специфично: в вопросе оценивания разработчиков значение сдвига позволяет понять, насколько тот или иной разработчик склонен недо- или переоценивать себя в отношении времени исполнения задач. Но сама по себе эта характеристика без априорной информации о конкретном исполнителе мало информативна. Например, если менеджер не имеет опыта работы с текущей командой, и соответственно, не знает никого из ее членов, то знание значения сдвига слабо поможет ему решить задачу точности оценки временных затрат. И наоборот: если менеджер хорошо знает своих разработчиков, то эта оценка не даст ему новых управленческих инструментов, а только позволит подтвердить собственные представления о разработчике.

3. Ряд проверок других практически значимых проверяемых гипотез не приводили ни к подтверждению, ни к принятию альтернатив, и в целом полученные характеристики плохо подвергались анализу или рациональному использованию.

В результате проведенных работ даже получение некоторого прогресса в применении общепринятых методов приводит к тому выводу, что область их практического применения достаточно узкая и мало связана с общим управлением проектами, а большинство предположений о масштабировании или обобщении полученных результатов – к выводу, что либо это сложно реализуемо, либо накладывая достаточно специфические ограничения на всю проектную команду, которая должна их придерживаться для получения значимого результата.

3.2. Общие требования к прикладному математическому решению в области управления проектами в программной инженерии

Если подвести некоторые промежуточные итоги применения различных прикладных математических или(и) алгоритмических методов[35] к отрасли управления проектами по разработке программного обеспечения, можно сделать следующие достаточно противоречивые выводы.

[34] Как результат проверки на ранговую однородность.
[35] Например, методов машинного обучения.

Аналитические методы, которые хорошо себя зарекомендовали на общем уровне управления проектами или же в вопросах оптимизации инженерных процессов, плохо применимы при переносе их в контекст программной инженерии ввиду ее специфики, выраженной, с одной стороны, гибкостью и творческой природой данной деятельности, которые плохо согласуются с строгими инженерными методами, с другой – наличием инженерной составляющей и хотя и неярко выраженного технологического процесса, лежащего в основе разработки программного обеспечения.

Класс методов, основанный на детальном графике проекта, плохо применим ввиду того, что составление подобного графика экономически нецелесообразно в большом проценте проектов по разработке программного обеспечения.

Адаптивные математические и алгоритмические методы требуют построения модели проекта или же способа формально описывать его состояние и прогресс, что сводится к следующим случаям:

1) появление альтернативного вида артефакта для календарного плана-графика, что требует сравнительно такого же объема времени на создание и поддержание его в актуальном состоянии;

2) разработка альтернативной модели проекта, которая зачастую создается без учета специфики программной инженерии.

3) Если создатели этих моделей учитывают специфику разработки программного обеспечения, то подобные типы моделей для своей эффективной работы также требуют хорошо подобранной и структурированной обучающей выборки с актуальными данными, которые можно получить только в профессиональном сообществе, которое необходимо убедить пойти на связанные со сбором данных расходы. А поскольку большая часть предыдущих попыток академического сообщества разработать подобные прикладные методы сложно оценить как успешные, то, необходимы убедительные аргументы.

Проведенный анализ показал, что в области управления проектами по разработке программного обеспечения прикладной подход должен:

1) быть достаточно общим, чтобы не накладывать жесткие ограничения на его применимость к проектным командам профессионального сообщества;

2) позволять учитывать специфику связанной с разработкой программного обеспечения деятельности, но учитывать ее не строго, иначе это негативно отразится на применимости;

3) получать достаточно значимые с практической точки зрения результаты, не имея при этом гарантировано рабочей и тщательно отобранной обучающей выборки;

4) быть улучшаемым и модифицируемым в зависимости от полученных результатов, если метод не является строго математически выведенным для данной задачи;

5) не требовать больших ресурсных затрат от команды;

6) хорошо интегрироваться с программными средами для управления проектами.

3.3. Применение байесовских динамических сетей для программной инженерии

3.3.1. Переход от проекта по разработке программного обеспечения, выраженного через теоретический контур Essence, к динамической байесовской сети

В рамках данной диссертации в качестве математической модели для построения системы поддержки принятия решений предлагается использовать подход, основанный на использовании динамических байесовских сетей. Математическая теория использования байесовских сетей была разработана Д. Перлом в 1988 г. [101]. Согласно определению, байесовская сеть представляет собой ориентированный ацикличный граф, вершины которого составляют случайные переменные, а дуги фиксируют отношения условной зависимости для этих переменных. Формально это можно зафиксировать следующим образом: под байесовской сетью понимается направленный циклический граф $G = < V, E >$, в котором каждой вершине $v \in V$ поставлена в соответствие случайная величина Y_v и каждое ребро $(u, v) \in E$ представляет прямую зависимость Y_v от Y_u. Пусть $parents(v) = = u|(u, v) \in E$, тогда в байесовской сети каждой вершине $v \in V$ графа должно быть сопоставлено распределение условных вероятностей вершин из $parents(v)$:

$$P(Y_1, \dots Y_n) = \prod_{i=1}^{n} P\big(Y_i \big| parents(Y_i)\big). \quad (1)$$

Использование байесовских сетей позволяет описать следующие вероятностные запросы и получить на них ответы: нахождение вероятности свидетельства[36], определение априорных маргинальных вероятностей, определение апостериорных маргинальных вероятностей, вычисление наиболее вероятного объяснения наблюдаемого события, вычисление апостериорного максимума [104].

[36] Под свидетельством здесь понимается утверждение типа «в конкретном узле наступило событие».

Рассмотрим переход, описывающий, каким образом можно использовать математический аппарат динамической байесовской сети для описания теоретического контура Essence. Для этих целей будем использовать представление теоретического контура в форме Alpha State Game.

Поскольку Alpha State Game сводится к последовательному анализу утверждений[37], зафиксированных на карточках состояний Alpha для текущего проекта, и принятию решений об их истинности или ложности, то можно обозначить: X_i – 204-битовая строка, описывающая -ое состояние рассматриваемого проекта, где $x_{i,j}$ – это бит, обозначающий истинность или ложность утверждения j из Alpha State Game для состояния i проекта.

В связи с тем, что в модели Essence каждое утверждение семантически фиксирует завершение работ с определенными типами рисков проекта, то в идеальной теоретической модели любой проект должен перейти от состояния, в котором все 204 бита – нули: состояние начала проекта, к состоянию, в котором все 204 бита – единицы: проект полностью завершен и закрыт. Формально это можно записать следующим образом: $X_0 ::= \{x_{0,j} = 0\} \rightarrow X_n ::= \{x_{n,j} = 1\}, \forall j \in [1..204]$, где n – окончательное состояние рассматриваемого проекта. Учитывая тот факт, что переход из начального состояния в конечное происходит через процедуру принятия решений, для любого проекта, который был завершен, можно зафиксировать последовательность $X_0, X_1, X_2, \dots, X_{n-1}, X_n$, которая будет описывать историю его развития с точки зрения теоретического контура Essence.

Если рассматривать незавершенный проект, эта последовательность находится в стадии формирования, и будет выглядеть следующим образом: $X_0, X_1, X_2, \dots, X_k$, где k – текущее состояние проекта. При этом в процессе выполнения проекта менеджер определяет планы по распределению работ, тем самым фиксируя множество $A_k := \{x_{k,j}\}$ утверждений, с которыми будут вестись работы в следующий период времени выполнения проекта. Следуя логике реализации проекта, на следующем шаге получим

$$\forall k \, \forall j \, x_{k,j} \in A_k \rightarrow x_{k+1,j} = 1. \quad (2)$$

Таким образом, на практическом уровне одна из задач менеджера – составление множества A_k в зависимости от общего проектного плана и текущего состояния проекта X_k. Соответственно, потенциальной задачей системы поддержки принятия решений может быть поиск тех $x_{k,j}$, которые наиболее вероятно должны попасть в A_k. Вероятность попадания $x_{k,j}$ зависит от множества субъективных и плохо формализуемых факторов, к которым можно отнести категорию проекта, тип разрабатываемой системы, текущий набор требований, договоренности

[37] Всего утверждений 204.

со стейкхолдерами и т. п. Необходимо также отметить тот факт, к сожалению, что в этом случае в связи с субъективной природой этих факторов плохо применимы стандартные вероятностные методы, основанные на частотном определении вероятности.

При этом существует как минимум один тип факторов, который можно зафиксировать для любого проекта. Он основан на том, что теоретический контур Essence явно подчеркивает неравнозначность утверждений на Alpha-карточках по отношению друг к другу. В этом легко убедиться: во-первых, явно задан набор связей между разными Alpha; во-вторых, состояния этих Alpha имеют последовательную нумерацию; и, наконец, существуют различные Activity Space, которые фиксируют состояния Alpha на входе и выходе. Таким образом, можно сделать вывод, что утверждения $x_{i,j}$ влияют друг на друга.

Это влияние имеет нестрогую причинно-следственную природу, поскольку и теоретический контур, и детализированные практики фиксируют только общие состояния Alpha-карточек, а не конкретные последовательности выполнения задач. Учитывая нестрогую причинно-следственную связь, можно сделать предположение, что это влияние описывается как условная зависимость. Для примера допустим, что детальные требования к программному обеспечению будут написаны еще до того, как стейкхолдеры подтвердят высокоуровневое описание системы. Однако проекты, в которых сначала происходит фиксация высокоуровневого определения, а уже потом детализируются требования, на практике встречаются гораздо чаще.

Необходимо учитывать и другую особенность управления проектами по разработке программного обеспечения – сильную экспертную составляющую. При использовании Alpha State Game именно менеджер принимает экспертное решение об истинности или ложности каждого утверждения. Однако, учитывая субъективную природу принятия этого решения, можно сделать вывод, что мнение менеджера об истинности утверждения может не означать истинность этого утверждения для проекта на самом деле, и в этом случае происходит ошибка менеджера.

Из вышеизложенного можно сделать вывод, что истинность любого $x_{i,j}$ для незавершенного проекта – это случайное событие, а мнение менеджера и значения истинности семантически связанных $x_{i,z}$, где $z \neq j$ – свидетельства для $x_{i,j}$. Это позволяет перейти к математическому аппарату динамических байесовских сетей, используя которые можно трансформировать описанную выше модель из последовательности X_i и $x_{i,j}$ в динамическую байесовскую сеть, состоящую из вершин $v_{j,i}$, фиксирующих индикаторную случайную величину – значение истинности утверждения j для -го состояния проекта, для каждой из которых можно определить множество вершин-родителей $parents(v_{j,i})$, к которому будут относится вершины из текущего

состояния проекта и вершина $v_{j,i-1}$. Данное представление можно расширить, введя дополнительный класс вершин $\mathrm{m}_{j,i}$, фиксирующих мнение менеджера об истинности суждения j в состоянии i, причем в зависимости от рассматриваемой задачи этот класс вершин может фиксировать наблюдаемые значения[38].

Использование аппарата динамических байесовских сетей, построенных для описания проектов по разработке программного обеспечения, позволит формулировать и решать достаточно широкий спектр разнообразных задач, например:

– решение стандартной задачи динамической оптимизации полного совместного распределения приведет к поиску вершин, изменение истинности которых максимально улучшит общий прогресс проекта;

– поиск наибольшего апостериорного максимума при наличии данных об истории будет иметь практическую интерпретацию анализа связей между различными утверждениями для обнаружения наиболее проблемных зон команды при выполнении проектов;

– аналогично сравнительный анализ нахождения вероятности свидетельства при наличии данных о том, какими практиками пользовалась команда в различных проектах, или анализ эффективности практик для снятия тех или иных рисков в процессе выполнения проектов.

В данной работе с учетом общего процента неуспешных проектов по разработке программного обеспечения рассматривается более актуальная для практического применения прикладная задача: поиск ложноположительных ошибок менеджера при анализе текущего состояния проекта. В математической постановке она выглядит следующим образом: необходимо найти такие вершины, что

$$P\left(v_{j,k} = 1 | m_{j,k} = 1\right) < \alpha, \tag{3}$$

где α – заданная пороговая величина.

Проверим, что аппарат динамических байесовских сетей и сформулированная задача удовлетворяют тем требованиям, которые были зафиксированы в § 3.2.

Во-первых, требования 1–2. Alpha-карточки, их состояния и утверждения на них являются достаточно общими практически для всех проектов по разработке программного обеспечения. Более того, в первой главе данной работы показано, что совокупность этих элементов стандарта OMG Essence обеспечивает полноту теоретического контура метода программной инженерии. Одновременно с этим, они разработаны ровно для описания прогресса в деятельности по разработке

[38] Непосредственно отметил или не отметил менеджер эти утверждения на карточках.

программного обеспечения. Байесовская сеть, построенная на основе утверждений из Alpha-карточек, также будет учитывать связанную семантику между утверждениями.

Во-вторых, требования 3–4. Обычно метод, основанный на использовании байесовских сетей, подразумевает наличие выборки данных, на основе которых определяются вероятности связей для максимизации правдоподобия. Однако в целом метод основан на экспертном построении структуры самой сети (например, в вопросах адаптивного обучения [103, 104] именно разработчики модели должны построить изначальный граф.), и нет никакого математического запрета на то, чтобы эксперты оценили вероятности связей между вершинами. С учетом того, что эти вероятности имеют семантическое объяснение, первичное приближение к решению данной задачи может быть основано именно на таких экспертных оценках. При этом появление реальных данных от профессиональной команды, которая возьмется использовать подобный подход, потенциально позволит улучшить точность вероятностей, определенных экспертами, или же перейти к более строгим и комплексным задачам.

В-третьих, требование 5. Затраты ресурсов на поддержку в проекте теоретического контура Essence, например в форме Alpha State Game [105], сводятся к 15–30 минутам в итерацию, причем зачастую достаточно одного менеджера. Необходимо заметить, что это затраты на фиксацию прогресса проекта с помощью Alpha State Game, что для компаний с большим количеством разнообразных проектов может иметь самостоятельную практическую ценность.

Наконец, требование 6. Данный метод автоматизируется и не требует экспертного сопровождения после выполнения процедур внедрения в проектную команду.

Учитывая практическую направленность настоящей работы, далее опишем последовательный переход от задачи в математической постановке к построению и первоначальному заполнению реальной динамической байесовской сети и ее реализации в среде управления проектами.

3.3.2. Представление теоретического контура Essence в форме динамической байесовской сети

Базовой сущностью теоретического контура Essence является утверждение, связанное с Alpha State, причем со стороны логики построения математической модели каждое утверждение может быть либо истинным, либо ложным. Однако теоретический контур Essence основан на субъективной оценке значения проекта его менеджером, т. е. по сути именно менеджер экспертно-субъективно принимает решение о том, является ли это утверждение истинным или ложным, что ведет

к очевидной проблеме: если по разным причинам менеджер ошибается, эта ошибка влияет на все состояние проекта. Поэтому, чтобы избежать полной зависимости от субъективного мнения менеджера, разделим каждое утверждение на две различные вершины – скрытую и явную. Таким образом для описания одного утверждения мы вводим элементы байесовской сети, представленные на рисунке 43, которую можно трансформировать в следующее, «если мы знаем мнение менеджера о значении состояния «истина / ложь» данного утверждения, то каково значение этого состояния на самом деле?».

