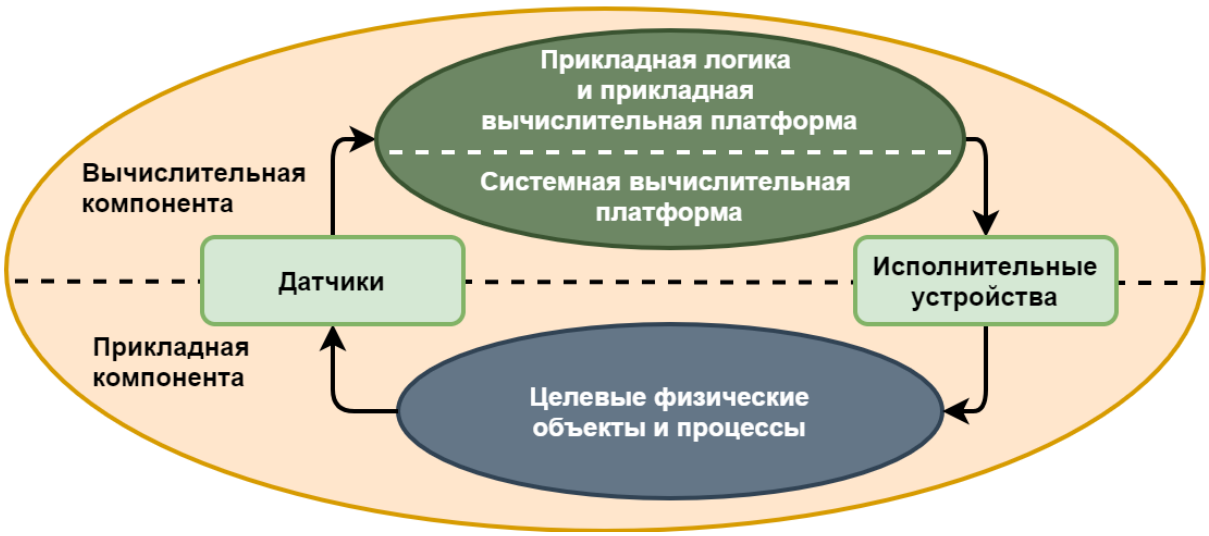
УНИВЕРСИТЕТ ИТМО

Я.Г. Горбачев, А.Е. Платунов, В.Ю. Пинкевич, М.В. Кольчурин

КИБЕРФИЗИЧЕСКИЕ СИСТЕМЫ. МЕТОДЫ ВЫСОКОУРОВНЕВОГО ПРОЕКТИРОВАНИЯ



Санкт-Петербург
2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

Я.Г. Горбачев, А.Е. Платунов, В.Ю. Пинкевич, М.В. Кольчурин

КИБЕРФИЗИЧЕСКИЕ СИСТЕМЫ. МЕТОДЫ ВЫСОКОУРОВНЕВОГО ПРОЕКТИРОВАНИЯ

УЧЕБНОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлениям подготовки 09.04.01 «Информатика и вычислительная техника», 09.04.04 «Программная инженерия» в качестве учебного пособия для реализации основных профессиональных образовательных программ высшего образования магистратуры

Санкт-Петербург
2022

Я.Г. Горбачев, А.Е. Платунов, В.Ю. Пинкевич, М.В. Кольчурин. Киберфизические системы. Методы высокоуровневого проектирования. – СПб: Университет ИТМО, 2022. – 48 с.

Рецензент:
Ключев А.О., кандидат технических наук, доцент (квалификационная категория «ординарный доцент») факультета программной инженерии и компьютерной техники, Университет ИТМО.

В учебном пособии представлены современные методы высокоуровневого проектирования, применяемые при создании киберфизических систем.

Стандарт Essence предлагает методологическое ядро и язык описания методов программной инженерии. Изменение ядра Essence для системной инженерии позволяет успешно применять его для анализа проектов киберфизических систем.

Метод ARCADIA предназначен для модельно-ориентированного архитектурного проектирования систем разного назначения. Инструментальная поддержка данного метода реализована в среде Capella.

Учебное пособие включает лабораторный практикум по данным методам высокоуровневого проектирования. Студенты работают с примерами реальных киберфизических систем, используют современные инструментальные средства и лабораторные стенды.

Университет ИТМО – национальный исследовательский университет, ведущий вуз России в области информационных, фотонных и биохимических технологий. Альма-матер победителей международных соревнований по программированию – ICPC (единственный в мире семикратный чемпион), Google Code Jam, Facebook Hacker Cup, Яндекс.Алгоритм, Russian Code Cup, Topcoder Open и др. Приоритетные направления: IT, фотоника, робототехника, квантовые коммуникации, трансляционная медицина, Life Sciences, Art&Science, Science Communication. Входит в ТОП-100 по направлению «Автоматизация и управление» Шанхайского предметного рейтинга (ARWU) и занимает 74 место в мире в британском предметном рейтинге QS по компьютерным наукам (Computer Science and Information Systems). С 2013 по 2020 гг. – лидер Проекта 5–100.

© Университет ИТМО, 2022
© Я.Г. Горбачев, А.Е. Платунов, В.Ю. Пинкевич, М.В. Кольчурин, 2022

Оглавление

Введение

На наших глазах происходит глубокая интеграция и трансформация различных отраслей хозяйства, областей деятельности, сегментов науки и техники на платформе инфокоммуникационных технологий (ИКТ или просто ИТ). Это радикально меняет существование человека, среду его обитания, способы общения. Но также меняется и сама вычислительная техника.

Наибольшую долю рынка ИКТ традиционно занимают задачи прикладного характера, направленные на внедрение вычислительной техники и получение от нее реальной отдачи. Вместе с тем, естественно возникшее и стремительно развивающееся разделение труда в инфокоммуникационной (ИК) индустрии и то, что область ИКТ еще не оформилась как традиционная наука, несут в себе значительные технологические риски и угрозы. Отсутствует даже единый термин, отражающий все аспекты деятельности в ИК-области.

Тенденция к созданию глубоко интегрированных прикладных систем на ИК-платформе предполагает системный подход к проектированию с равными приоритетами разработчиков всех ключевых профилей. Сегодня такие комплексные системы ассоциируются с киберфизическими системами (КФС, Cyber-Physical systems, CPS), социокиберфизическими системами (СКФС, SCPS), интернетом вещей (ИВ, Internet of Things, IoT).

Настоящее учебное пособие представляет проблематику высокоуровневого проектирования КФС и дает описание методов, применяемых в системном проектировании.

Для практического освоения материала предлагается три цикла лабораторных работ. Они направлены на развитие навыков системного проектирования КФС с помощью современных методов и языков архитектурного моделирования, а также развитие навыков исследования проектного пространства КФС на примере реализации компонента КФС на микропроцессорном устройстве.

Учебное пособие предназначено для использования в курсах «Проектирование киберфизических систем», «Моделирование киберфизических систем», «Организация вычислительного процесса», «Системная инженерия».

1. Проблемы проектирования киберфизических систем

1.1. Киберфизические системы, интернет вещей, встраиваемые системы

Согласно одному из официальных определений, киберфизические системы – созданные в результате инженерной деятельности системы, которые основаны на бесшовной интеграции вычислительных алгоритмов и физических компонентов и зависят от этой интеграции (National Science Foundation, США; здесь и далее приводится авторский перевод терминов и определений). КФС могут быть небольшими и автономными, например, искусственная поджелудочная железа, или очень большими, сложными и распределенными, например, региональная энергосистема.

Как отмечают многие специалисты (например, [1, 2]), появление таких категорий систем, как КФС и Интернет вещей, явилось естественным развитием встраиваемых вычислительных систем (ВсС, Embedded Systems, ES), сетевых или распределенных ВсС (PBcC, Networking Embedded Systems, NES) и беспроводных сенсорных сетей (Wireless Sensor Network, WSN).

Все входящие в состав КФС вычислительные и коммуникационные средства, включая ВсС, коммуникационную инфраструктуру, базы данных, облачные сервисы, мобильные приложения будем называть вычислительной компонентой КФС, а «физическую» составляющую – прикладной компонентой КФС (рис. 1) [3].

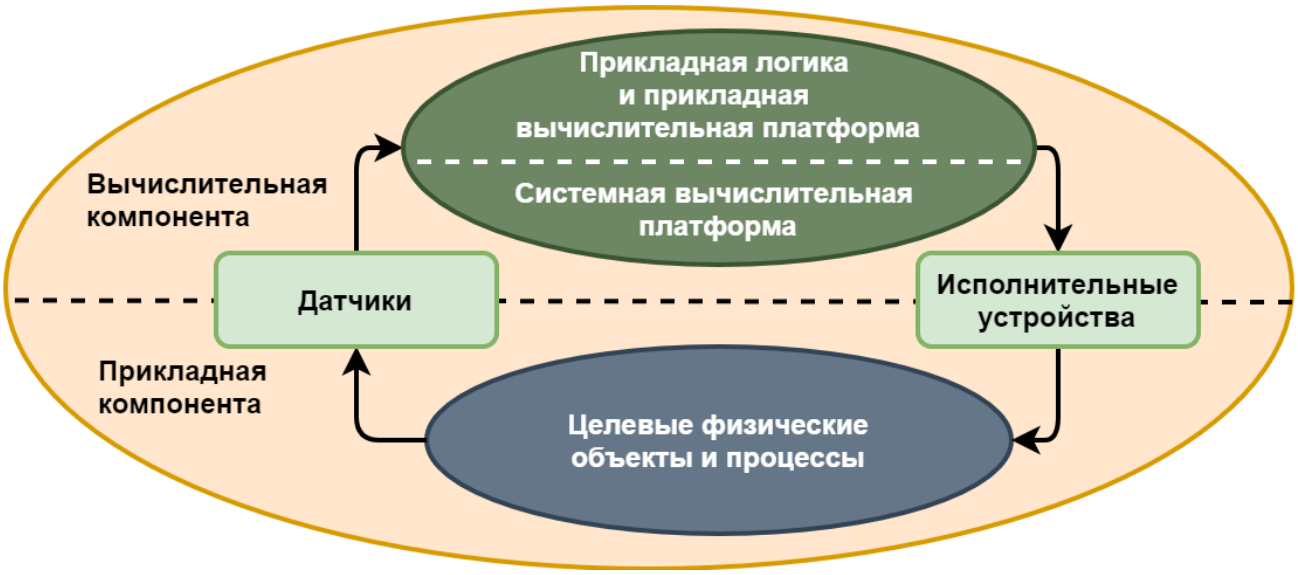


Рисунок 1 – Обобщенная структура КФС

Можно выделить две основные трактовки того, что следует относить к КФС.

«Узкая» трактовка: новый отдельный класс систем, чья ИК-платформа характеризуется тесной связью с объектом мониторинга и/или управления, распределенностью, интеллектуальными алгоритмами (Artifical Intellignce – AI, Big Data, Virtual/Augmented Reality – VR/AR, …), широким использованием вычислительных сетей различного масштаба. Считается, что такие системы требуют «особых» методов проектирования.

«Широкая» трактовка: новое поколение технических систем, которое характеризуется качественным изменением всей технологии проектирования и внедрением новых технических и технологических решений в части ИК-платформы (в т.ч. AI, IoT, Big Data, VR/AR…) для получения качественно новой функциональности. Включает узкую трактовку и не отрицает возможности проектировать системы любого назначения и масштаба по новым принципам.

Узкая трактовка относит к КФС системы автоматизации производства, отвечающие критериям инициативы «Индустрия 4.0», но исключает «простые» системы автоматики, хотя они могут отличаться только набором функций, а не принципом проектирования. Далее будем придерживаться широкой трактовки, считая ее более перспективной.

ВсС остаются наиболее характерными представителями вычислительной компоненты КФС. Их можно определять как вычислительные системы (ВС), непосредственно взаимодействующие с физическим объектом мониторинга и/или управления, объединенные с таким объектом логикой работы и, часто, единой конструкцией [4]. Это иногда приводит к ложному восприятию: либо ВсС (без физического объекта) отождествляется с КФС, либо системы, в которых вычислительная компонента представлена в виде традиционной ВсС (без «модных» технологий, но с использованием методов бесшовной интеграции), считают «неполноценными» КФС, забывая, что мода проходит.

ИВ не является обязательным компонентом КФС и обычно используется в качестве унифицированной коммуникационной среды в открытых распределенных системах автоматики. По определению стандарта ISO/IEC 20924:2018, ИВ – инфраструктура взаимосвязанных и взаимодействующих объектов, людей, систем и информационных ресурсов вместе со службами, которые обрабатывают информацию и реагируют на информацию из физического и виртуального мира.

Появление КФС, на наш взгляд, связано с двумя основными факторами:

– отрасль ИКТ достигла состояния, которое позволяет широко внедрять ИК-компоненты в системы и процессы почти любого масштаба и назначения;

– созданы доступные массовому разработчику методы и технологии, позволяющие эффективно и относительно дешево проектировать комплексные системы с большой долей ИК-компонентов.

Разнообразие областей применения КФС делает востребованным качественно новый по сложности уровень автоматизации, позволяющий проектировать и массово производить системы, которые иначе было бы невозможно создать в принципе (например, беспилотные автомобили). В таких, на первый взгляд, далеких от компьютеров системах вычислительная компонента начинает играть ведущую роль, а обязанности ИТ-специалиста расширяются до полноценного системного инженера.

Сегодня реализация сбалансированного киберфизического подхода остается скорее искусством, чем отработанной технологией. Существующие методики и маршруты проектирования тяготеют либо к «физико-центричному», либо к «компьютеро-центричному» подходу. В первом случае основные усилия

направлены на разработку прикладной компоненты КФС с ограниченным учетом возможностей и рисков при создании вычислительной компоненты, что может привести к критическим проблемам с надежностью и эффективностью системы в целом. Во втором случае вычислительная компонента создается до начала глубокого погружения в прикладную функциональность. К данным крайностям обычно тяготеют специалисты без достаточного кругозора, вынужденно играющие роль системного инженера в проектах КФС.

На рис. 2 показана эволюция управляющих ВС, их элементной базы, рост степени их интеграции с объектом мониторинга и/или управления. В настоящее время методы проектирования элементной базы становятся все более близкими к методам проектирования управляющих ВС, особенно в части вычислительной архитектуры. Сегодня система на кристалле может «закрывать» значительную часть вычислительных задач КФС.

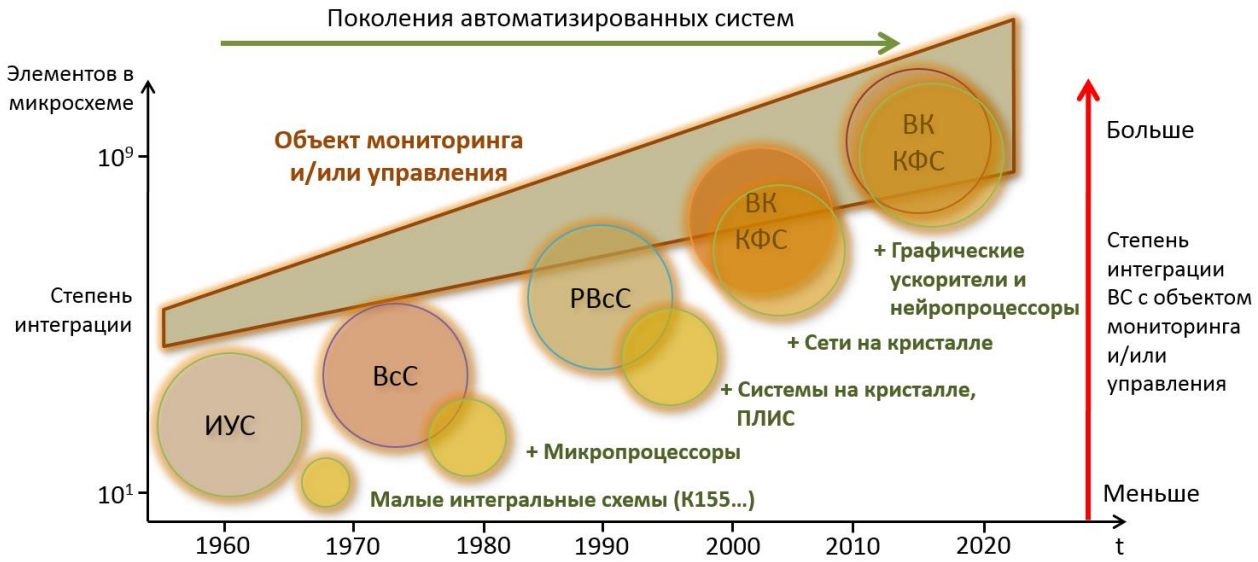


Рисунок 2 – Эволюция вычислительной компоненты (ВК) КФС (ИУС – информационно-управляющие системы)

1.2. Проблемы проектирования вычислительной компоненты КФС

В ИК-отрасли существуют общие проблемы, которые профильные ИТ-специалисты зачастую даже не ощущают, но последствия от их нерешенности радикально ухудшают условия и результаты проектирования, что особенно сильно заметно в сфере создания КФС.

1.2.1. Отставание методов системного проектирования в ИК-отрасли

Необходимость массового проектирования КФС приводит к резкому росту междисциплинарности и комплексности проектов в пересчете на один коллектив. Вместе с новыми прикладными вычислительными технологиями это обостряет проблемы роста ИК-индустрии. Хроническое отставание методов и инструментальных средств разработки ВС не позволяет в полной мере использовать технологический потенциал ИК-отрасли в новых проектах.

Из-за объективных трудностей проектирования со сквозной оптимизацией технических решений по множеству вариантов во всех стеках технологий задачу сводят либо к крупноблочному проектированию из готовых компонентов и по готовым шаблонам, либо к заказному проектированию с использованием ряда навязанных или выбранных без должного обоснования макро- и микроархитектурных решений. Так, в рамках используемых в настоящее время индустриальных методов процесс проектирования КФС жестко разделен на ряд фаз, которые поддерживаются инструментальными средствами, системами автоматизированного проектирования (САПР). Соответственно существует узкая специализация разработчиков, которая формируется в вузе и совершенствуется в процессе практической деятельности. Можно выделить три сегмента деятельности, непосредственно относящихся к созданию вычислительной компоненты КФС:

– проектирование систем на кристалле (включая технологии ASIC, ASIP, FPGA);

– проектирование специализированных контроллеров и ВсС на их основе;

– проектирование систем автоматики на базе готовых программируемых логических контроллеров, SCADA-систем, технологий информационных систем.

Внутри сегментов процессы проектирования выстроены в виде эффективных и почти «бесшовных» маршрутов, степень автоматизации которых постоянно растет «снизу вверх» от рутинных операций уровня инженерной реализации к уровням проектирования, которые сегодня слабо формализованы. В каждом сегменте существуют канонические решения, позволяющие с приемлемым качеством «закрывать» значительную часть сегодняшних задач. Однако, при кажущемся благополучии ситуации, использование ограниченного числа типовых проектных ИК-платформ для множества разнообразных задач снижает качество создаваемых продуктов. Использование существующих методов и средств проектирования не позволяет выполнять адекватный анализ вариантов, влиять на внутреннее, глубинное устройство платформы, чтобы эффективно решать многие вопросы, среди которых:

– нехватка вычислительной мощности;

– большое энергопотребление;

– сложность программирования задач реального времени (РВ);

– сложность реализации и программирования гетерогенных многопроцессорных структур;

– низкая информационная безопасность.