Рисунок 43 – Представление элемента байесовской сети для фиксации одного утверждения

Или, если учитывать, что и мнение менеджера может быть либо истинным, либо ложным, и скрытое состояние тоже может быть или истинным, или ложным, мы можем перейти к следующему: пусть A_i – событие, что менеджер считает -ое утверждение истинным, а B_i – событие, что i-ое утверждение истинно. Тогда можно начинать формулировать задачи об определении $P(B_i|A_i)$, т. е. вероятности того, что утверждение истинно при условии, что менеджер считает также – иными словами, менеджер не ошибается. Здесь важно отметить, что в ситуации применения к управлению проектом при использовании теоретического контура A_i – это наблюдаемая величина, т. е. A_i не является случайным событием при конкретных расчетах.

Дальнейшее усложнение связано с динамическим аспектом разработки программного обеспечения. В большинстве случаев весь проект по разработке программного обеспечения принято делить на части, которые в зависимости от используемого процесса разработки имеют различные названия, такие как итерация, спринт, спираль, недельный план. Фактически такой подход подразумевает, что анализ достигнутых результатов и планирование следующей части проекта происходят регулярно и проект изменяется динамически. В связи с этим необходимо также внести вершину, которая выполняла бы функцию динамического аспекта изменения проекта (рисунок 44).

Рисунок 44 – Элемент байесовской сети с динамическим аспектом изменения проекта

Таким образом, к изложенным выше событиям мы можем добавить C_i – утверждение i было истинным на предыдущей итерации проекта. Эта вершина байесовской сети в целом является скрытой, поскольку ее значение мы могли узнать аналогичным образом, что и B_i, но на одну итерацию раньше. С целью обеспечения удобства подсчета вероятностей в самой первой итерации проекта для C_i можно указывать вероятность того, что это утверждение может быть истинным в начале проекта: например, в некоторых случаях заказчик может прийти к команде, уже имея строго написанное техническое задание. Более того, если рассматривать примеры жизненных циклов, которые описаны в приложении стандарта OMG Essence, можно указать даже конкретные типы проектов, для которых эти состояния уже не будут являться случайным событием, но для общей структуры байесовской сети это не существенно.

В связи с тем, что все утверждения связаны с определенной Alpha State, при группировке и учете связей через Activity Space для Alpha State также потребуются дополнительные вершины, однако поскольку Essence формально определяет правило истинности Alpha State[39], вероятности истинности события, что Alpha State достигнуто, можно выразить графически (рисунок 45) или в виде формулы

$$P(B) = \prod_i P(B_i), \tag{4}$$

где событие B – то, что Alpha State B – истинно;
B_i – утверждения, относящиеся к Alpha State.

[39] Все утверждения Alpha State должны быть истинны.

Рисунок 45 – Группировка элементов байесовской сети для определенной Alpha State

После определения общей структуры вершин байесовской сети, необходимой для выражения отдельных утверждений из теоретического контура Essence, можно построить несвязанную байесовскую сеть, отражающую основные связи, характерные для всех утверждений. Примеры вершин этой сети приведены в таблице 2. Для дальнейшей записи используемых утверждений примем следующий шаблон обозначения узлов:

аббревиатура названия ALPHa + номер Alpha State +номер Утверждения по списку из стандарта + + (опционально) специальный символ.

Таблица 2 – Примеры вершин байесовской сети с введеным шаблоном обозначения

Пример вершины	Расшифровка	Пояснение
S13	Alpha: Stakeholders State: 1 Утверждение: 3 «Stakeholder groups identified»	Вершина, отвечающая за скрытый узел, связанный с утверждением, событие B_i
SS23C	Alpha: Software System State: 2 Утверждение: 3 «Critical interfaces demonstrated» Служебный символ: C	Вершины со служебным символом C – это вершины, которые отвечают за наблюдаемое мнение менеджера об этом утверждении, событие A_i
WW41S	Alpha: Way of Working State: 4 Утверждение: 1 «Accessible to whole team» Служебный символ: S	Вершины со служебным символом S – это вершины, которые отвечают за динамическую логику изменения проекта, в т. ч. начальное распределение вероятности, событие C_i
R2	Alpha: Requirements State: 2	Вспомогательные вершины, отвечающие за соответствующие Alpha State, событие B

С использованием введенных обозначений для утверждений ALPHa каждой из зон получим следующие байесовские сети, представленные на рисунках 46–48. Если объединить все утверждения и связи, явно указанные на уровне теоретического контура Essence, получим следующую общую сеть, представленную на рисунке 49.