Необходимо констатировать, что на сегодня отсутствуют эффективные языковые средства и способы документирования, предоставляющие возможность даже просто описать структуру, принципы и механизмы функционирования современных ВС с многоуровневой и гетерогенной организацией, которыми (во многом умозрительно) оперирует ИТ-специалист, детализируя архитектуру КФС. Следствием этого является высокая непрозрачность программных и аппаратных средств и технологий, создаваемых

в ИК-отрасли. Описания технических решений оказываются «раздроблены» на части по технологическому принципу, во многих случаях сводятся только к конечным техническим документам (принципиальные электрические схемы, тексты программ, технические описания сложно-функциональных микросхем).

Анализ таких документов для получения достоверной информации о работе системы или ее части («реверс-инжиниринг») является крайне трудоемким сам по себе и не способствует получению цельной картины для специалиста-проектировщика (архитектора), вынужденного работать с полным стеком технологий. Таким образом, результаты заказного проектирования системы или ее компонентов, полученные с использованием существующих низкоуровневых технологий проектирования и программирования, плохо масштабируются и малопригодны для повторного использования. Ситуацию усугубляет часто проявляющееся «плоское» мышление специалиста – нежелание или невозможность оперировать уровневыми представлениями вычислительной системы и ее частей.

Тем не менее, практика показывает, что при наличии команды высококвалифицированных разработчиков можно получать эффективные решения с высокой степенью повторного использования при умеренных затратах, что подтверждает отсутствие адекватных методик проектирования, доступных массовому разработчику. Поэтому крайне необходимыми и по-прежнему требующими научной активности остаются технологии и инструменты, принадлежащие ко всем основным задачам создания как специализированных ВС, так и вычислительных платформ широкого назначения. Приведем примеры таких областей:

– формальные методики проектирования;
– системная архитектура;
– модельно-ориентированная инженерия;
– многоядерные и многопроцессорные платформы;
– модели вычислений;
– смешанные системы с критическими функциями (mixed-critical systems);
– системы жесткого РВ.

1.2.2. Принцип Копеца и адаптивность КФС

Пример кардинально различающихся точек зрения, определяющих наполнение и ход проекта КФС, сформулирован в «принципе Копеца» [5]:

– многие (прогнозируемые) свойства, которые мы приписываем системам (детерминизм, своевременность реакции, надежность, безопасность), на самом деле являются не свойствами реализованной системы, а скорее свойствами модели системы;

– опираясь на выбранные модели, мы можем делать выводы о свойствах реализации системы; достоверность этих выводов зависит от качества модели, которая всегда является приблизительной.

На практике система почти никогда полностью не совпадает с ее восприятием инженером (моделью системы) хотя бы потому, что нет полной

информации об использованных в проекте сторонних компонентах. Важно безоговорочно не приписывать реализации системы свойства модели, а напротив, рассчитывать на возможность проявления «неожиданных» свойств, в том числе при поиске причин неадекватного функционирования системы.

При создании КФС, имея дело со сложной, интеллектуальной автоматикой, необходимо закладывать в проект значительный «запас гибкости», дополнительные «страхующие» механизмы и решения, реализуя принцип адаптивности не только на уровне прикладных алгоритмов, а по возможности на всех уровнях организации системы. Сегодня создание таких адаптивных систем – открытая научная и техническая проблема, решение которой в области вычислительной компоненты КФС требует пересмотра парадигмы проектирования.

1.2.3. Приоритеты в проектировании КФС

По замыслу авторов термина КФС усилия научно-технического профильного сообщества должны в приоритетном порядке быть направлены на решение проблем вычислительной платформы КФС, на развитие системного и параллельного (совместного) стиля проектирования, а также на вопросы подготовки кадров [2, 6].

Однако у значительной части специалистов существует иллюзия отсутствия фундаментальных проблем в области проектирования ВС, что приводит к увеличению массы незрелого и непрозрачного продукта и смещению приоритетов в исследованиях в ИК-сфере. Освоение и развитие системных, кросс-уровневых, сквозных технологий и инструментов проектирования в области вычислительной техники отодвигается на второй план. Вместо этого значительные силы в последнее время оказались направлены на развитие новых, безусловно актуальных, областей (IoT, Big Data, AI, Digital Industry, VR/AR и др.), значительно сократив финансирование и снизив престижность работ, решающих важнейшие открытые проблемы фундамента вычислительной техники.

Для повышения качества проектирования КФС должен быть восстановлен баланс в направлениях исследований и в подготовке ИТ-специалистов.

1.3. Подготовка специалистов для проектирования КФС

1.3.1. Структура компьютинга

Для обозначения всей области ИКТ является удачным используемый в документах ACM и IEEE [7] термин «компьютинг» – целенаправленная деятельность, требующая компьютеров, использующая компьютеры, создающая компьютеры [8, 9]. С течением времени область компьютинга и сообщество ИТ-специалистов распадаются на отдельные сегменты, которые все больше изолируются друг от друга. Приведем определения основных направлений компьютинга (в скобках указаны документы-источники из [7]).

Компьютерная инженерия (CE – computer engineering) охватывает науку и технологию проектирования, конструирования, внедрения и обслуживания программных и аппаратных компонентов современных вычислительных систем, оборудования с компьютерным управлением и сетей интеллектуальных устройств (CE2016).

Информатика (CS – computer science) охватывает широкий диапазон от теоретических и алгоритмических основ компьютинга до передовых разработок в области робототехники, компьютерного зрения, интеллектуальных систем, биоинформатики и в других перспективных областях (CC2005).

Программная инженерия (ПИ, SE – software engineering) – это применение систематического, дисциплинированного, поддающегося количественной оценке подхода к разработке, эксплуатации и обслуживанию программного обеспечения (ПО); то есть приложение инженерии к ПО (SE2014).

Информационные технологии (IT – information technologies) – это изучение и использование системных методик выбора, разработки, применения, интеграции и администрирования безопасных вычислительных технологий, позволяющих пользователям достигать своих личных, организационных и социальных целей (IT2017).

Информационные системы (IS – information systems) – область, специалисты которой сосредоточены на интеграции решений информационных технологий и бизнес-процессов для удовлетворения информационных потребностей предприятий, что позволяет последним эффективно и действенно достигать своих целей. Упор делается на технологии создания, обработки и распространения информации (CC2005).

Кибербезопасность (CSEC – cybersecurity) – компьютерная дисциплина, в которой задействованы технологии, люди, информация и процессы, обеспечивающие гарантии деятельности в контексте существования и действий противников (CSEC2017).

Состав направлений компьютинга продолжает расширяться. CSEC выделяется с 2017 г. С 2020 г. кандидатом на отдельное направление является наука о данных (DS – data science). На рис. 3 показаны «зоны ответственности» ИТ-специалистов, принадлежащих основным направлениям компьютинга [8], а на рис. 4 – изменение соотношения этих направлений во времени.

1.3.1. Профиль подготовки КФС-специалиста

На сегодня представлен целый ряд предложений по наполнению учебной программы подготовки КФС-специалиста. Они различаются объемом (от вводных курсов до полноценных магистерских программ), наполнением (зависит от трактовки термина КФС), степенью связи с компьютингом. Подробный обзор учебных программ представлен в [10].

Широту взглядов на наполнение области КФС и ее взаимосвязь со смежными дисциплинами, в том числе на взаимоподчиненность и связь с сегментами компьютинга, демонстрируют документы комитетов ТК 194 «Кибер-

физические системы» Росстандарта РФ и ISO/IEC JTC 1/SC41 “Internet of Things and digital twin”.

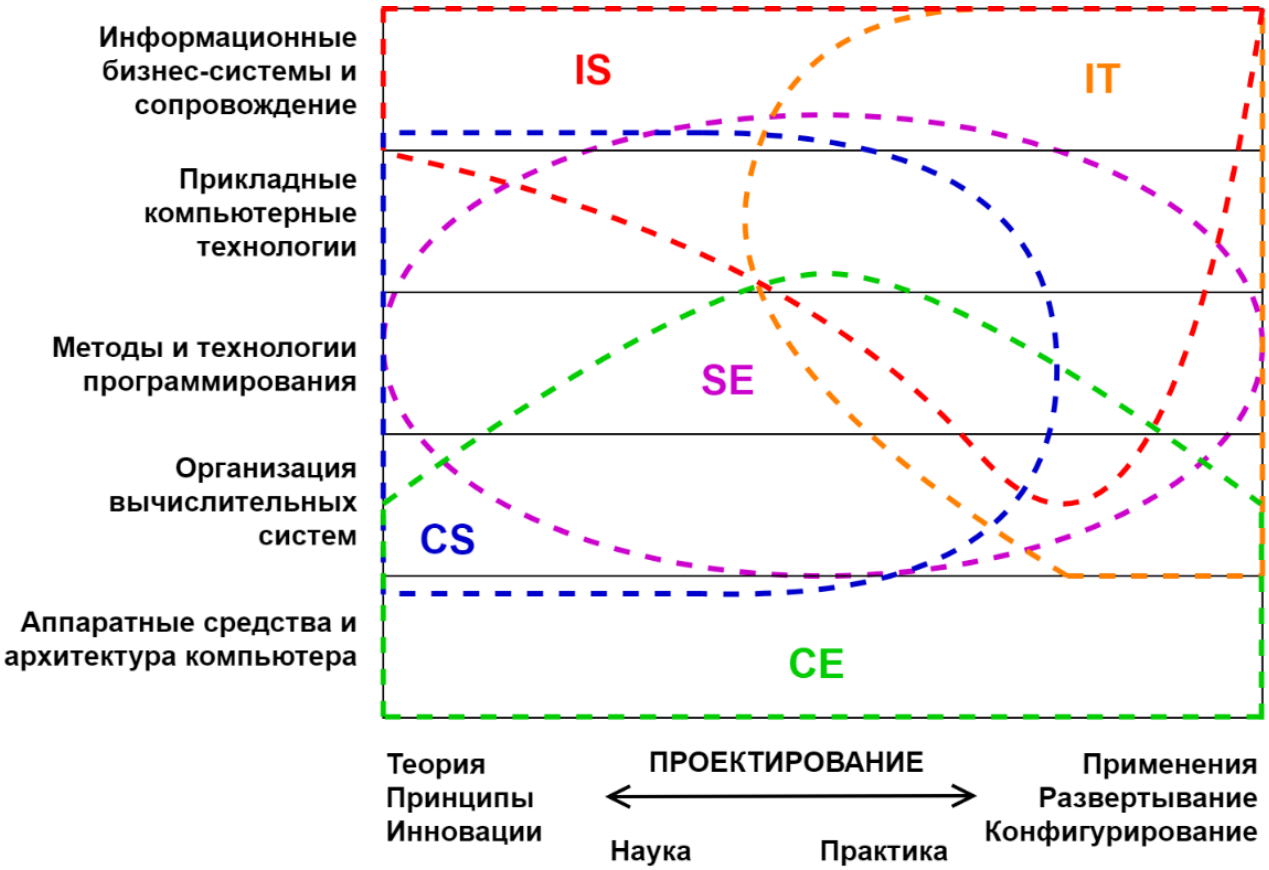


Рисунок 3 – Зоны ответственности специалистов основных направлений компьютинга [8]

В [11] перечисляются следующие разделы подготовки КФС-специалиста:
– математика, статистика;
– электроника, компьютерная архитектура;
– компиляторы, операционные системы;
– встроенное ПО;
– оптимизация;
– модели вычислений и формальные методы;
– интернет и веб-ПО, беспроводная связь;
– информационная безопасность;
– системы управления, системная инженерия;
– человеко-машинное взаимодействие;
– работа в команде, управление проектами;
– управление развертыванием и конфигурирование;
– процессный подход, проблемы жизненного цикла;
– стандарты, экологичность и другое.

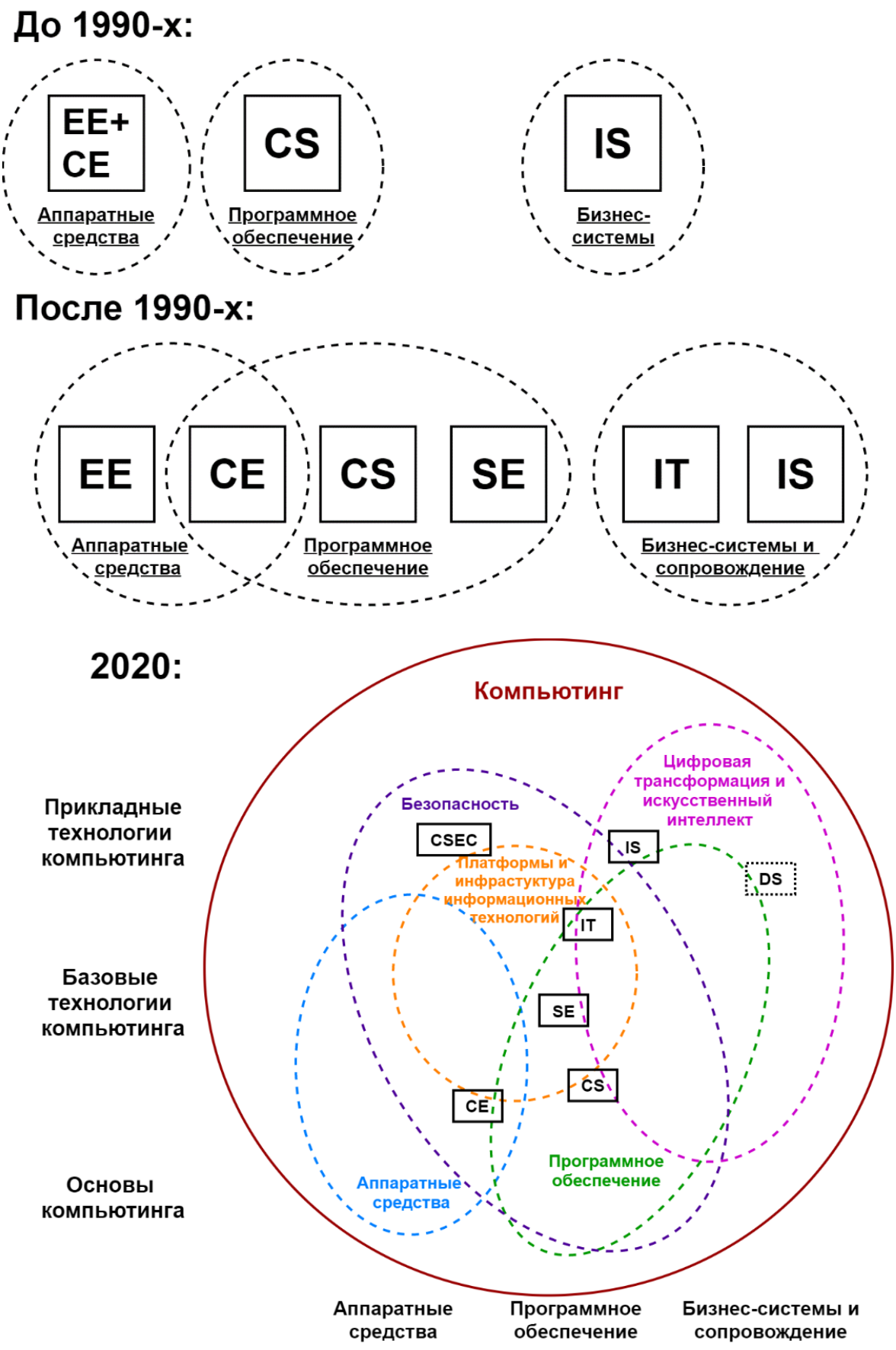


Рисунок 4 – Эволюция областей деятельности внутри компьютинга (EE – electrical engineering, электротехника и электроника) [8]

Одним из наиболее полных и интересных документов, предлагающих развернутый набор компетенций специалиста по КФС, можно считать [8]. Приведем тезисы из раздела “Foundations for a CPS curriculum”:

1. Базовые вычислительные концепции, помимо тех, что рассматриваются во вводных курсах по программированию, такие как ВсС, структуры данных, теория автоматов и ПИ.

2. Организация вычислений с учетом особенностей и ограничений объектов физического мира, что предполагает понимание:

– свойств физического мира;

– организации и особенностей ВсС РВ;

– ограничений вычислительных ресурсов, таких как энергопотребление и размер памяти.

3. Владение дискретной математикой и такими разделами непрерывной математики, как дифференциальные уравнения, вероятностные и случайные процессы, линейная алгебра.

4. Проектирование вычислительной компоненты КФС, объединяющее:

– применение датчиков, исполнительных устройств, алгоритмов управления, коммуникационных протоколов и систем, вычислений, предполагающее их междоменное и межуровневое (cross-cutting) взаимодействие;

– учет центральной роли взаимодействия между физическими и вычислительными аспектами;

– контроль над коммуникационными сетями, получением данных от датчиков, обработкой сигналов и выдачей управляющих воздействий с учетом ограничений РВ.

5. Моделирование неоднородных и динамических систем, объединяющих управление, вычисления и связь, с акцентом на неопределенность свойств объекта и гетерогенность системы, включая такие методы, как линейное и нелинейное моделирование, стохастические модели, дискретно-событийные и гибридные модели, а также связанные с ними методологии проектирования, основанные на оптимизации, теории вероятностей и динамическом программировании.

6. Методы проектирования КФС, особенно для критически важных по безопасности, надежности и отказоустойчивости применений, требующие охвата фаз жизненного цикла от технического задания до тестирования, сертификации и сопровождения, включая вопросы формальной верификации и валидации, а также возможности адаптации, позволяющие обеспечивать эволюционирование системы.

Желание определить область компетенций для подготовки системообразующего КФС-специалиста заставляет приоритетно объединять направления CE (в том числе ВсС) и IS, дополняя их разделами из CS, SE, CSEC, IT, теории автоматического управления, мехатроники. Это необходимо для проектирования с гораздо более глубоким погружением в организацию

вычислительной компоненты КФС, чем это обычно делают в сфере IS при создании бизнес-систем.

Также необходимо знакомство с прикладной физической и бизнес-аналитикой, моделями описания соответствующих процессов, технологиями программирования, тестирования, соответствующими разделами математики.

1.4. Контрольные вопросы

1. Перечислите основные структурные части КФС.
2. В чем отличия "узкой" и "широкой" трактовки понятия КФС?
3. Перечислите основные факторы появления появления КФС как нового класса систем.
4. Опишите связь понятий Интернета вещей и КФС.
5. Дайте характеристику актуальных проблем проектирования КФС.
6. Как принцип Копеца характеризует соотношение системы и ее моделей?
7. Почему важно закладывать высокую степень адаптивности в современные системы автоматики?
8. Перечислите направления компьютинга в соответствии с Computing Curricula 2020.
9. В каких областях знаний должен иметь подготовку специалист по проектированию КФС?