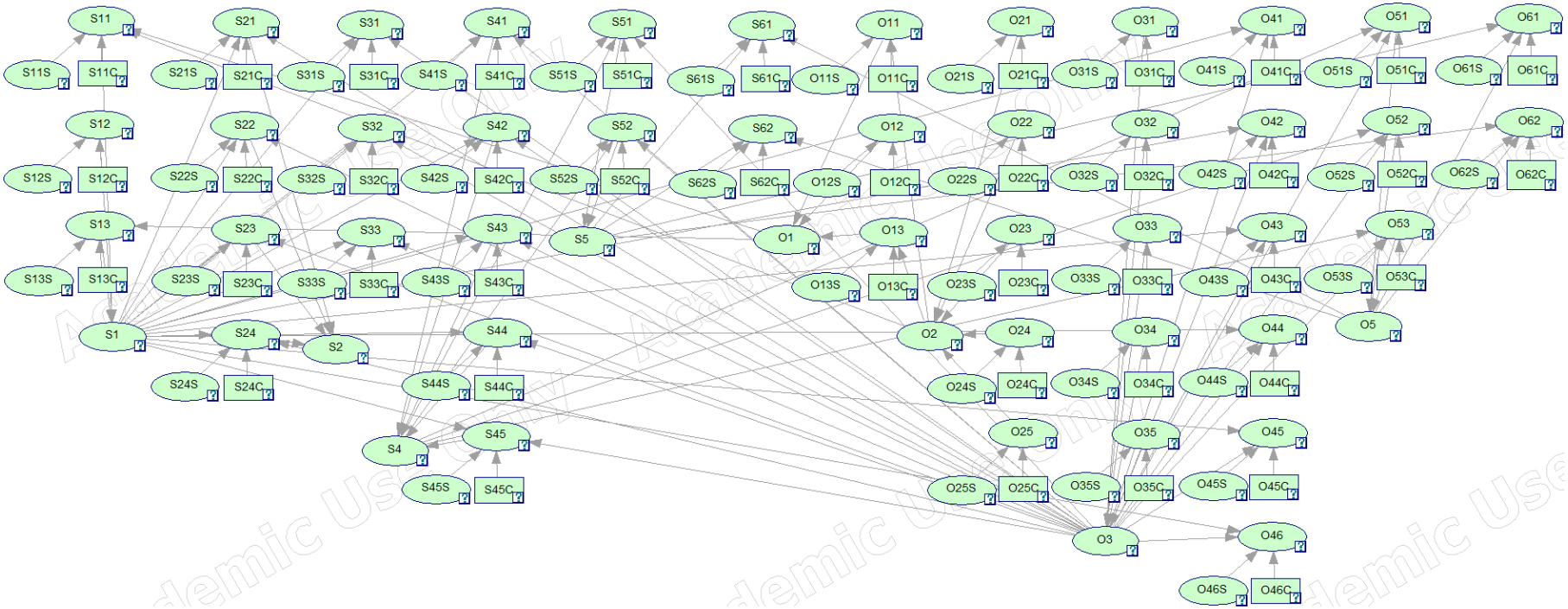


Рисунок 46 – Байесовская сеть для утверждений Alpha State зоны Customer

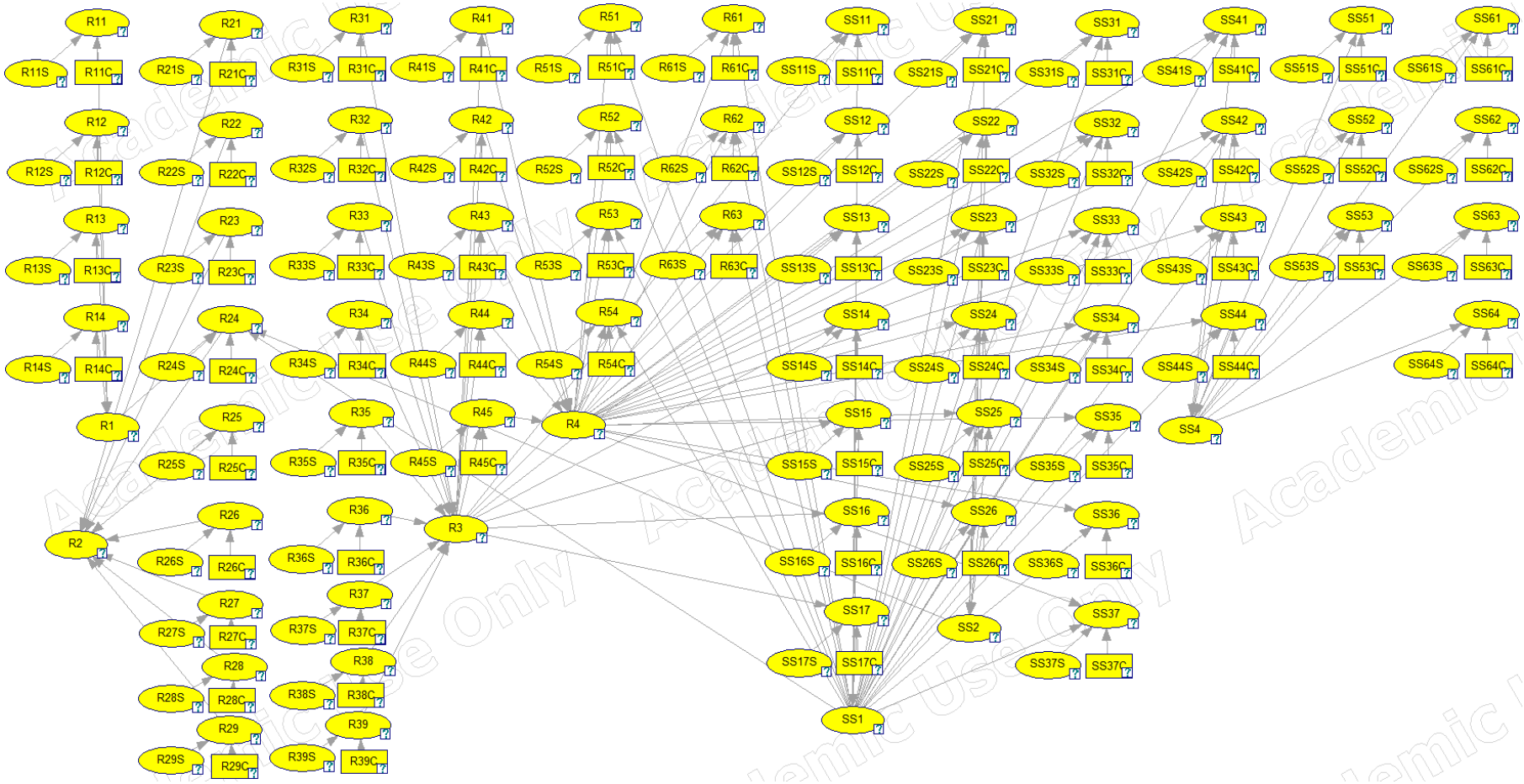


Рисунок 47 – Байесовская сеть для утверждений Alpha State зоны Solution

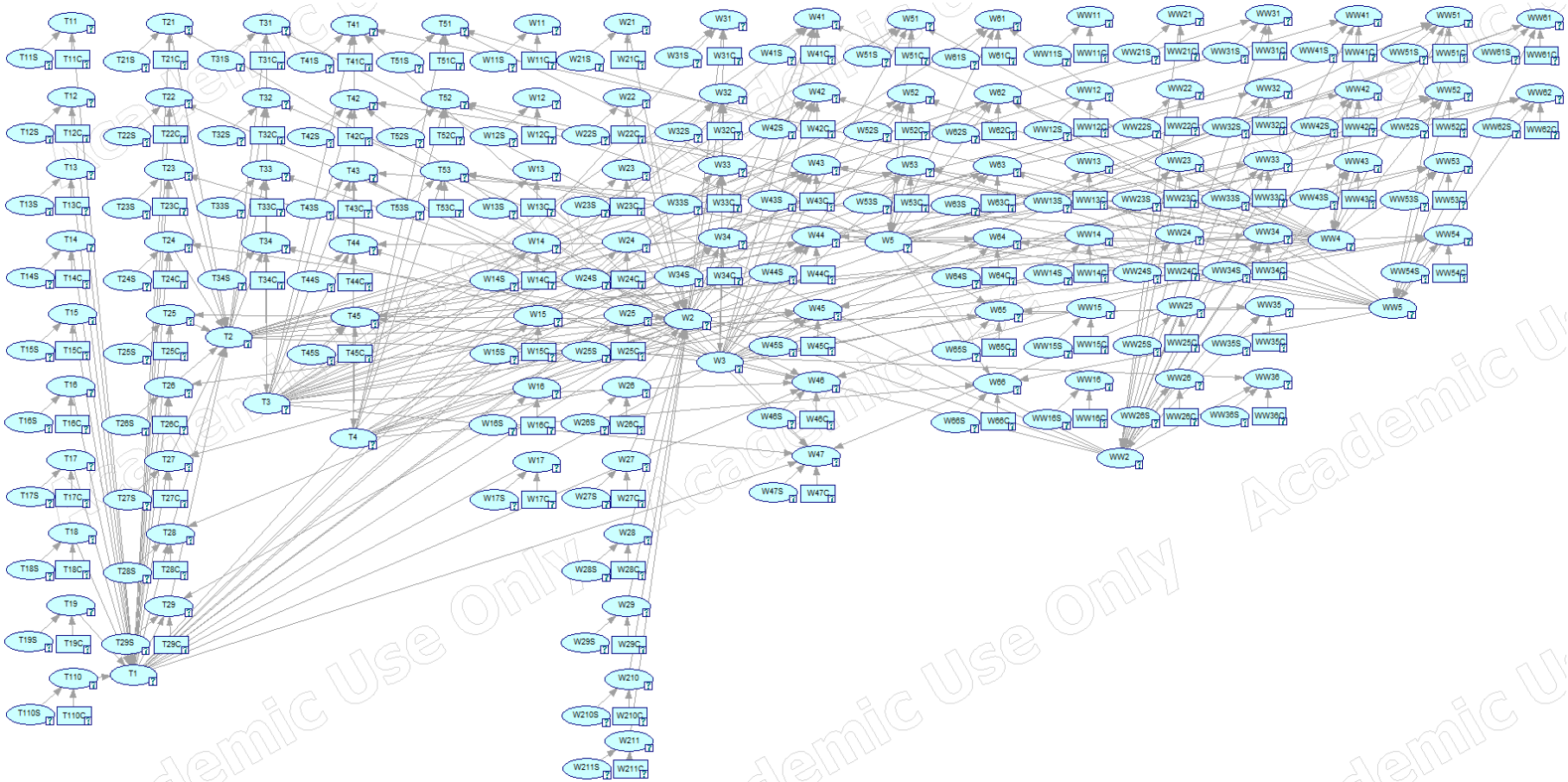


Рисунок 48 – Байесовская сеть для утверждений Alpha State зоны Endeavour

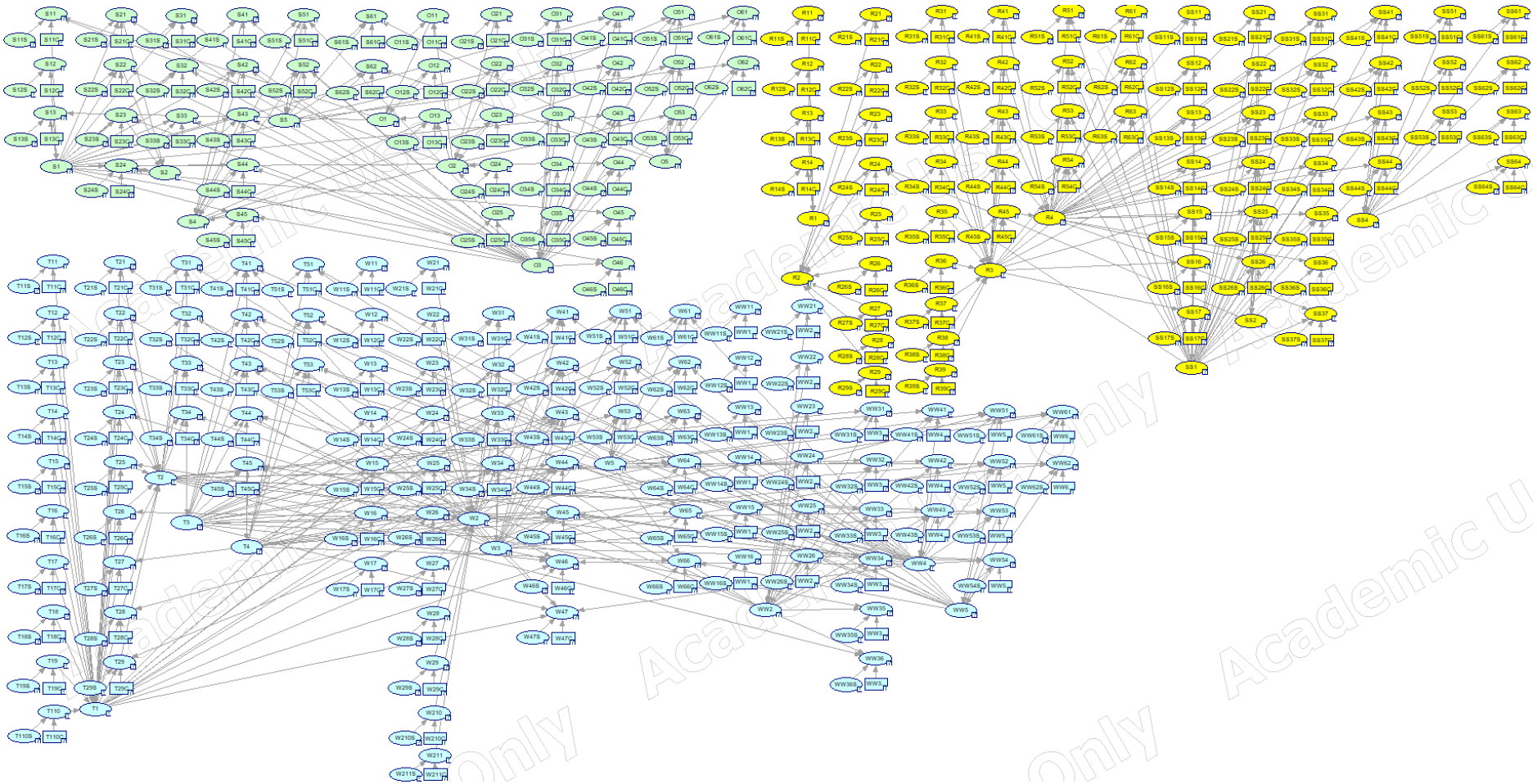


Рисунок 49 – Общая байесовская сеть для различных зон Alpha State

Как можно заметить, для теоретического контура Essence (рисунок 11) и построенной по нему байесовской сети (рисунок 49) одна из особенностей заключается в не связанности элементов из альф, принадлежащих разным зонам, друг с другом, даже если семантически эта связь явно существует. Например, для Alpha «Requirements» State «Conceived» утверждение «Stakeholders agree system is to be produced» явно подразумевает, что стейкхолдеры должны быть как минимум выявлены[40], и желательно – иметь своих представителей[41], которые должны быть авторизованы[42], чтобы соглашаться на разработку системы. Однако на основе существующих формальных описаний Essence Kernel или теоретического контура Essence эти связи явно не указаны, в связи с чем возникает задача о расширении полученной сети за счет добавления семантических связей между утверждениями, решение которой рассмотрено в следующем пункте работы.

3.3.3. Расширение байесовской сети теоретического контура Essenceза счет добавления семантических связей между утверждениями

В целом тот факт, что элементы теоретического контура явно не специфицируют иногда достаточно очевидные семантические связи между отдельными утверждениями, уже отмечался, и была предпринята попытка использовать эту семантически значимую информацию в диссертации Й. Пипера[43] [106]. Однако в качестве дополнительных связей рассматривались семантически значимые связи между Alpha State и утверждениями других Alpha State. Если проводить параллель с примером выше, то добавлялась связь не с отдельными утверждениями «Stakeholder groups identified», «Key stakeholder groups represented», «Representatives authorized», которые влияют на «Stakeholders agree system is to be produced», а то, что Alpha «Stakeholders», States «Recognized» и «Represented» должны быть выполнены для выполнения «Stakeholders agree system is to be produced». Такое решение позволяет учитывать дополнительную логику, идущую из семантики, однако для некоторых отдельных элементов теоретического контура будет происходить достаточно большое добавление зависимостей: так, некоторые Alpha State – условно, причины – могут иметь десять и более утверждений, причем для описываемой логики они все должны быть выполнены, чтобы работала связь с одним-единственным утверждением из другого Alpha State, а на уровне семантики явно значимым может оказаться только одно утверждение из Alpha State причины.

[40] Stakeholder groups identified.
[41] Key stakeholder groups represented.
[42] Representatives authorized.
[43] Jöran Pieper (нем.)

Исходя из вышесказанного, можно сделать вывод, что необходимо расширить связи напрямую между утверждениями за счет анализа семантики происходящего в управлении проектами и согласно формулировкам утверждений на Alpha State карточках. Причем подобное расширение должно также предполагать разные степени влияния утверждений друг на друга. Поскольку в некоторых случаях – как в рассмотренном выше примере с «Stakeholders agree system is to be produced» – точно подразумевает, что стейкхолдеры должны быть хотя бы выделены, и это сильное влияние, однако в целом допустимо получить согласие стейкхолдеров на разработку системы, даже не имея «Representatives authorized», т. к. в целом авторизация стейкхолдеров на действия делает их согласие более значимым, однако в этом нет явного противоречия с тем, чтобы получить согласие стейкхолдеров без полной авторизации их представителей.

Для формализации связей было принято решение составлять карточки утверждений, внутри которых указывать дополнительные связи (пример приведен на рисунке 50).

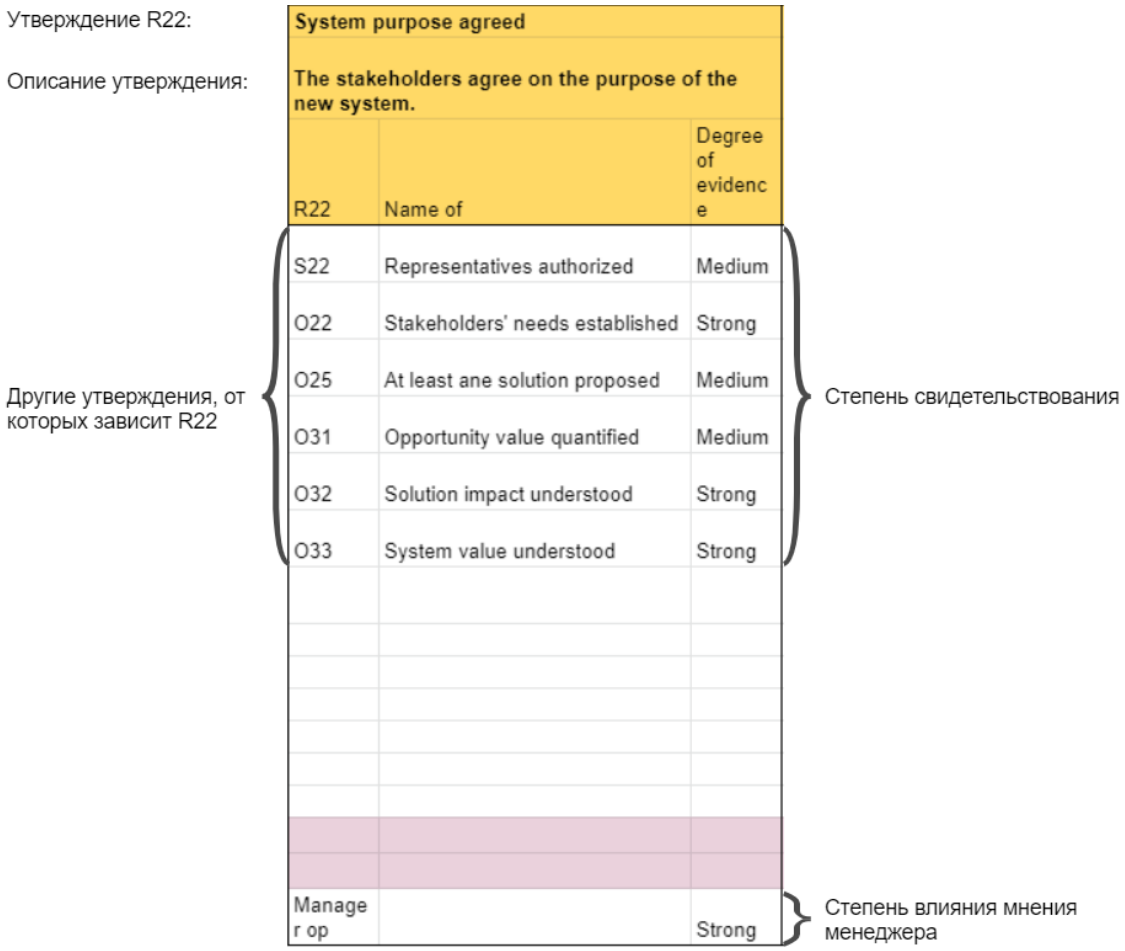


Рисунок 50 – Карточка утверждения

Данная карточка позволяет зафиксировать связи с утверждениями, которые можно представить как следующий фрагмент байесовской сети (рисунок 51).

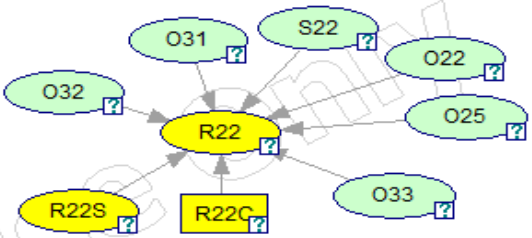


Рисунок 51 – Фрагмент байесовской сети для утверждения R22

В общем случае подобный вид карточки позволяет описать утверждение, его номер по принятым обозначениям, а также перечислить все утверждения, влияющие, согласно экспертному анализу, на данное. Помимо самой связи предлагается также оценить степень влияния этой связи на текущие утверждения, выбрав одну из категорий – Strong, Medium или Weak. Здесь важно отметить две особенные связи и их оценки.

1. Строка «Manager Op»[44] введена для оценки влияния мнения менеджера на утверждения, что позволят отразить тот фактор, что в некоторых утверждениях менеджеры более склонны к тому, чтобы переоценивать прогресс и, соответственно, степень влияния тогда более низкая.

2. На карточке не представлена связь с вершиной из предыдущей итерации[45], отвечающей за динамический аспект байесовской сети. После анализа было принято решение, что для всех утверждений теоретического контура Essence степень влияния предыдущей итерации – Weak, что связано с тем, что следствием того, что команда развивает проект будет, то что утверждения будут становится истинными и меняться от того, что все утверждения ложны к тому, что все утверждения станут истинными (ну при успешном прогрессе проекта), и, в связи с этим предыдущая итерация в целом влияет на анализ прогресса, однако не так существенно, как подготовка новых артефактов или изменение состояний связанных утверждений.

В ходе выполнения данной работы был проведен семантический анализ связей для всех состояний ALPHa из ядра OMG Essence, результатом которого стал набор, состоящий из 204 карточек [107]. В качестве примера на рисунке 52 продемонстрирован набор карточек утверждений для ALPHa «Software System» State «Demonstrable» с добавленными связями от прочих утверждений. На рисунке 53 представлена байесовская сеть с учетом всех добавленных связей, полученная на основе представленной выше сети.

[44] Она же вершина R22C в рассматриваемом примере.
[45] Она же вершина R22S.

SS2																	
Demonstrable																	
Key architecture characteristics demonstrated			System exercised &perfomance measured			Critical HW configurations demonstrated			Critical interfaces demonstrated			Integration with environment demonstrated			Architecture accepted as fit-for-purpose		
Key architectural characteristics have been demonstrated.			The system can be exercised and its performance can be measured.			Critical hardware configurations have been demonstrated.			Critical interfaces have been demonstrated.			The integration with other existing systems has been demonstrated.			The relevant stakeholders agree that the demonstrated architecture is appropriate.		
SS21	Name of	Degree of evidence	SS22	Name of	Degree of evidence	SS23	Name of	Degree of evidence	SS24	Name of	Degree of evidence	SS25	Name of	Degree of evidence	SS26	Name of	Degree of evidence
R22	System purpose agreed	Medium	R35	Essential characteristics clear	Strong	SS21	Key architecture characteristics demonstrated	Strong	SS14	System boundary known	Strong	SS14	System boundary known	Strong	R39	Team knows &agrees on what to deliver	Strong
R23	System success clear	Medium	SS13	Technologies selected	Strong	SS15	Decision on system organization made	Strong	SS15	Decision on system organization made	Strong	SS15	Decision on system organization made	Strong	SS22	System exercised &perfomance measured	Strong
R31	Requirements shared	Strong	SS17	Key technical risks agreed to	Strong	SS12	HW platform identified	Strong	R39	Team knows &agrees on what to deliver	Strong	SS24	Critical interfaces demonstrated	Strong	SS23	Critical HW configurations demonstrated	Strong
R39	Team knows &agrees on what to deliver	Strong	SS21	Key architecture characteristics demonstrated	Strong	SS13	Technologies selected	Strong				R39	Team knows &agrees on what to deliver	Strong	SS24	Critical interfaces demonstrated	Strong
SS17	Key technical risks agreed to	Strong	R45	The requirements are testable.	Medium	R34	Conflicts addressed	Medium				T27	External collaborators identified	Strong	SS25	Integration with environment demonstrated	Strong
SS14	System boundary known	Strong				R35	Essential characteristics clear	Strong				W211	Integration points defined	Strong	R51	Enough addressed to be acceptable	Strong
						R39	Team knows &agrees on what to deliver	Strong							R41	Acceptable solution described	Strong
															W33	Definition of done in place	Strong
R4	Requirements - Acceptable	Strong	R4	Requirements - Acceptable	Strong	R4	Requirements - Acceptable	Strong	R4	Requirements - Acceptable	Strong	R4	Requirements - Acceptable	Strong	R4	Requirements - Acceptable	Strong
SS1	Software System - Architecture Selected	Strong	SS1	Software System - Architecture Selected	Strong	SS1	Software System - Architecture Selected	Strong	SS1	Software System - Architecture Selected	Strong	SS1	Software System - Architecture Selected	Strong	SS1	Software System - Architecture Selected	Strong
Manager op		Strong	Manager op		Strong	Manager op		Strong	Manager op		Strong	Manager op		Strong	Manager op		Strong

Рисунок 52 – Набор карточек утверждений для ALPHa «Software System» State «Demonstrable»

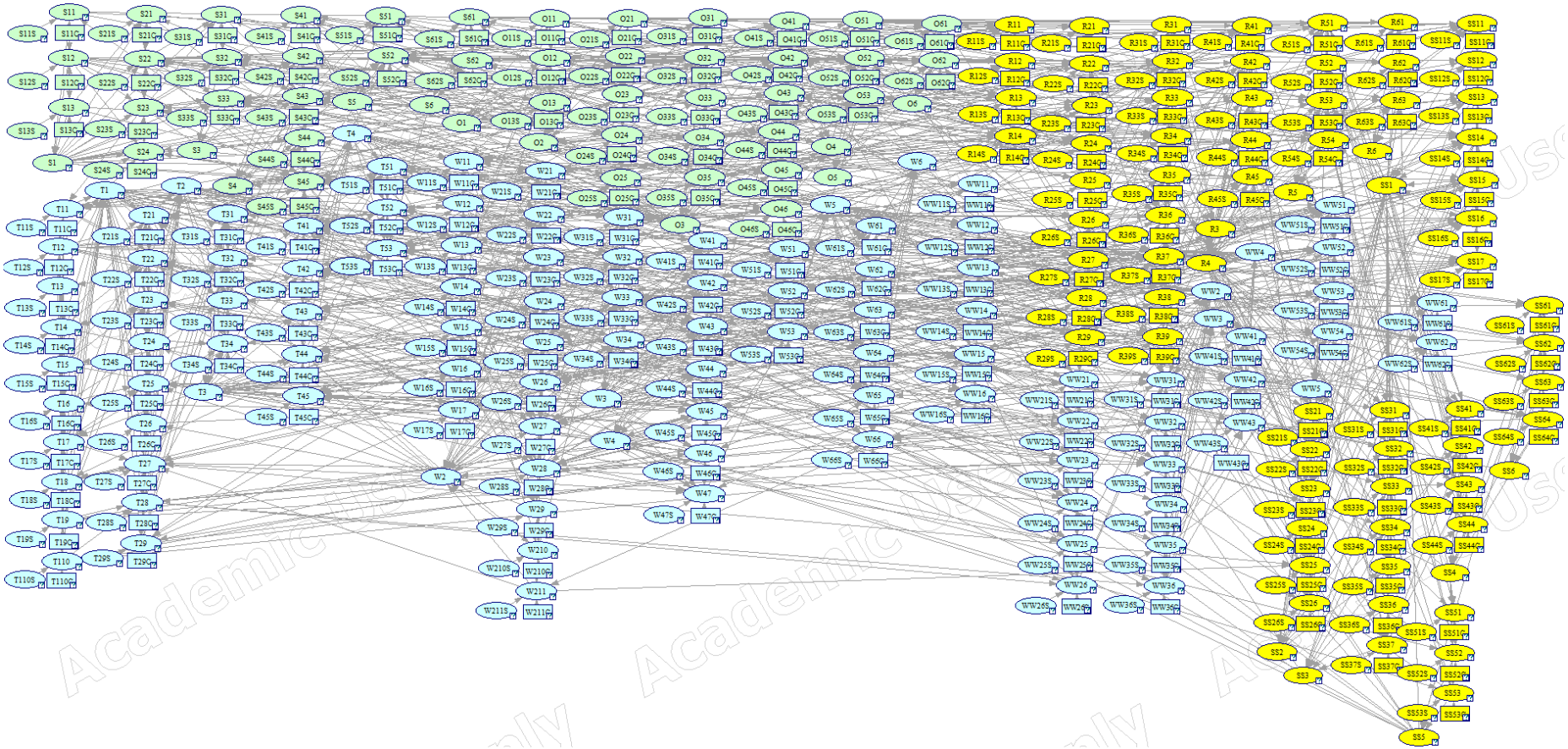


Рисунок 53 – Байесовская сеть с учетом всех добавленных связей

Таким образом, получается полная структура всех утверждений, зафиксированных в теоретическом контуре Essence, с расширенными связями между утверждениями, которые оценены по степени влияния, и позволяют учитывать это влияние для анализа прогресса проекта. Далее на примере поиска ложноположительных ошибок менеджера о состоянии проекта в Alpha State Game рассмотрим, как подобную динамическую байесовскую сеть можно использовать для помощи в управлении проектами.

3.3.4. Подсчет вероятностей скрытых вершин утверждений байесовской сети и поиск ложноположительных ошибок менеджера

В целом подходы, основанные на динамических байесовских сетях, предполагают фиксировать вероятности пребывания вершин в различных состояниях в зависимости от того, в каких состояниях находятся все вершины, влияющие на текущую (далее вершины-родители). Иными словами, если рассматривать пример, приведенный на рисунке 53, то существование подобной сети позволяет работать со следующей вероятностью

$$P(R22|R22S, R22C, O33, O25, O22, S22, O31, O32, O33), \quad (5)$$

где название каждой вершины означает событие, что соответствующее ей утверждение истинно.

И поскольку мы работаем только с двумя возможными значениями состояний утверждений – истинно или ложно, то необходимо учитывать все возможные комбинации значений вершин-родителей, от которых зависит R22. Отсюда следует вывод, что для каждой вершины-утверждения – B_i динамической байесовской сети – необходимо определить вектор вероятностей, в котором каждый элемент будет означать вероятность того, что B_i принимает истинное значение, при условии, что значения вершин-родителей соответствуют битовому представлению порядкового номера элемента.

Рассмотрим пример утверждения «W14» «Initiator identified», чьими вершинами-родителями являются только «W14C» и «W14S». Для этого утверждения данный вектор будет выглядеть следующим образом:

$$(0.99, 0.86087, 0.12913, 0), \quad (6)$$

где $P(W14|W14C, W14S) = 0.99;$
$P(W14|W14C, !W14S) = 0.86087;$
$P(W14|!W14C, W14S) = 0.12913;$
$P(W14|!W14C, !W14S) = 0.$

Зафиксировав такую структуру вероятностного вектора для вершин-утверждений, необходимо определить алгоритм: каким образом на основании оценок влияния связей вершин-родителей на текущую получить вероятностный вектор текущей вершины. Важно заметить, что даже при полной истинности всех вершин-родителей было принято решение считать итоговую вероятность за 0,99: всегда существует вероятность того, что в проекте допущена ошибка даже в самом формальном вопросе, и связи с этим должна быть вероятность того, что это может произойти. Однако для случая, когда все вершины-родители – ложны, вероятность истинности текущей вершины утверждения равна нулю, поскольку иначе допускается ситуация, в которой уже на старте проекта система разработана, внедрена и приносит прибыль заказчику, что противоречит здравому смыслу. Теперь опишем данный алгоритм в форме диаграммы деятельности (рисунок 54).

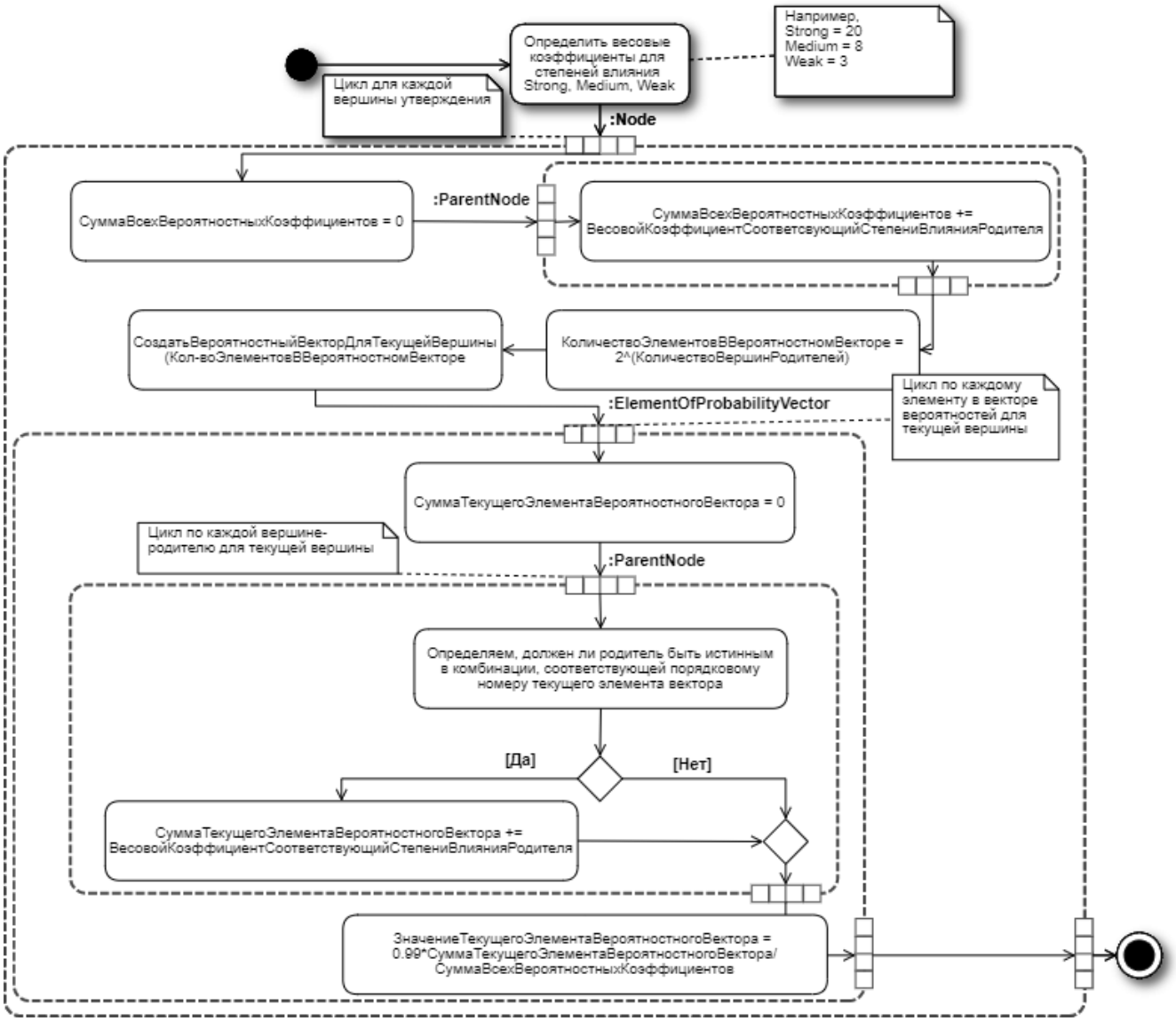


Рисунок 54 – Алгоритм получения вероятностного вектора на основе оценок степеней влияния вершин-родителей

Описанный выше алгоритм позволяет получить вероятностный вектор для каждой вершины утверждения, представленной в динамической байесовской сети, что позволяет рассчитывать вероятности истинности каждого утверждения при заполненном менеджером Alpha State Game на текущую итерацию и обеспечивает хранение полной истории всех итераций до текущей. Алгоритм для данного расчета изображен ниже в форме диаграмм деятельности UML (рисунки 54, 55). В общем виде подсчет вероятностей истинности каждой вершины представлен на рисунке 55.

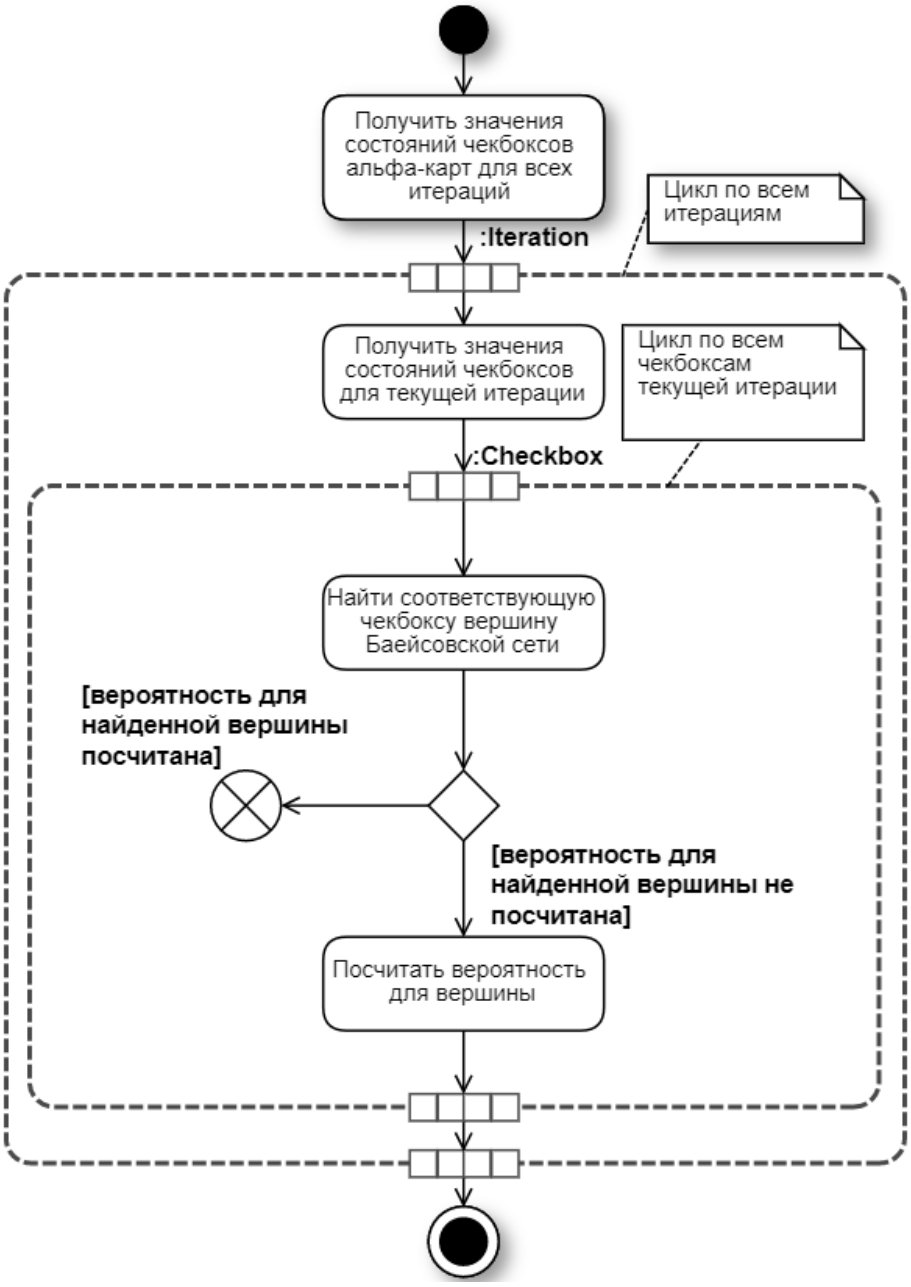


Рисунок 55 – Пересчет вероятностей вершин при наборе итераций

Очевидно, что наибольший интерес представляет активность «Посчитать вероятность для вершины», функция подсчета которой описана на диаграмме последовательности (рисунок 56).