2. Применение стандарта Essence в системной инженерии

2.1. Описание стандарта

Essence [12] представляет собой методологическое ядро и язык для описания, создания, использования на практике и улучшения методов программной инженерии. Стандарт Essence, при минимальных изменениях, может использоваться также и в более широком смысле – в области системной инженерии, в том числе и при создании КФС.

Суть подхода Essence можно прояснить с помощью так называемой «архитектуры метода», представленной на рис. 5.

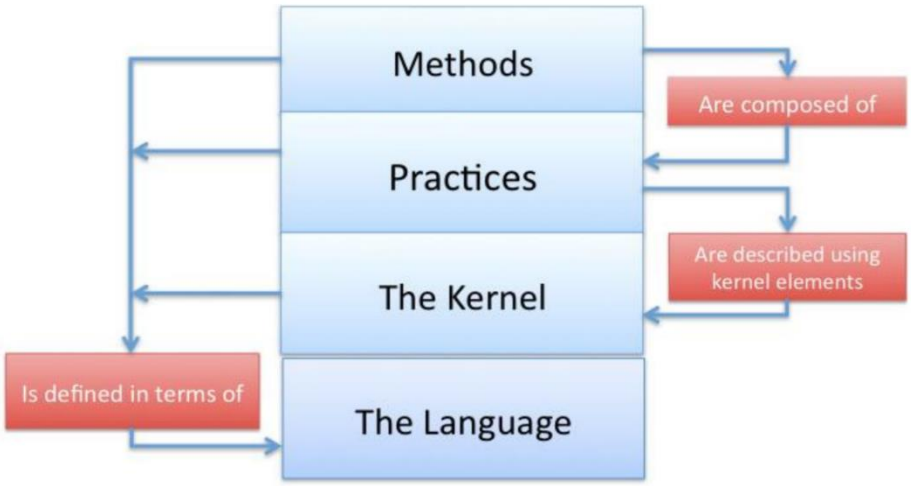


Рисунок 5 – Архитектура метода в Essence [12]

Методы состоят из практик. С точки зрения Essence, методы представляют собой не просто статические описания и наставления для разработчиков, а являются динамическими, подразумевают повседневную активность, отслеживание прогресса и состояния работ. Другими словами, они описывают не то, что планировалось сделать при старте работ, а то, что реально делается, отражая расхождения реального положения вещей от планируемого. Методами предполагается возможность замены набора используемых практик «на лету» для приведения работ в соответствие с начальным планом.

Практики определяются как повторяющиеся подходы к какой-либо деятельности, которые предоставляют систематический и верифицируемый подход к работе с каким-либо аспектом проекта, направленные на достижение определённой цели. Практики могут быть частью множества методов.

Ядро является основой для описания методов и включает в себя элементы, которые всегда присутствуют в любом маршруте проектирования, например такие, как требования к системе и команда разработчиков. Элементы имеют состояния, изменяющиеся со временем. Одной из основных функций ядра является помощь проектировщикам в сравнении методов и принятии оптимальных решений по их выбору.

Ядро описывается при помощи языка, который определяет абстрактный, графический и текстовый синтаксисы, а также динамическую семантику. Языком поддерживается составление из отдельных практик новых комбинированных практик и методов.

Ядро Essence формирует словарь базовых понятий, который представляет собой карту контекста программной инженерии, с помощью которого может быть объявлена и описана любая практика или метод. Само по себе ядро может быть использовано даже при отсутствии конкретного метода, так как все его элементы так или иначе присутствуют в любом процессе разработки программного обеспечения.

2.2. Ядро Essence

Ядро описывается в трёх областях интересов: заказчика, технического решения и менеджмента (в некоторых переводах «предпринятия», не путать с предприятием). В каждой из областей интересов находится некоторое количество базовых элементов: «альфы», области деятельности и компетенции.

«Альфы» (ALPHA, Abstract-Level Progress Health Attribute) фиксируют ключевые понятия, которые используются в программной инженерии, позволяют контролировать прогресс и состояние проекта и обеспечивают общую основу для определения методов и практик. Каждая «альфа» имеет набор заданных состояний, и в каком из них находится «альфа» в конкретный момент определяется простыми тестами. Переходы между состояниями – не простые линейные последовательности, возможны возвраты назад и множества итераций.

«Альфы» не должны рассматриваться как некое физическое разделение на конкретные работы, абстрактные рабочие продукты или подсистемы. Скорее, они представляют собой критические показатели тех аспектов, которые

являются наиболее важными для отслеживания и контроля. Важно то, что «альфы» совершенно не зависят от выбранных практик и методов.

На рис. 6 перечислены «альфы» ядра. Их суть и назначение:

1. Возможность (Opportunity) – совокупность обстоятельств, которая делает разработку вообще возможной. Причина для создания новой системы отражает общее понимание потребностей заинтересованных сторон, помогает формировать требования к будущей системе.

Состояния: идентифицирована, необходимость решения подтверждена, польза от успешного решения установлена, решение признано жизнеспособным, продемонстрированное решение удовлетворяет потребность, извлекается выгода.

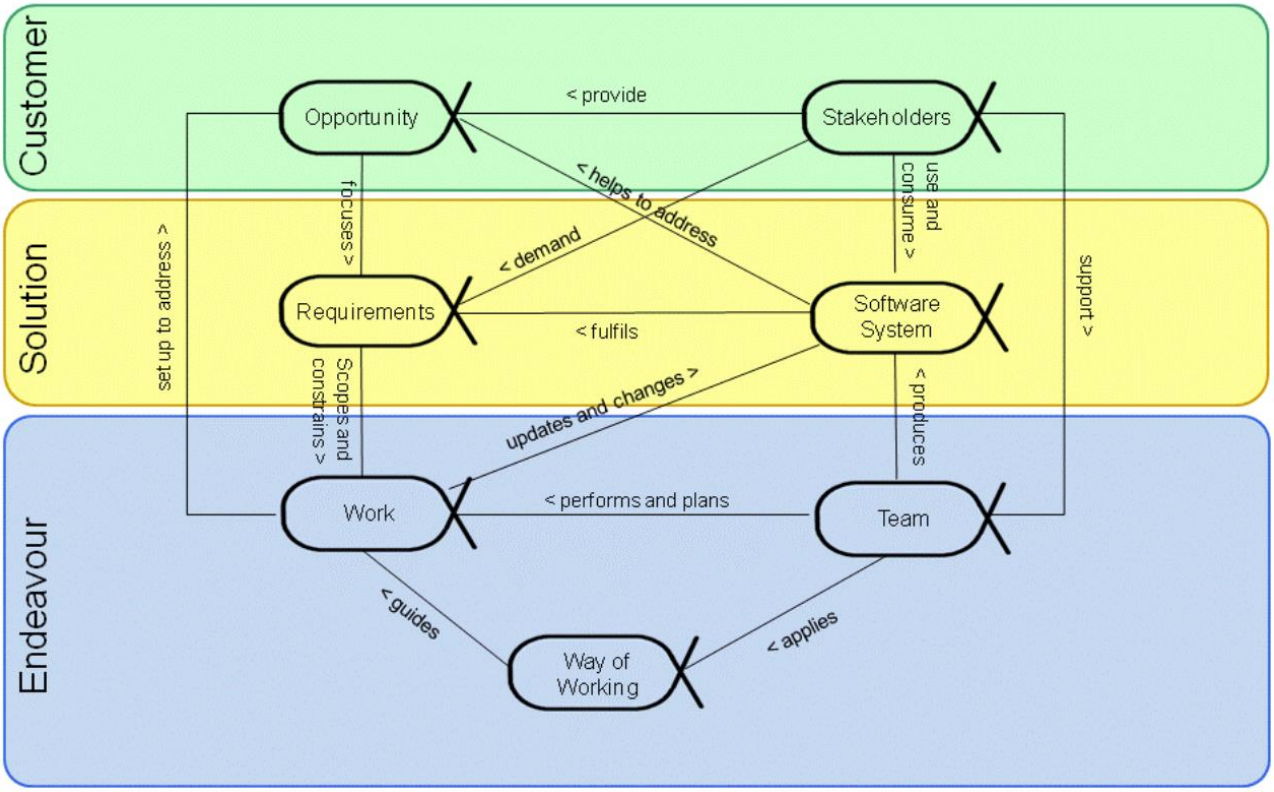


Рисунок 6 – «Альфы» ядра Essence

2. Стейкхолдеры, заинтересованные стороны (Stakeholders) – все люди, коллективы и организации, вовлечённые в процесс, включая заказчиков и разработчиков.

Состояния: определены, представлены, вовлечены в работу, достигли взаимопонимания, минимальные ожидания стейкхолдеров удовлетворены, система удовлетворяет требованиям стейкхолдеров или превышает их.

3. Требования (Requirements) – то, что программное обеспечение должно выполнять, чтобы соответствовать заявленной цели и удовлетворять стейкхолдеров.

Состояния: потребность в новой системе согласована, назначение новой системы определено, требования описывают систему непротиворечиво, требования описывают приемлемую для стейкхолдеров систему, введено

достаточно требований для удовлетворения стейкхолдеров, система полностью удовлетворяет требованиям.

4. Программная система (Software System) – система, состоящая из аппаратного обеспечения, программного обеспечения и данных, которая реализует некую предначертанную целевую функцию посредством выполнения программного обеспечения.

Состояния: архитектура выбрана, архитектура демонстрируема, система готова к использованию и демонстрирует все заявленные качества, система готова к внедрению, система используется, система больше не поддерживается.

5. Работа (Work) – умственные или физические усилия, предпринимаемые для достижения результата. В контексте разработки программного обеспечения работа включает всё, что делается командой для создания программного обеспечения, соответствующего требованиям.

Состояния: намечена, подготовлена, начата, под контролем, закончена, закрыта.

6. Команда (Team) – группа людей, активно вовлечённых в работу с программным обеспечением на протяжении всего жизненного цикла, включая проектирование, разработку, тестирование, техническую поддержку и сопровождение.

Состояния: намечена, сформирована, вовлечена в производственный процесс, результативно работает, освобождена.

7. Технология работы (Way of Working) – набор практик и инструментов, используемых командой разработчиков в качестве руководств и для автоматизации своих работ.

Состояния: определяющие технологию принципы и ограничения установлены, основа заложена, используется некоторыми членами команды, используется всеми членами команды, используется эффективно, выведена из употребления.

«Альфы» могут содержать «суб-альфы». Графически это представляется как связь «альфы» и «суб-альф» линиями с указанием количества «суб-альф», причём из «альфы» линия выходит вертикально вниз, начинается с закрашенного ромба, и только потом разветвляется. К «альфам» могут подсоединяться «рабочие продукты», графически это изображается аналогичным образом, только линия от «альфы» отходит горизонтально (рис. 7).

Следующие элементы ядра – области деятельности – дополняют «альфы» и представляют взгляд на программную инженерию, основанный на действиях. Все области деятельности, приведённые на рис. 8, интуитивно понятны и не нуждаются в дополнительных пояснениях.

Третьим элементом, составляющим ядро, являются компетенции. Они также описываются для трёх областей интересов и раскрывают основные способности и навыки, требуемые, чтобы успешно выполнять работу. Типы компетенций так же, как и в предыдущем случае, приведены на рис. 9 и интуитивно понятны.

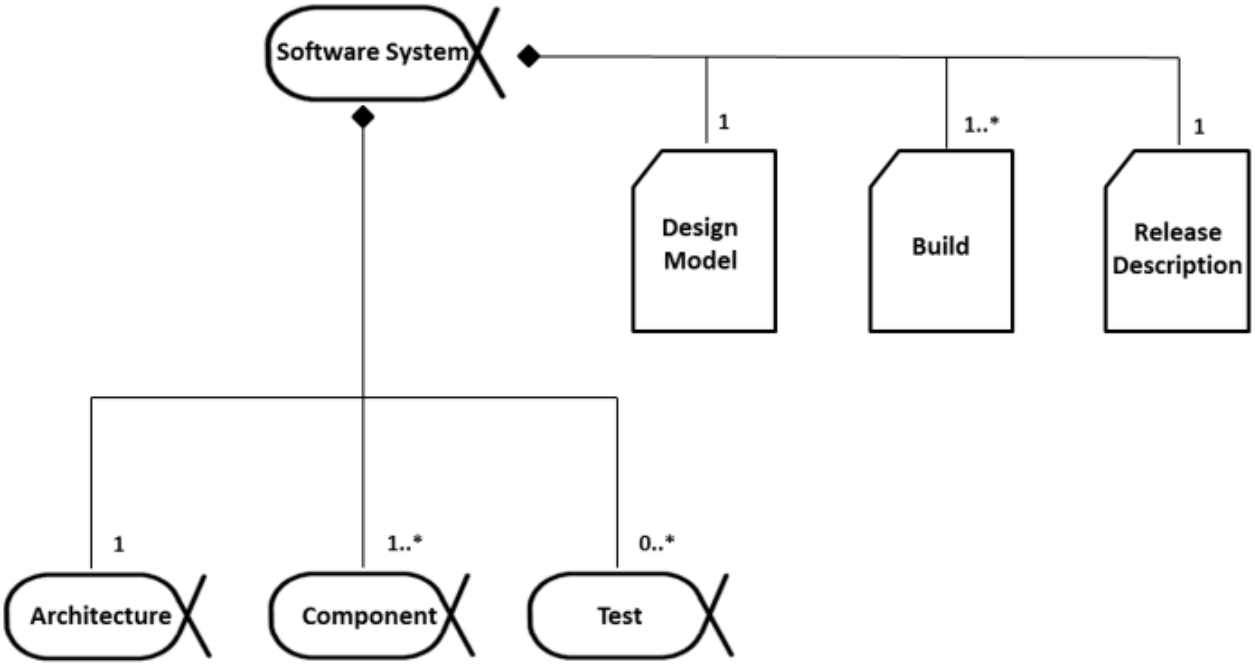


Рисунок 7 – «Суб-альфы» и рабочие продукты программной системы

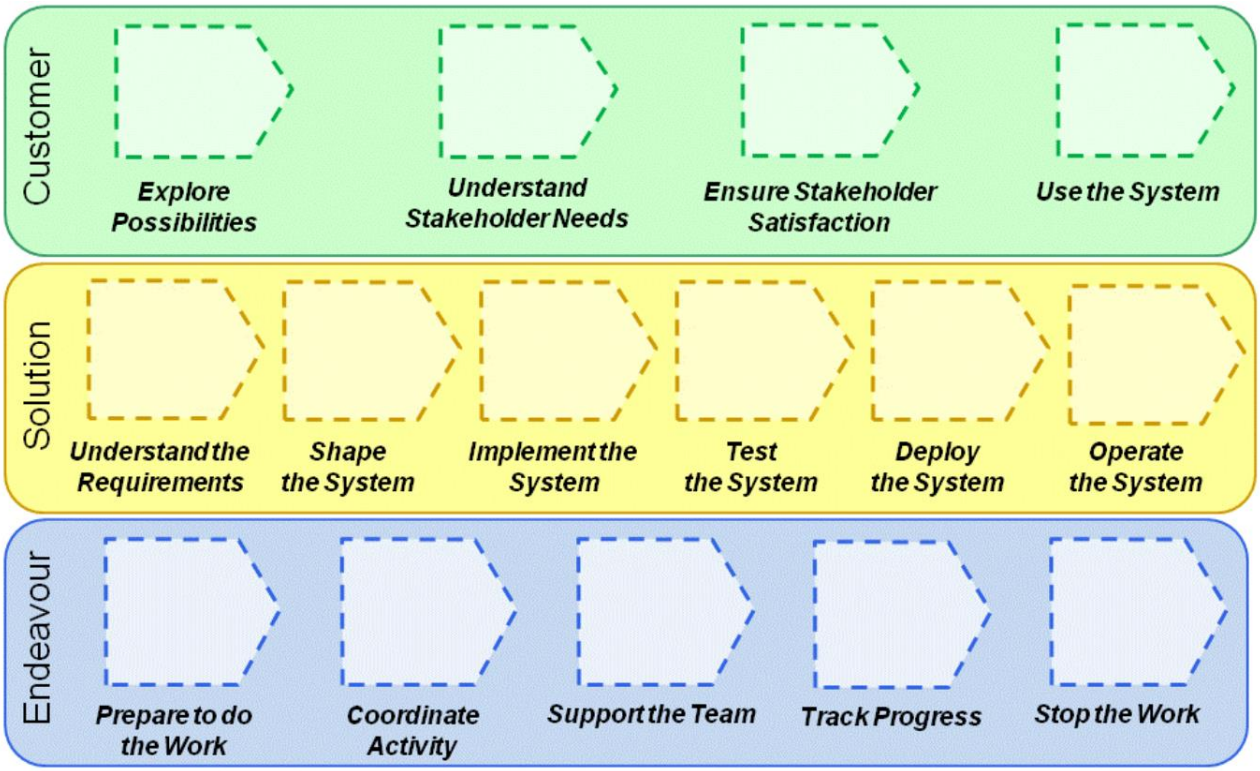


Рисунок 8 – Области деятельности ядра Essence

Выделяют пять условных уровней владения компетенциями:

1. Базовое понимание концепций и способность выполнять инструкции.
2. Способность применять концепции в простейших задачах, шаблонно используя предыдущий опыт.
3. Способность применять концепции в большинстве задач и опыт самостоятельной работы.

4. Способность принимать решения, когда и как применять концепции в более комплексных задачах, способность делиться опытом с другими.

5. Признанный эксперт, способный распространять концепции на различные задачи в различных контекстах и вдохновлять на их использование других.

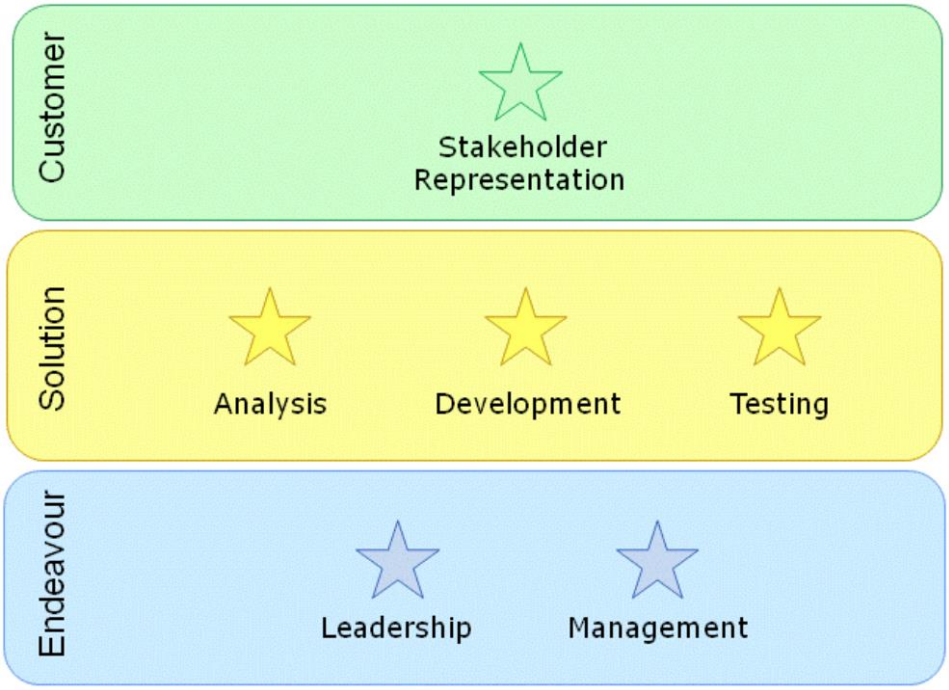


Рисунок 9 – Компетенции

2.3.Язык Essence

Основные элементы языка и их важнейшие связи показаны на рис. 10. Элементы по центру («альфа»; состояние «альфы»; область деятельности, она же деятельность; компетенция), используются для описания ядра. Это абстрактные и необходимые сущности, то, что приходится делать, то, с чем приходится работать, или то, что необходимо знать. Считается достаточным знать эти четыре элемента, чтобы иметь возможность говорить о состоянии, степени готовности и успешности работ по воплощению программного проекта.