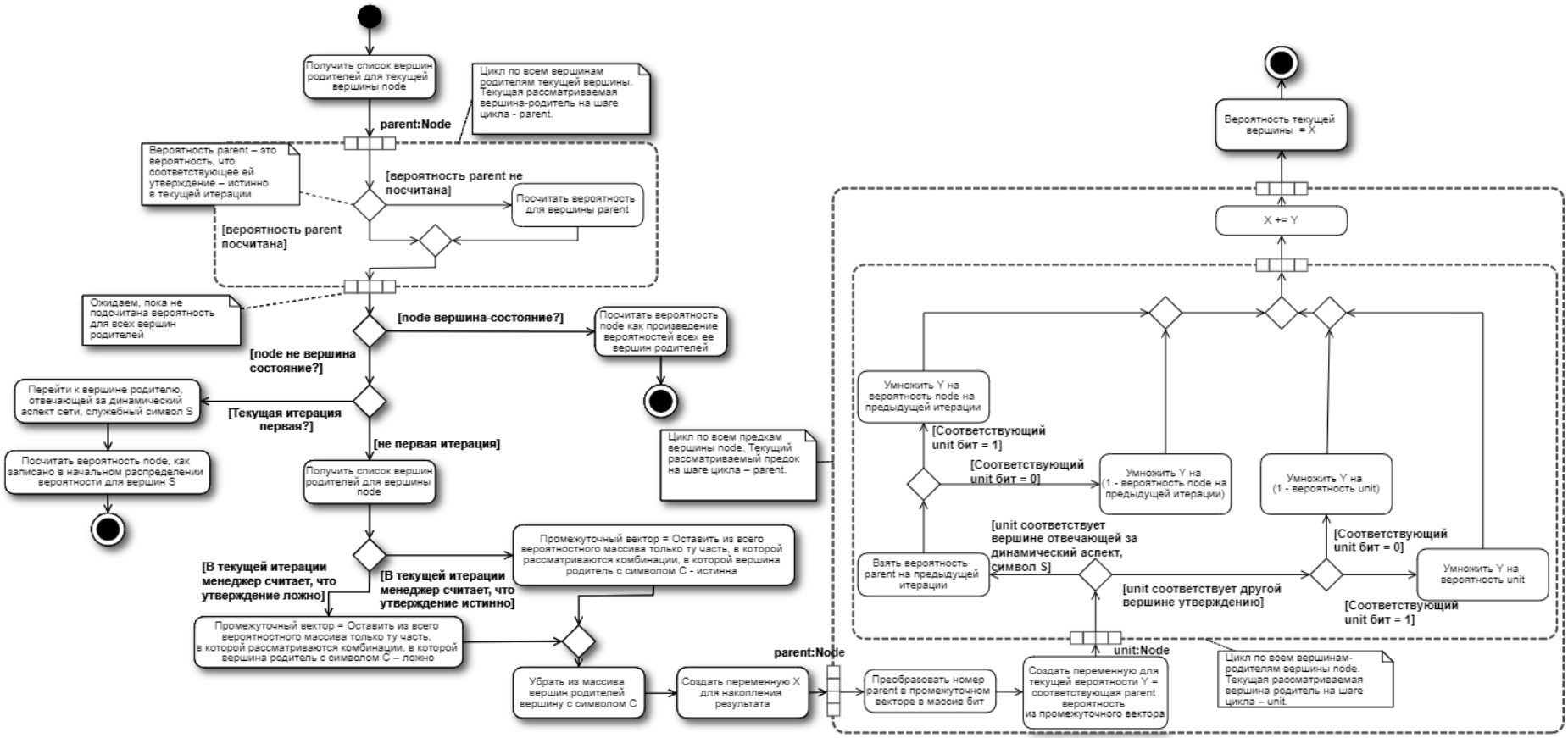


Рисунок 56 – Функция подсчета вероятности текущей вершины

Благодаря вышеописанному алгоритму для всех утверждений можно посчитать вероятность того, что в текущей итерации они являются истинными, с учетом текущего мнения менеджера о них. После проведенных расчетов, для того чтобы найти ложноположительные ошибки менеджера, теперь достаточно выполнить два простых шага:

1) выбрать все утверждения, которые менеджер считает истинными;

2) выбрать те из них, вероятность истинности которых ниже порогового значения, которое можно определить и как достаточно строгий критерий достоверности в 0,95, и как допустимый для текущего проекта – например, в 0,5.

3.4. Демонстрация работы эволюционного прототипа системы поддержки принятия решений

Описанный в этой главе алгоритм был реализован как часть расширения к системе управления Redmine, описанное во второй главе. В качестве демонстрации поиска ложноположительных ошибок менеджера рассмотрим следующий случай, для описания которого приведем состояния начальных Alpha проекта, где есть утверждения, которые менеджер считает истинными, и состояние Alpha Requirements (рисунки 57–61).