В то время как элементы, используемые в ядре, отображают абстрактные вещи, конкретика может вводиться с помощью конкретных практик путем добавления элементов, подобных тем, которые показаны на правой стороне рисунка. Рабочие продукты представляют собой конкретные сущности, результаты работ, влияющие на состояния «альф». Например, исходный код предоставляет сведения о том, реализован ли компонент полноценно или только в виде временной «заглушки». Работы содержат четкие указания о том, как производить или обновлять рабочие продукты, изменения в которых в итоге должны приводить к изменениям состояния «альф».

Рабочие продукты могут иметь различные уровни детализации, отображаемые отдельно как состояния «альф». Они показываются в виде сужающихся книзу трапеций, соединённых стрелками, идущими от объектов меньшего уровня детализации к большему (рис. 11).

Динамическая семантика языка связана с состояниями «альф» и работами. Основываясь на том, в каких состояниях находится проект, и на том, каких состояний команда желает достичь, принимается решение инициировать те работы, которые приведут к заданной цели.

К областям деятельности могут быть привязаны различные работы. Графически это отображается либо нарисованными внутри области деятельности символами работ, либо символами, вынесенными вбок на соединительной линии аналогично привязке к «альфам» рабочих продуктов (рис. 12).

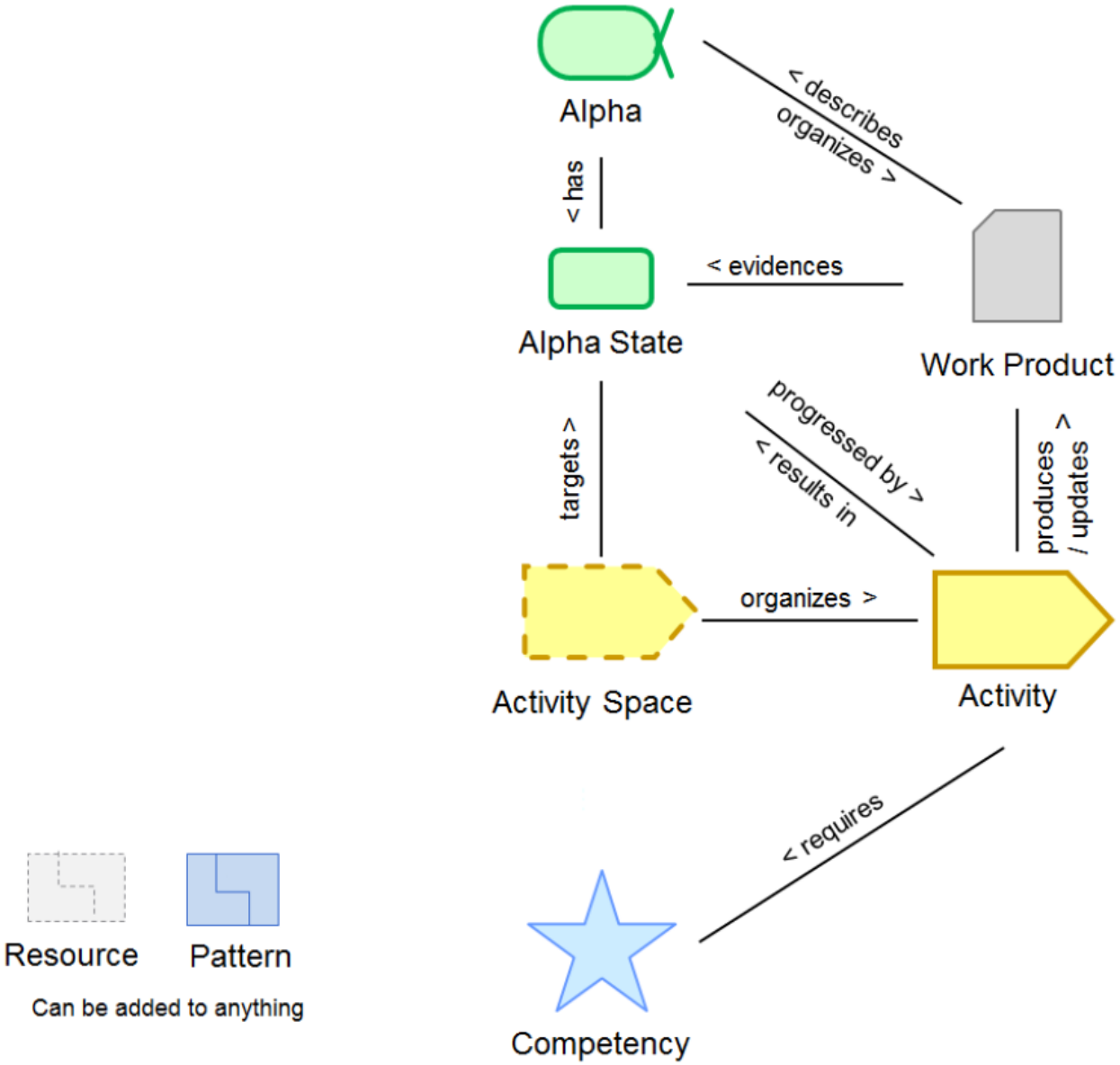


Рисунок 10 – Концептуальный обзор языка

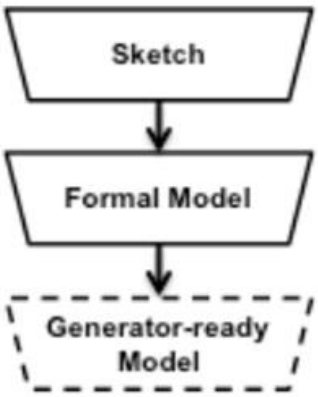


Рисунок 11 – Различные уровни детализации на примере рабочего продукта «архитектурное описание»

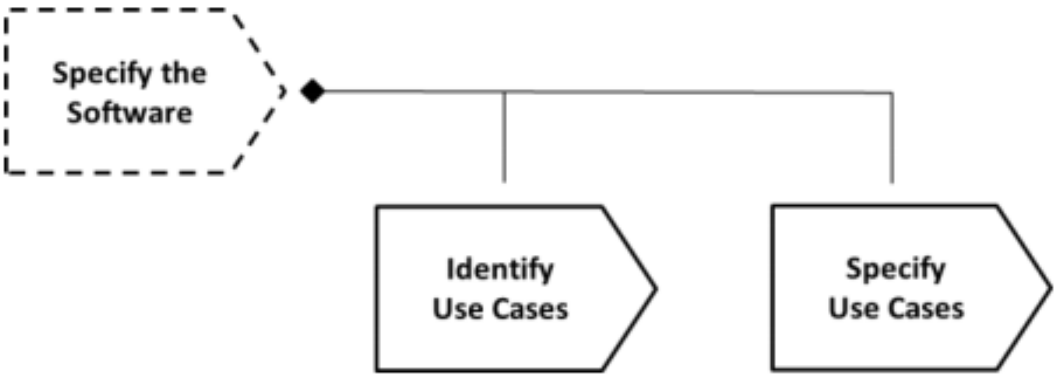


Рисунок 12 – Связь областей деятельности и работ на примере тестирования

Связи между работами, которые не подразумевают включение одной работы в другую, отображаются стрелками (рис. 13).

Уровни компетенций отображаются в виде прямоугольников со словесным описанием уровня компетенции и его числовым значением в кружочке внутри прямоугольника компетенции (рис. 14).

Шаблоны (паттерны) и ресурсы представляют собой общие концепции, которые могут быть привязаны к любому элементу языка и не пересекаются с динамической семантикой языка. Они не учитываются динамической семантикой языка, определенной в данной спецификации. Примеры ресурсов включают:

– шаблоны, привязанные к рабочим продуктам;

– скрипты или инструменты, привязанные к работам;

– учебные материалы или тесты, привязанные к компетенциям.

Шаблоны привязываются к другим объектам языка линиями, исходящими из круга с именем связи внутри (рис. 15).

Помимо этих основных элементов, язык содержит дополнительные элементы для детализации связей и обработки метаданных. Подробное и исчерпывающее описание языка содержится в стандарте [12].