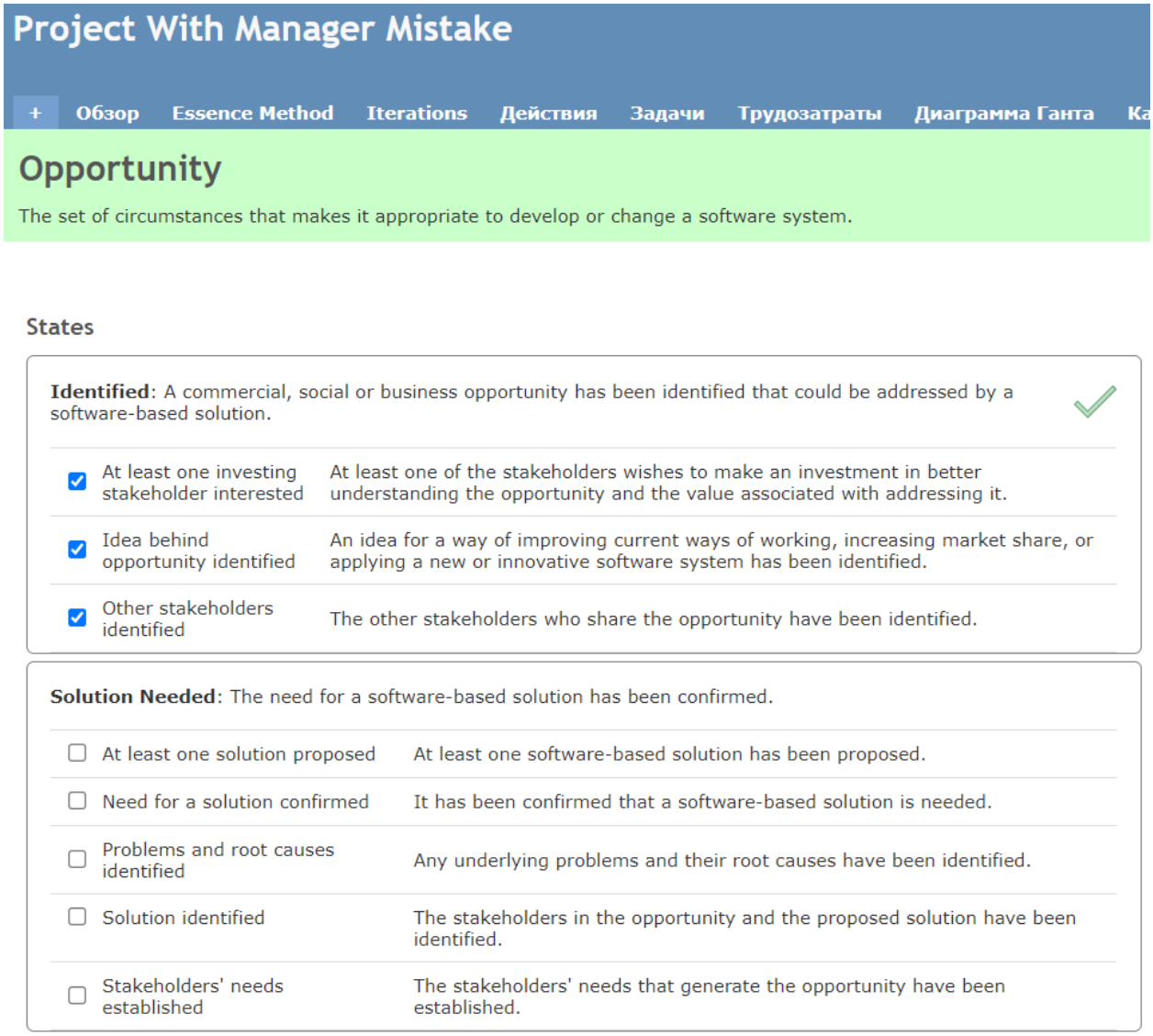


Рисунок 57 – Скриншот ALPHы Opportunity

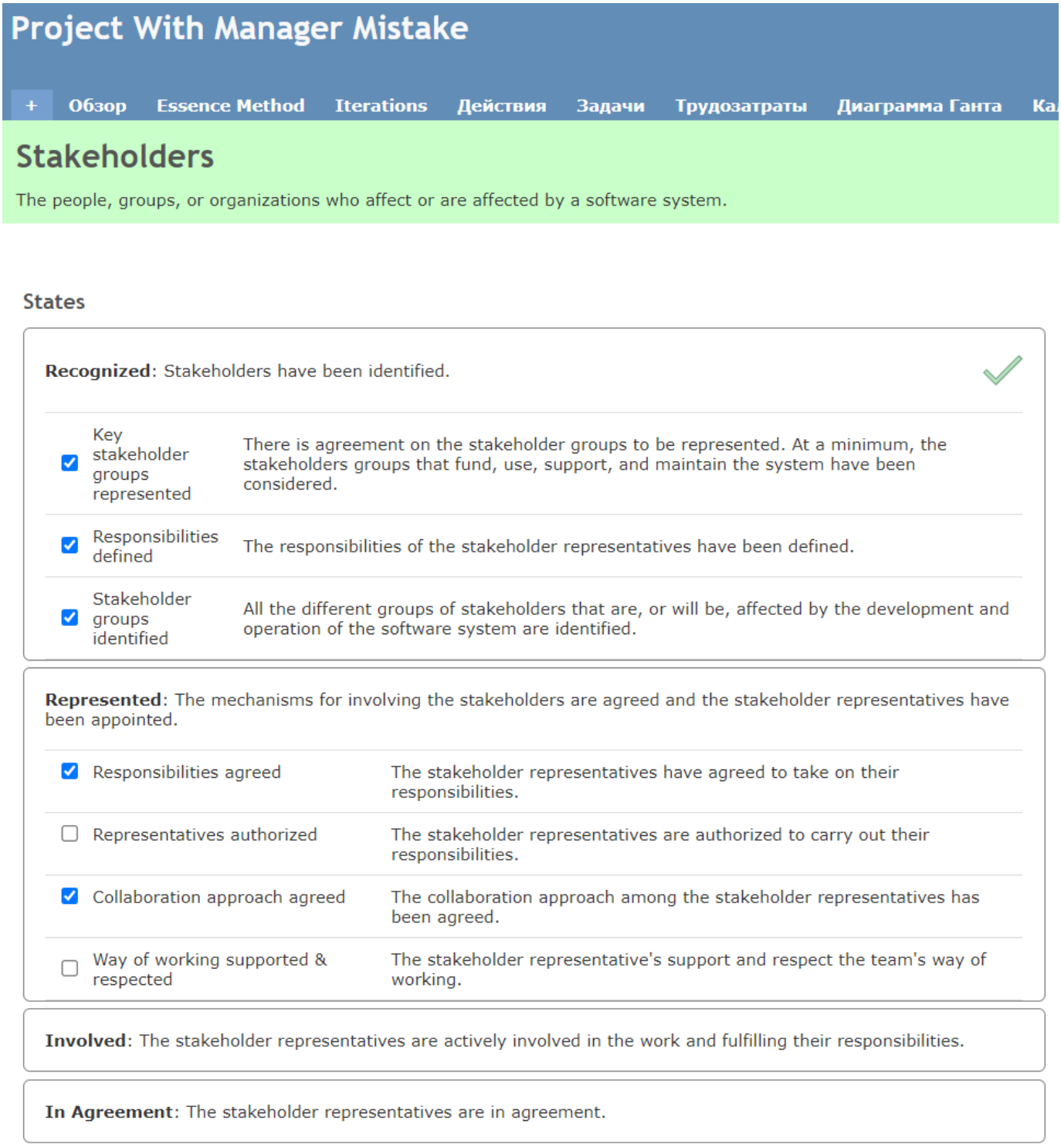


Рисунок 58 – Скриншот ALPHы Stakeholders

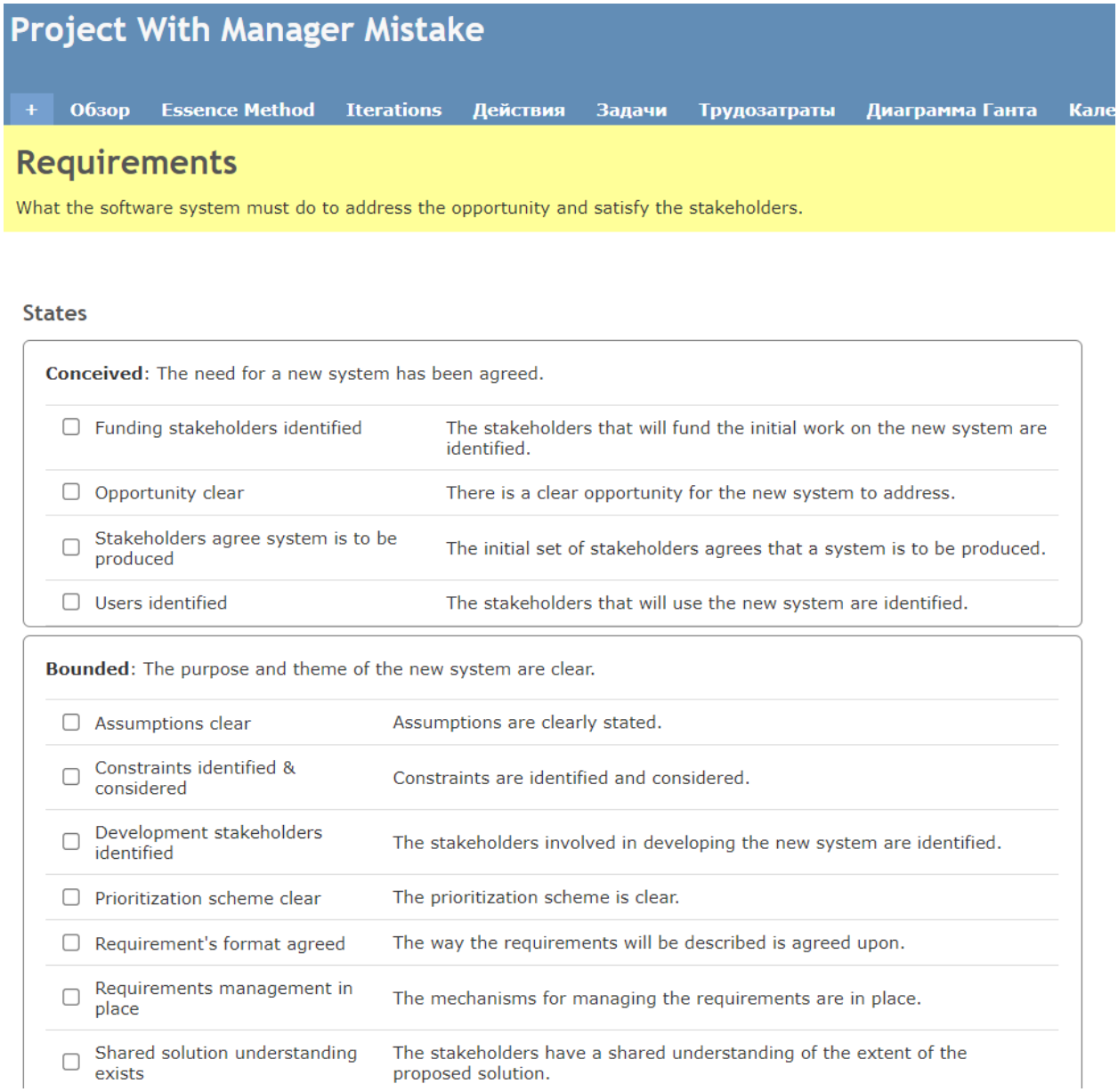


Рисунок 59 – Скриншот ALPHы Requirements

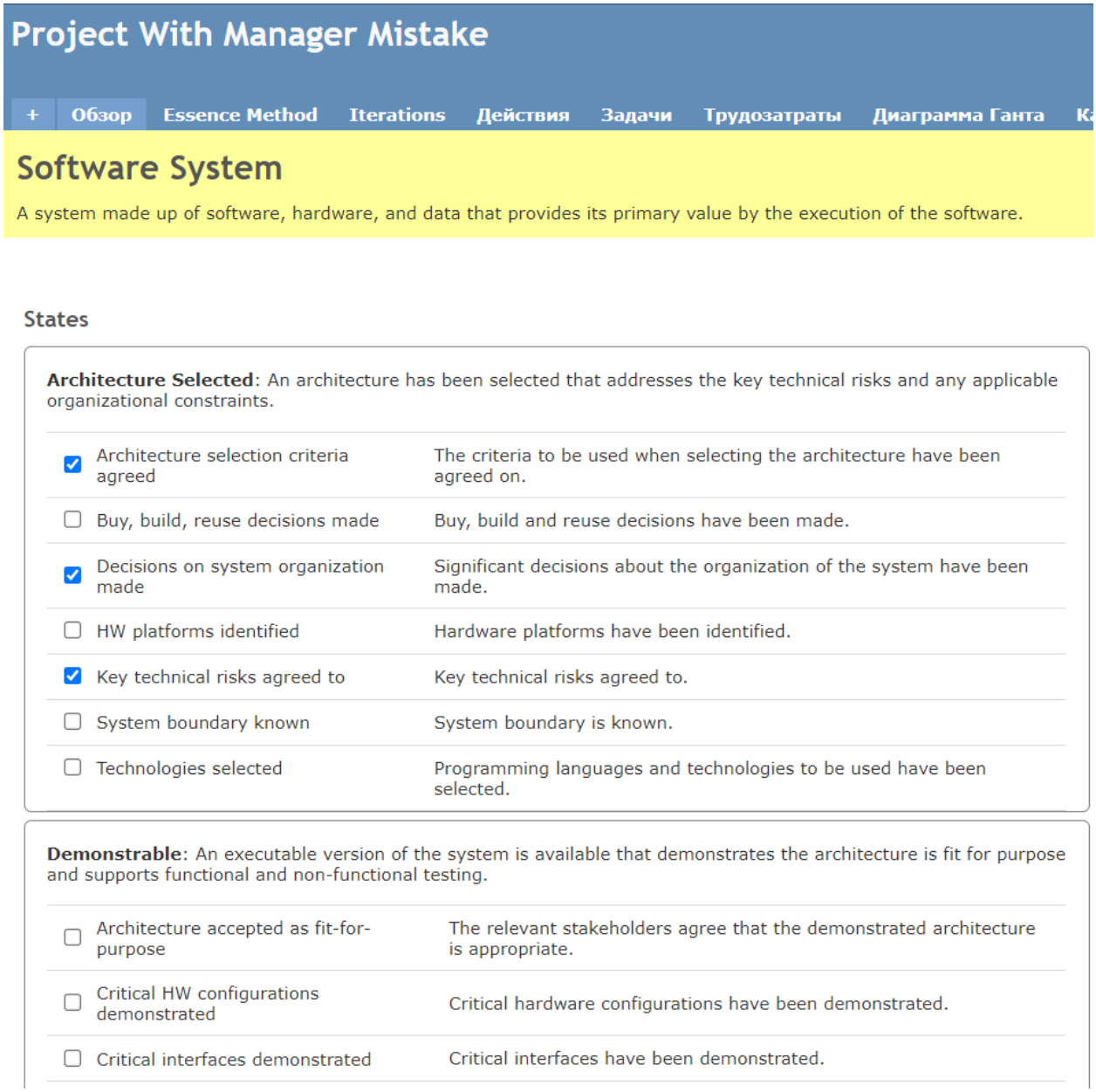


Рисунок 60 – Скриншот ALPHы Software System

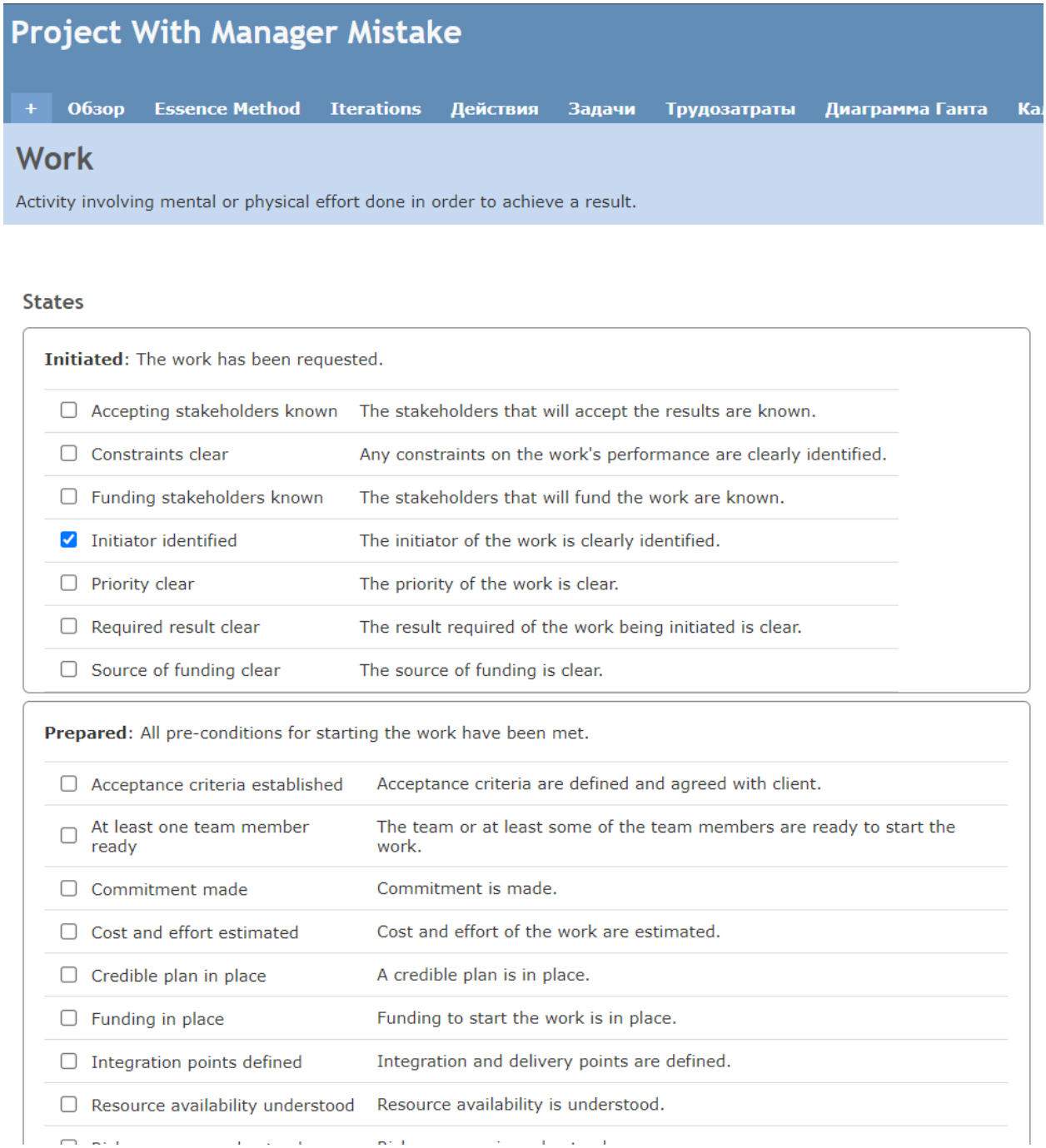


Рисунок 61 – Скриншот ALPHы Work

Перечислим только те утверждения, которые менеджер предположительно посчитал истинными:

– O11, O12, O13, что в общем можно проинтерпретировать как «идентифицированы перспективы стейкхолдеров от этого проекта»;

– S11, S12, S13 – «стейкхолдеры идентифицированы»;

– S21, S23 – «договорились о способах коллаборации со стейкхолдерами и о разделении обязанностей между всеми участниками соответственно»;

– W14 – «известен инициатор проекта»;

– SS11, SS13, SS15 – «определены критерии выбора архитектуры», «принято решение о высокоуровневой организации архитектуры» и «договорились об основных технических рисках».

Все остальные утверждения менеджер считает ложными. В целом данная ситуация описывает одну из типовых ошибок начинающих менеджеров, когда, не получив формальное описание и подтверждение даже начальных требований к системе, а только лишь представление об основных пожеланиях заказчиков, команда начинает проектировать архитектуру. В этой ситуации система поддержки принятия решений должна явно указать, что считать истинными начальные утверждения из Alpha «Software System» – ошибка. Зададим начальные вероятности всех утверждений быть истинными равными нулю, поскольку это начало нового проекта, а пороговое значение для ложноположительных ошибок в 0,5. В результате работы сети мы получим следующие ложноположительные ошибки менеджера с указанием вероятности этих утверждений быть истинными (рисунок 62).

Probabilities

Node	Probability
SS13	0.2565476057
SS15	0.3367743159
SS11	0.3562560269
S21	0.4036519801

Рисунок 62 – Результат поиска ложноположительных ошибок менеджера

Ошибки SS11, SS13, SS15 – легко объяснимы, к тому же они и предполагались как результат работы данного подхода, т. к. в общей логике управления проектами по разработке программного обеспечения рискованно проектировать даже высокоуровневую архитектуру, не получив хотя бы общее представление о требованиях. Довольно интересен вопрос об S21, если быть точнее – то, что менеджер посчитал, что стейкхолдеры договорились о принятии на себя обязательств, но с точки зрения теоретического контура Essence стейкхолдеры могут принять на себя обязательства только в том случае, если понятно, зачем они это делают, т. е. Opportunity достигает третьего состояния, а в данном примере все утверждения в этой альфе остаются только на первом состоянии.

Теперь смоделируем ситуацию, в которой менеджер решил исправить свои ошибки, поработать с бизнес-требованиями, а также начать процесс формализации требований к программному продукту. В этом случае он должен организовать такой набор задач, в результате выполнения которых следующие утверждения будут считаться истинными:

– O11, O12, O13, O21, O22, O23, O24 – «выполнены первые два состояния альфы Opportunities»;

– S11, S12, S13, S21, S22, S23, S24 – «выполнены первые два состояния альфы Stakeholders»;

– W14 – «известен инициатор проекта»;

– R11, R12, R13, R14, R21, R22, R23, R27, R28 – «выполнено первое состояние альфы Reqiurments», а также «предположения ясны», «ограничения зафиксированы», «определены стейкхолдеры, вовлеченные в разработку», «есть общее понимание предложенного решения», «назначение системы согласовано»;

– SS11, SS13, SS15 – «определены критерии выбора архитектуры», «принято решение о высокоуровневой организации архитектуры» и «договорились об основных технических рисках».

В таком случае тот же алгоритм, запущенный при том же начальном распределении вероятностей с той же пороговой величиной ошибки в 0,5 на следующей итерации, пересчитает вероятности для предыдущих «проблемных» вершин уже следующим образом (рисунок 63).

Probabilities

Node	Probability
SS11	0.6014091942
S21	0.5250864167
SS13	0.3816220938
SS15	0.4640762696

Рисунок 63 – Результат пересчета вероятности утверждений после внесенных изменений

SS11 получает вероятность истинности большую, чем пороговое значение, и больше не считается ошибкой, т. к. при начальном определении требований более разумно утверждение, что определены принципы выбора архитектуры, чем в том случае, если требований к программной системе нет. S21 тоже перестает быть ошибкой, т. е. более вероятно, что стейкхолдеры соглашаются со своими обязательствами, когда их выгода в форме Opportunities становится более формализованной, что и позволяет S21 получить вероятность истинности чуть выше, чем пороговое

значение. Возросли вероятности SS13 и SS15 быть истинными, т. к. требования действительно начали прорабатываться, но недостаточно подробно, чтобы более детальные, чем выбор общих принципов, решения можно было признать истинными[46].

Выводы главы

Несмотря на то что предложенный анализ текущего прогресса академического сообщества в решении задачи оптимизации управления проектами показывает достаточно пессимистичную картину, потенциально импульс к оптимизации управления проектами кроется в математических концепциях и теориях, относящихся к отраслям, связанным с условиями неопределенности, хаоса, нарушения строгих причинно-следственных связей, информационной энтропии, или – в поиске источника данных и датасетов, которые позволят обучить следующее поколение интеллектуальных систем для решения довольно творческой и неопределенной задачи, коей является управление проектами по разработке программного обеспечения. Однако представленные в данной работе математическая модель в форме динамической байесовской сети и метод поиска ложноположительных ошибок наглядно доказывают, что дальнейшие исследования в этом направлении имеют потенциал для достижения весьма эффективных результатов, в связи с чем можно считать, что текущее состояние математического обеспечения удовлетворяет основным требованиям к эволюционному прототипу.

Примечание 1. У описанного выше подхода можно отметить один существенный недостаток, связанный с тем, что и связи, и их оценка во избежание избыточной субъективности полученных результатов производились экспертно. Найденные связи и оценка связей были обсуждены вместе с ведущими разработчиками и проектными менеджерами компаний «Education ERP», «Kreosoft», «CODE» и «Сибирские Информационные Системы».

Примечание 2. Необходимо отметить, что файл со структурированной байесовской сетью и подсчитанными вероятностями [108] имеет структуру, определенную приложением GeNIe Academic, которая была использована для построения байесовских сетей. Из достаточно принципиальной разницы можно отметить только то, что вероятностный вектор, который используется в самом файле, немного отличается от подразумевающегося в работе. Разница заключается в том, что в файле вероятностный вектор для каждой вершины предусматривает также

[46] С точки зрения теоретического контура Essence, прежде чем садиться за Software System даже на уровне проектирования архитектуры, Requirements должны достигнуть третьего состояния.

элементы, которые содержат значение вероятности быть ложной при той же комбинации вершин-родителей, но, поскольку вероятность вершины быть истинной при определенной комбинации вершин-родителей и вероятность вершины быть ложной при той же комбинации вершин-родителей – вычисляемые одно из другого значения, дополнительные элементы в вероятностном векторе не упоминались в алгоритмах, т. к. несущественны для них. Для повтора результата достаточно из каждого вероятностного вектора оставить только нечетные элементы.

ЗАКЛЮЧЕНИЕ

В рамках данной диссертационной работы выполнено исследование, посвященное прикладным методам для помощи проектному менеджеру в проектах по разработке программного обеспечения на основе формального описания используемого процесса разработки.

Основное внимание в ходе исследования сосредоточено на новых возможностях, предлагаемых, с одной стороны, языком и практиками Essence как одним из последних достижений профессионального сообщества, с другой стороны, развитием разных подходов к более нестандартному решению задач, предлагаемых, в частности, байесовскими сетями. В работе произведен анализ как потенциальных проблем, связанных с текущим состоянием Essence и поддерживающих его инструментов, так и потенциальных проблем, связанных с поиском математических моделей и данных для применения к управлению проектами по разработке программного обеспечения.

Важнейшие достигнутые результаты сводятся к следующему:

1) предложена промежуточная модель представления процесса разработки программного обеспечения, формализованная средствами стандарта OMG Essence для практического использования в управлении проектами в области программной инженерии;

2) на основе предложенной промежуточной модели реализована процедура переноса практик и методов Essence в программные средства управления проектами;

3) построена оригинальная семантическая модель зависимостей между различными элементами ядра стандарта OMG Essence;

4) разработана математическая модель для решения прикладных задач в управлении проектами по разработке программного обеспечения на основе стандарта OMG Essence в виде динамической байесовской сети;

5) на основе предложенной математической модели решена задача поиска ложноположительных ошибок менеджмента проекта в рамках теоретического контура методов Essence.

Решение отдельных вопросов, не упомянутых в тексте диссертации, можно найти в публикациях автора диссертации [109, 110, 111, 112, 113, 114, 115].

Анализ достигнутых результатов позволяет предположить несколько различных векторов развития данного исследования, включающего в себя следующие потенциальные направления:

1) развитие в области расширений для сред управления проектами

a) разработка плагинов для дополнительных сред управления проектами,

b) улучшение UX разработанных решений,

c) добавление дополнительных функциональных возможностей, связанных с использованием Essence,

d) развитие интеграционных возможностей (подключение систем управления версиями, а также решениями, используемыми для CI/CD);

2) более детальное расширение применимости байесовских сетей к управлению проектами

a) за счет расширения языка описания практик Essence,

b) за счет анализа доступных в проекте артефактов и классификации их в качестве рабочих продуктов Essence,

c) построение байесовской сети не только на основе Essence, но также и других существующих в профессиональном сообществе стандартов – например PMBOK;

3) поиск дополнительных математических методов, которые были бы применимы к вопросам оптимизации управления проектами.

В целом несмотря на дальнейшую возможность расширения и развития полученных результатов, цель данного исследования достигнута и задачи выполнены.

СПИСОК ЛИТЕРАТУРЫ

1. Фаулер М. Архитектура корпоративных программных приложений. 2-е издание. М. : Издательский дом «Вильямс», 2006. 544 с.

2. CS302: Jared King's "The History of Software" // learn.saylor.org [website]. URL: https://learn.saylor.org/mod/page/view.php?id=12353 (дата обращения: 16.05.2021).

3. Воловик В. В., Щедровицкий П. Г. Конструктивное мышление: неучтенный фактор развития // Вопросы философии. 2018. № 9. С. 39–49.

4. Кун Т. Структура научных революций. М., 2009. 310 с.

5. Johnson P., Ekstedt M., Jacobson I. Where’s the Theory for Software Engineering? // IEEE Software. 2012. Vol. 29. Is. 5. P. 94–96. DOI: 10.1109/MS.2012.127.

6. Wojewoda S., Hastie S. Standish Group 2015 Chaos Report – Q&A with Jennifer Lynch // www.infoq.com [website]. URL: https://www.infoq.com/articles/standish-chaos-2015/ (дата обращения: 16.05.2021).

7. Rico D. F. Global Project Failures [Электронный ресурс]. 2010. URL: http://davidfrico.com/project-failures.pdf (дата обращения: 16.05.2021).

8. Тимофеев А. Н. Почему падают ИТ-проекты? [Электронный ресурс] // Практика проектирования систем. 2017. URL: http://reqcenter.pro/wp-content/uploads/pps/pps2017.pdf (дата обращения: 16.05.2021).

9. The Standish Group – CHAOS Report [Электронный ресурс] // Project Smart. URL: https://www.projectsmart.co.uk/white-papers/chaos-report.pdf (дата обращения: 01.11.2017).

10. Ishaque M., Zaidi A. K., Levis A. H. Project management using point graphs // Systems Engineering. 2009. Vol. 12. Is. 1. P. 36–54. DOI: 10.1002/sys.20107.

11. Project Scheduling Conflict Identification and Resolution Using Genetic Algorithms / M. Ramzan, M. A. Iqbal, M. A. Jaffar, A. Rauf, S. Anwar, A. A. Shahid // 2010 International Conference on Information Science and Applications. 2010. Vol. 1. P. 1–6. DOI: 10.1109/ICISA.2010.5480400.

12. A Mathematical Approach of Work Assignment for Human Resource in Software Development / X. Chen, S.-J. Lee, S.-C. Seo, B.-K. Kim // Jouranl of Digital Convergence. 2013. Vol. 11. Is. 2. P. 205–214. DOI: 10.14400/JDPM.2013.11.2.205.

13. Alba E., Chicano J. F. Management of software projects with GAs // MIC2005: The Sixth Metaheuristics International Conference. Viena, 2005. P. 13–18.

14. Klein R. Scheduling of Resource-Constrained Projects. Operations Research/Computer Science Interfaces Series. Springer US, 2000. ISBN 978-1-4613-7093-2. DOI 10.1007/978-1-4615-4629-0.

15. Ancveire I., Polaka I. Application of Genetic Algorithms for Decision-Making in Project Management: A Literature Review // Information Technology and Management Science. 2019. Vol. 22. P. 22–31. DOI: 10.7250/itms-2019-0004.

16. Wang Y. Software Science: On the General Mathematical Models and Formal Properties of Software // Journal of Advanced Mathematics and Applications. 2014. Vol. 3. № 2. P. 130–147. DOI: 10.1166/jama.2014.1060.

17. A statistical approach for prediction of projects based on simulation / M. M. de Souza, H. C. B. de Oliveira, A. M. L. de Vasconcelos, S. R. B. Oliveira // Proceedings of the 2008 ACM Symposium on Applied Computing (SAC). 2008. P. 23–27. DOI: 10.1145/1363686.1363693.

18. Miguel A., Madria W., Polancos R. Project Management Model: Integrating Earned Schedule, Quality, and Risk in Earned Value Management // 2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA). 2019. P. 622–628. DOI: 10.1109/IEA.2019.8714979.

19. Sackey S., Lee D.-E., Kim B.-S. Duration Estimate at Completion: Improving Earned Value Management Forecasting Accuracy // KSCE Journal of Civil Engineering. 2020. Vol. 24. № 3. P. 693–702. DOI: 10.1007/s12205-020-0407-5.

20. Soltan S., Ashrafi M. Predicting project duration and cost, and selecting the best action plan using statistical methods for earned value management // Journal of Project Management. 2020. № 5. P. 157–166. DOI: 10.5267/j.jpm.2020.3.002.

21. Lipke W. Earned Schedule Forecasting Method Selection [Электронный ресурс] // PM World Journal. 2019. Vol. VIII. Is. 1. P. 1–15. URL: https://pmworldlibrary.net/wp-content/uploads/2019/01/pmwj78-Jan2019-Lipke-earned-schedule-forecasting-method-selection.pdf (дата обращения: 17.05.2021).

22. Aplicação de Métodos Estatísticos em Engenharia de Software: Teoria e Prática / T. F. M. Sirqueira, M. A. Miguel, H. L. de Oliveira Dalpra., M. A. P. Araújo // Engenharia no Século XXI. 2020. Vol. 18. P. 228–246. DOI: 10.36229/978-65-86127-82-9.CAP.24.

23. Vaid K., Ghose U. Predictive Analysis of Manpower Requirements in Scrum Projects Using Regression Techniques // Procedia Computer Science. 2020 Vol. 173. P. 335–344. DOI: 10.1016/j.procs.2020.06.039.

24. Statistical Analysis of the Effects of Heavyweight and Lightweight Methodologies on the Six-Pointed Star Model / M. A. Akbar, J. Sang, A. A. Khan, F.-E-Amin, Nasrullah, S. Hussain, M. K. Sohail, H. Xiang, B. Cai // IEEE Access. 2018. Vol. 6. P. 8066–8079. DOI: 10.1109/ACCESS.2018.2805702.

25. Buch I., Pokiya K. Project Management using Machine Learning. 2020. DOI: 10.13140/RG.2.2.23066.06085.

26. A genetic algorithms based approach for conflicts resolution in requirement / M. Ramzan, M. Q. Khan, M. A. Iqbal, M. Aasem, A. Jaffar, S. Anwar, A. Adnan, A. Tamleek, M. Ali, M. A. Alam // International Journal of the Physical Sciences. 2011. Vol. 6. № 4. P. 828–836. DOI: 10.5897/IJPS10.623.

27. A Hybrid ACO Algorithm for the Next Release Problem / H. Jiang, J. Zhang, J. Xuan, Z. Ren, Y. Hu // Proceedings of 2nd International Conference on Software Engineering and Data Mining (SEDM 2010). 2010. P. 166–171. DOI: arXiv:1704.04777.

28. Souza R. G. M., Stadzisz P. C. Problem-Based Software Requirements Specification (SRS) // Revista Eletrônica de Sistemas de Informação. 2016. Vol. 15. № 2. P. 1–25. DOI: 10.21529/RESI.2016.1502002.

29. del Sagrado J., del Águila I. M. Stability prediction of the software requirements specification // Software Quality Journal. 2018. Vol. 26. Is. 2. P. 585–605. DOI: 10.1007/s11219-017-9362-x.

30. Addala S. SRS – Software Requirements Specification for SNAPCHAT. Lovely Professional University, 2019. 40 p. DOI: 10.13140/RG.2.2.33860.24965.

31. Gupta A. K., Deraman A., Siddiqui S. T. A Survey of Software Requirements Specification Ambiguity // ARPN Journal of Engineering and Applied Sciences. 2019. Vol. 14. № 17. P. 3046–3061.

32. Guarnieri S. A., Tripp O., Pistoia M. Differential static program analysis // Patent Application Publication № US 2014/0115563 A1. 2014.

33. Gopinath K. Static Program Analysis for Security. Bangalore : Indian Institute of Science, 2014.

34. Cousot P., Cousot R. Modular Static Program Analysis // Lecture Notes in Computer Science. 2002. Vol. 2304. P. 159–179. DOI: 10.1007/3-540-45937-5_13.

35. Arceri V., Mastroeni I. Static Program Analysis for String Manipulation Languages // Electronic Proceedings in Theoretical Computer Science. 2019. Vol. 299. P. 19–33. DOI: 10.4204/EPTCS.299.5.

36. Jiang Y., Li M., Zhou Z.-H. Software defect detection with Rocus // Journal of Computer Science and Technology. 2011. Vol. 26. Is. 2. P. 328–342. DOI: 10.1007/s11390-011-9439-0.

37. Hall R. J. Editorial: Software defect detection // Automated Software Engineering. 2010. Vol. 17. Is. 3. P. 213–215. DOI: 17. 213-215. 10.1007/s10515-010-0071-y.

38. Loubser N. Hosting and CI/CD // Software Engineering for Absolute Beginners. 2021. P. 313–324. ISBN 978-1-4842-6622-9. DOI: 10.1007/978-1-4842-6622-9_11.

39. Vadavalasa R. M. End to end CI/CD pipeline for Machine Learning // International Journal of Advance Research, Ideas and Innovation in Technology. 2020.Vol. 6. Is. 3. P. 906–913.

40. Liao Q. Modelling CI/CD Pipeline Through Agent-Based Simulation // IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). 2020. Vol. 1. P. 155–156. DOI: 10.1109/ISSREW51248.2020.00059.

41. Parsons D. Agile software development methodology, an ontological analysis 2010. DOI: 10.13140/2.1.3298.6883.

42. Sheeba T., Krishnan R., Bernard M. J. An Ontology in Project Management Knowledge Domain // International Journal of Computer Applications. 2012. Vol. 56. № 5. P. 1–7. DOI: 10.5120/8884-2881.

43. Parsons D. An ontology of agile aspect oriented software development // Research Letters in the Information and Mathematical Sciences. 2011. Vol. 15. P. 1–11.

44. De Nicola A., Missikoff M., Navigli R. A software engineering approach to ontology building // Information Systems. 2009. Vol. 34. Is. 2. P. 258–275. DOI: 10.1016/j.is.2008.07.002.

45. Barcellos M. P., de Almeida Falbo R. A Software Measurement Task Ontology // SAC '13 Proceedings of the 28th Annual ACM Symposium on Applied Computing. 2013. Vol. 1. P. 311–318. DOI: 10.1145/2480362.2480428.

46. Barcellos M. P., de Almeida Faldo R., Rocha A. R. A Well-founded Software Process Behavior Ontology to Support Business Goals Monitoring in High Maturity Software Organizations // EDOCW '10 Proceedings of the 2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops. 2010. P. 253–262. DOI: 10.1109/EDOCW.2010.15.

47. de Oliveira Arantes L. R. de Almeida Falbo, G. Guizzardi. Evolving a Software Configuration Management Ontology // WOMSDE II Workshop on Ontologies and Metamodeling in Software and Data Engineering. 2007. P. 123–134.

48. Werewka J., Szwed P., Rogus G. Integration of classical and agile project management methodologies based on ontological models // Production engineering in making. Krakow: AGH University of Science and Technology Press, 2010. P. 7–28.

49. Siddiqui F., Alam M. A. Ontology Based Feature Driven Development Life Cycle // IJCSI International Journal of Computer Science Issues. 2012. Vol. 9. Is. 1. № 2. P. 207–213.

50. Biffl S., Sunindyo W. D., Moser T. Semantic Integration of Heterogeneous Data Sources for Monitoring Frequent-Release Software Projects // International Conference on Complex, Intelligent and Software Intensive Systems. 2010. Vol. 1. P. 360–367. DOI: 10.1109/CISIS.2010.58.

51. Dippelreiter B. Semantic based Project Management: dissertation for the degree of Doctor of Social and Economic Sciences. Vienna University of Technology, 2012. 142 p.

52. de Almeida Falbo R., Ruy F. B., Dal Moro R. Using Ontologies to Add Semantics to a Software Engineering Environment // Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'2005). 2005. P. 151–156.

53. Fitsilis P. Comparing PMBOK and Agile Project Management software development processes // Advances in Computer and Information Sciences and Engineering, Proceedings of the International Conference on Systems, Computing Sciences and Software Engineering (SCSS), CISSE. 2007. Vol. 1. P. 378–383. DOI: 10.1007/978-1-4020-8741-7_68.

54. Usman M., Soomro T. R., Brohi M. N. Embedding project management into XP, Scrum and RUP // European Scientific Journal. 2014. Vol. 10. № 15. P. 293–307.

55. Sutherland J., Ahmad N. How a Traditional Project Manager Transforms to Scrum: PMBOK vs. Scrum [presentation] // Agile Conference. Salt Lake City, 2011.

56. Rebaiaia M.-L., Vieira D. R. Integrating PMBOK Standarts, Lean and Agile Methods in Project Management Activities // International Journal of Computer Applications. 2014. Vol. 88. № 4 P. 40–46. DOI: 10.5120/15343-3680.

57. Callegari D. A., Bastos R. M. Project Management and Software Development Processes: Integrating RUP and PMBOK // 7th International Conference on Systems Engineering and Modeling (ISDA). 2007. P. 1–8. DOI: 10.1109/ICSEM.2007.373327.

58. Rosito M. C. Callegari D. A., Bastos R. M Project Management and Software Development Processes: Integrating PMBOK and OPEN // International Journal of Computer, Electrical, Automation, Control and Information Engineering. 2012. Vol. 6. № 2. P. 182–190.

59. Rosito M. C., Bastos R. M. SPIM: An Integrated Model of Software Project Management and Organizational Workflows // International Journal of Computer Information Systems and Industrial Management Applications. 2014. Vol. 6. P. 160–170.

60. Griffiths M. Using Agile Alongside the PMBOK // PMI Global Congress Proceedings. Anaheim, 2004. P. 1–8.

61. Rosito M. C., Bastos R. M. A model to integrate software project management with organizational workflows // 12th International Conference on Intelligent Systems Design and Applications (ISDA). 2012. P. 40–45. DOI: 10.1109/ISDA.2012.6416510.

62. Troughton R. Scrum Evolution Over Time: Part 2 – Roles // Agile Forest [website]. 2012. URL: https://agileforest.com/2012/02/26/scrum-evolution-over-time-part-2-roles/ (дата обращения: 06.06.2021).

63. SEMAT вчера, сегодня и завтра: перспективы промышленного использования / Дж. С. Пак, И. Якобсон, Б. Майбург, П. Джонсон // Программная инженерия. 2014. № 11. С. 6–16.

64. Jacobson I., Meyer B., Soley R. Software engineering method and theory – a vision statement [Электронный ресурс] // SEMAT [website]. 20 p. URL: http://semat.org/documents/20181/27952/SEMAT-vision.pdf (дата обращения: 06.06.2021).

65. The Essence of Software Engineering: Applying the SEMAT Kernel / I. Jacobson, P.-W. Ng, P. E. McMahon, I. Spence, S. Lidman. Addison-Wesley, 2013. 224 p. ISBN 978-0321885951.

66. Essence – Kernel and Language for Software Engineering Methods Ver 1.0: Specification [Электронный ресурс] // Object Management Group [website]. 2014. URL: https://www.omg.org/spec/Essence/1.2/PDF (дата обращения: 06.06.2021).

67. Позин Б. А. SEMAT – Software Engineering Method and Theory. О чем, зачем и кому это нужно? // Программная инженерия. 2014. № 11. С. 3–5.

68. Левенчук А. И. Основа – сущности и язык для методов программной инженерии (OMG Essence) // Живой журнал [сайт]. 2012. URL: https://ailev.livejournal.com/1051048.html (дата обращения: 06.06.2021).

69. P. Tell et al. Towards the statistical construction of hybrid development methods // Journal of Software: Evolution and Process. 2020. Vol. 33. Is. 1. Special Issue: ICSSP 2019. 20 p. DOI: 10.1002/smr.2315.

70. C. M. Zapata-Jaramillo, G. V. Maturana Gonzalez, J. M. Calle-Gallego. A Board Game to Simulate the Software Development Process Based on the SEMAT Essence Standard // 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T). 2020. P 1–4. DOI: 10.1109/CSEET49119.2020.9206177.

71. Jacobson I., Spence I., Seidewitz E. Industrial Scale Agile – from Craft to Engineering // Queue. 2016. Vol. 14. Is. 5. P. 99–130. DOI: 10.1145/3012426.3012428.

72. Jacobson I., Spence I., Bittner K. USE-CASE 2.0: The Guide to Succeeding with Use Case. Ivar Jacobson International [publisher], 2011. 55 p.

73. Scrum Essential Cards Help Introduce and Apply Scrum // Ivar Jacobson International [website]. URL: https://pages.services/ss.ivarjacobson.com/essential-scrum (дата обращения: 06.06.2021).

74. Fowler M. SEMAT // martinFowler.com [website]. 2010. URL: http://martinfowler.com/bliki/Semat.html (дата обращения: 06.06.2021).

75. Cockburn A. A Detailed Critique of the SEMAT Initiative [electronic data]. URL: http://alistair.cockburn.us/A+Detailed+Critique+of+the+SEMAT+Initiative (дата обращения: 06.06.2021).

76. Aranda J. Against SEMAT // Catenary (Jorge Aranda's blog) [website]. 2009. URL: http://catenary.wordpress.com/2009/11/29/against-semat/ (дата обращения: 06.06.2021).

77. Zapata C., Jacobson I. A First Course in Software Engineering Methods and Theory // Dyna. 2014. Vol. 81. № 183. P. 231–241. DOI: 10.15446/dyna.v81n183.42293.

78. Позин Б. А., Горбунова Е. Развитие базовой модели SEMAT для жизненного цикла заказных ответственных программных систем // Материалы четвертой Научно-практической конференции «Актуальные проблемы системной и программной инженерии»: Сборник трудов. М. : НИУ ВШЭ, 2015. С. 170–171.

79. UNI-DUE Essence Model Viewer 1.0 [Электронный ресурс] URL: https://www.s3.uni-duisburg-essen.de/pub/essence/demo/Wobbleboard.html (дата обращения: 06.06.2021)

80. Essence Enterprise 365 // Essence in Practice [website]. URL: https://www.ivarjacobson.com/essence-enterprise (дата обращения: 06.06.2021). Режим доступа: по запросу.

81. SEMAT Essence Kernel Tool [website]. URL: https://semat.herokuapp.com/ (дата обращения: 06.06.2021).

82. Graziotin D., Abrahamsson P. A Web-based modeling tool for the SEMAT Essence theory of software engineering // Journal of Open Research Software. 2013. Vol. 1. Is. 1. P. e4. DOI: 10.5334/jors.ad.

83. Agile Practices Workbench | Agile Development Tools // Ivar Jacobson International [website]. URL: https://www.ivarjacobson.com/esswork-practice-workbench (дата обращения: 06.06.2021).

84. OMG Meta Object Facility (MOF) Core Specification Ver. 2.5.1: Specification [Электронный ресурс] // Object Management Group [website]. 2019. URL: https://www.omg.org/spec/MOF/2.5.1/PDF (дата обращения: 06.06.2021).

85. Extensible Markup Language (XML) 1.0 (Fifth Edition) // World Wide Web Consortium (W3C) [website]. 2008. URL: https://www.w3.org/TR/xml/ (дата обращения: 06.06.2021).

86. Geisler R., Klar M., Pons C. F. Dimensions and Dichotomy in Metamodeling: Technical Report № 98-5. Berlin : TU Berlin, 1998. 1–31 p.

87. The Essentials of Modern Software Engineering / I. Jacobson, H. B. Lawson, P.-W. Ng, P. E. McMahon, M. Goedicke. ACM books, 2019. 371 p. ISBN 978-1947487246.

88. Practice Library [website]. URL: https://practicelibrary.ivarjacobson.com (дата обращения: 06.06.2021).

89. Змеев О. А., Змеев Д. О., Даниленко А. Н. Перенос практик Essence в среду Azure DevOps Server // Программная инженерия. 2020. Т. 11. № 6. С. 311–321. DOI: 10.17587/prin.11.311-321.

90. Zmeev D., Zmeev O., Tamazlykar D. Implementation of Essence Practice into project management system Redmine // 2019 Actual Problems of Systems and Software Engineering (APSSE 2019). 2019. P. 116–125. DOI: 10.1109/APSSE47353.2019.00022.

91. Model-Driven Development: Processes and Practices / P. Parviainen, J. Takalo, S. Teppola, M. Tihinen // VTT Working Papers 114. VTT Technical Research Centre of Finland, 2009. 108 p. ISBN: 978-951-38-7175-8.

92. Буч Г., Рамбо Д., Якобсон А. Язык UML. Руководство пользователя. 2-е изд.: пер. с англ. Мухина Н. М. М. : ДМК Пресс, 2006. 496 с.

93. Пример файла с элементами метода в формате промежуточной модели [электронные данные]. URL: https://drive.google.com/file/d/11Cg-OFkZw_EJhuVedAmftDRRFk7yu0se/view (дата обращения: 16.05.2021).

94. Redmine [website]. URL: https://www.redmine.org/ (дата обращения: 16.05.2021).

95. 8 толковых книг по Ruby на русском языке // Библиотека программиста [сайт]. URL: https://progerlib.ru/ruby-books (дата обращения: 16.05.2021).

96. Patterns of Enterprise Application Architecture / M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, R. Stafford. Addison Wesley Professional, 2002. 560 p. ISBN: 978-0321127426.

97. Крейн Д., Паскарелло Э., Джеймс Д. AJAX в действии / пер. с англ. В. В. Вейтмана и А. В. Назаренко. М. : Издательский дом «Вильямс», 2006. 640 с. ISBN 978-5-8459-1034-X.

98. Statistical Software Engineering. Washington, D. C. : National Academy Press, 1996. 84 p. ISBN: 978-0309053440.

99. Змеев Д. О., Дмитриев Ю. Г. Применение статистических оценок в управлении проектами по разработке программного обеспечения // Труды Томского государственного университета. Т. 305. Серия физико-математическая: Математическое и программное обеспечение информационных, технических и экономических систем: материалы Международной научной конференции. Томск, 28–30 мая 2020 г. / под общ. ред. И. С. Шмырина. Томск : Издательство Томского государственного университета, 2020. С. 211–216.

100. Dmitriev Y. G., Zmeev D. O. On a Combined Estimator of Probabilistic Characteristics // Eighth International Conference on Risk Analysis and Design of Experiments in Vienna, 2019: book of abstracts / edited by Moder K. and Spangl B. Vienna : University of Natural Resources and Life Sciences, 2019. P. 147–148.

101. Pearl J. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. San Francisco : Morgan Kaufmann Publishers, Inc., 1988. 552 p. ISBN: 978-1-55860-479-7.

102. Darwiche A. Modeling and Reasoning with Bayesian Networks. Cambridge : Cambridge University Press, 2009. 562 p. ISBN 978-0521884389.

103. Beyond Knowledge Tracing: Modeling Skill Topologies with Bayesian Networks / T. Käser, S. Klingler, A. Schwing, M. Gross // 12th International Conference Intelligent Tutoring Systems (ITS 2014), Lecture Notes in Computer Science. Vol. 8474. P. 188–198. DOI: 10.1007/978-3-319-07221-0_23.

104. David Y. B., Segal A., Gal Y. Sequencing educational content in classrooms using Bayesian knowledge tracing // Proceedings of the Sixth International Conference on Learning Analytics & Knowledge (LAK'16). New York : Association for Computing Machinery, 2016. P. 354–363. DOI: 10.1145/2883851.2883885

105. Alpha State Card Games: Instructional Guide [electronic resource]. Ivar Jacobson International [publisher], 2015. 10 p. URL: https://www.ivarjacobson.com/publications/brochure/alpha-state-card-games (дата обращения: 06.06.2021). Режим доступа: по запросу.

106. Pieper J. Simulation and Digital Game-Based Learning in Software Engineering Education: An Integrated Approach to Learn Engineering Methods: dissertation for the degree of Doctor of Engineering. Rostock : University of Rostock, 2016. 412 p.

107. Семантический анализ связей всех состояний ALPHa из ядра OMG Essence URL: https://docs.google.com/spreadsheets/d/11Et20mr830kRYb88YZk8o9ePMZdE3gMdpBAf-YKZfG4/edit#gid=1370754939 (дата обращения: 16.05.2021).

108. Файл со структурированной байесовской сетью и подсчитанными вероятностями [электронные данные]. URL: https://drive.google.com/file/d/1ez6L5EybUUpW8FTv_n3XD9ziwPMfXXAu/view (дата обращения: 16.05.2021).

109. Змеев Д., Иванова Л., Рафикова Р. О представлении прогресса проекта по разработке программного обеспечения в форме динамической байесовской сети // Информационные технологии и математическое моделирование (ИТММ-2020): Материалы XIX Международной конференции имени А. Ф. Терпугова (2−5 декабря 2020 г.). – Томск : Изд-во НТЛ, 2021. С. 291–297.

110. Zmeev D. O., Zmeev O. A. Project-oriented Course of Software Engineering Based on Essence // 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T). 2020. P. 296–298. DOI: 10.1109/CSEET49119.2020.9206240.

111. Импорт модели SEMAT Essence Practice Workbench в среду управления проектами и задачами Redmine / А. Н. Даниленко, Д. О. Змеев, О. А. Змеев, Д. В. Тамазлыкарь // Информационные технологии и математическое моделирование (ИТММ-2019): Материалы XVIII Международной конференции имени А. Ф. Терпугова (26−30 июня 2019 г.). Томск : Изд-во НТЛ, 2019. Ч. 1. С. 27–30.

112. Zmeev D. O., Sokolov D. A., Zmeev O. A. Simple Model – a Way to Integrate Software Development Processes and Project Management Software // Математическое и программное

обеспечение информационных, технических и экономических систем: материалы VI Международной молодежной научной конференции. Томск, 24–26 мая 2018 г. / под общ. ред. И. С. Шмырина. Томск : Издательский Дом Томского государственного университета, 2018. С. 357–361.

113. Змеев Д. О., Соколов Д. А. Интеграция процессов разработки и сред управления проектами // Информационные технологии: МНСК-2018: Материалы 56-й Междунар. науч. студ. конф., 22–27 апреля 2018 г. / Новосиб. гос. ун-т. Новосибирск : ИПЦ НГУ, 2018. С. 166.

114. Змеев О. А., Змеев Д. О., Соколов Д. А. Реализация проектного метода обучения на основе обобщенной модели процесса разработки // Информатика и образование. 2017. № 6 (285). С. 51–57.

115. Разработка механизма применяющего процесс разработки в среде управления проектами / Д. О. Змеев, А. С. Луговая, Д. А. Соколов, О. А. Змеев // Наука. Технологии. Инновации. Сборник научных трудов в 9 ч. / под ред. асс. Макарова С. В. Новосибирск : Изд-во НГТУ, 2016. Часть 1. С. 88–90.

СПИСОК ИЛЛЮСТРАЦИЙ И ТАБЛИЦ