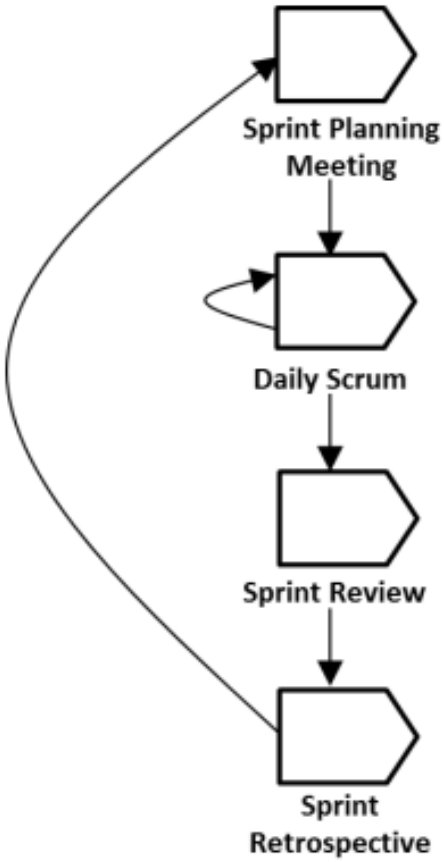


Рисунок 13 – Пример связей между работами

Рисунок 14 – Уровни компетенции «лидерство»

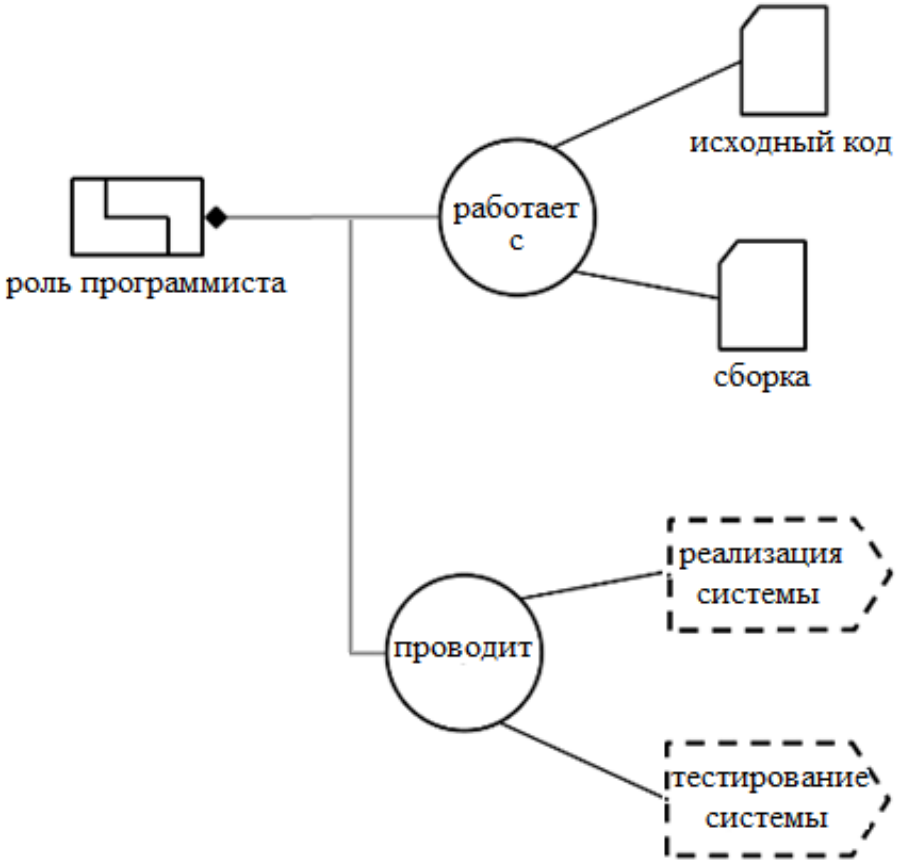


Рисунок 15 – Пример присоединения шаблона «роль программиста»

2.4.Модификация Essence для системной инженерии

При том, что маршруты проектирования и подходы к проектированию программного обеспечения и систем произвольного назначения во многом схожи, у них всё же имеются и отличия. Стандартное ядро Essence не полностью корректно отображается на процесс системного проектирования и не во всем соответствует его этапам.

Использование ядра Essence вне области программного обеспечения ограничивается как минимум из-за наличия «альфы» программной системы. Ограничения сегмента области «решения» требованиями и программной реализацией концептуально привязывает инструмент исключительно только к программным системам.

Также, одна из особенностей Essence заключается в отсутствии в стандарте архитектуры в явном виде. Разработчики стандарта связывают это с тем, что якобы некоторые программные системы не имеют её в явном виде, или, по крайней мере, она не учитываются в процессе проектирования. Тем не менее, почти во всех областях проектирования архитектура является одним из ключевых и важнейших понятий, вокруг и на основе которого, собственно, организуется процесс разработки. Поэтому для работы с инструментом Essence крайне желательно введение этой сущности как таковой и описание правил

работы с ней. Концептуально архитектура может быть включена как «альфа», как «суб-альфа» или как рабочий продукт.

В [13] предлагаются незначительные изменения ядра, которые позволяют Essence отражать более широкий спектр задач (заменяются обе «альфы» области решения, рис. 16).

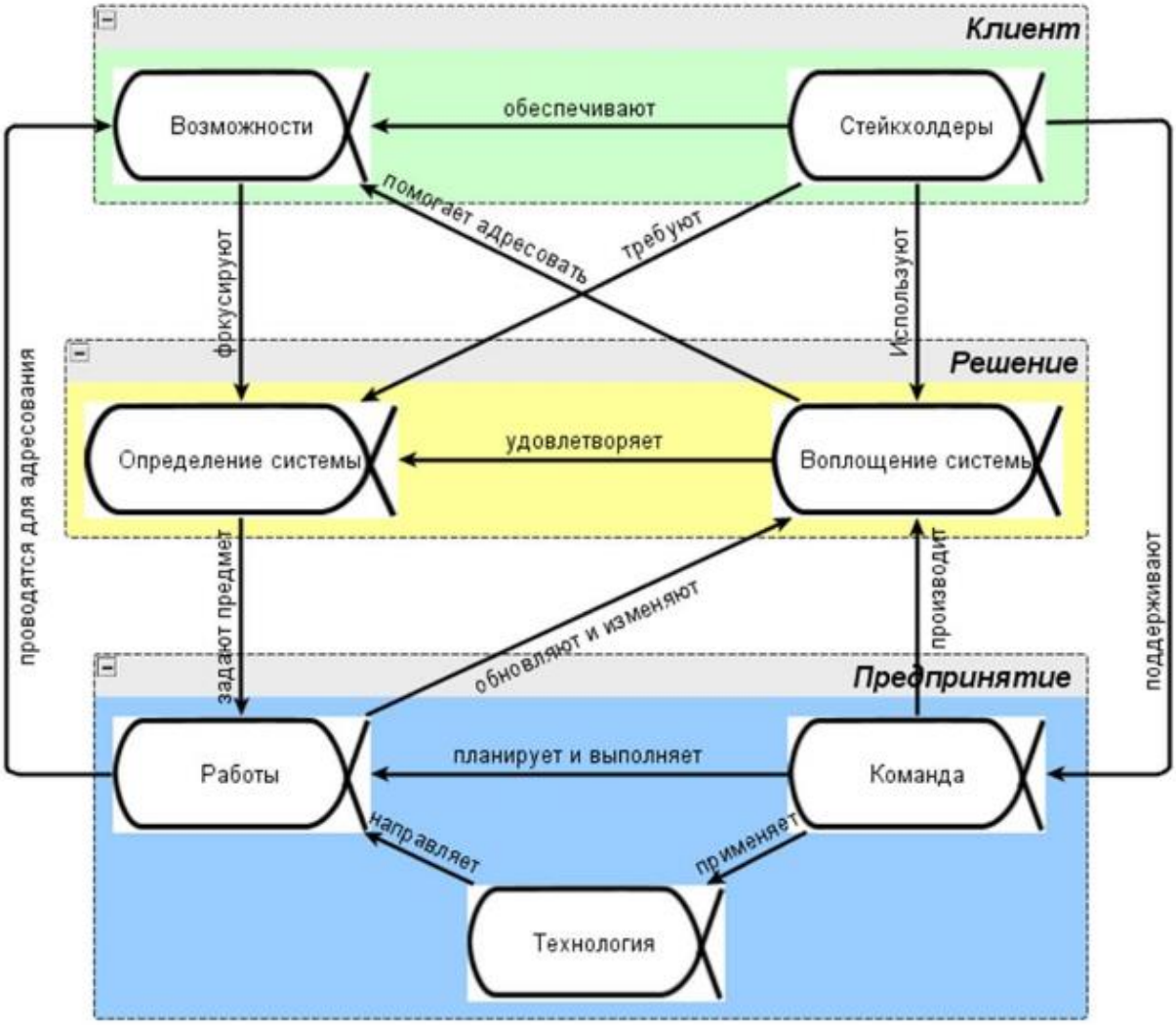


Рисунок 16 – Ядро Essence для системной инженерии [13]

При этом подходе архитектура является «альфой», вернее, «суб-альфой», а архитектурное описание представляет собой рабочий продукт, привязанной к «суб-альфе» архитектуры. Таким образом, сама архитектура остаётся абстрактным понятием, атрибутом состояния процесса проектирования, а её описание становится вполне осязаемым и доступным для работы предметом.

В [14] предлагается структура «альф» и «суб-альф», которая позволяет Essence соответствовать ISO 15288, в частности V-диаграмме. Согласно V-диаграмме, проектирование происходит в следующем порядке (рис. 17):

– согласуются требования предметной области;
– согласуются требования к системе;
– согласуются требования к отдельным элементам;

– запускается процесс разработки (в контексте киберфизических систем – это процесс разработки аппаратного и программного обеспечения, а также, возможно, механических и т. д. частей);

– верифицируются сначала элементы системы, потом система в целом, а в конце – система в «полевых условиях», в составе какой-то другой, более крупной системы, или же производится проверка на выполнение всех требований заказчика при рассмотрении системы как чёрного ящика.

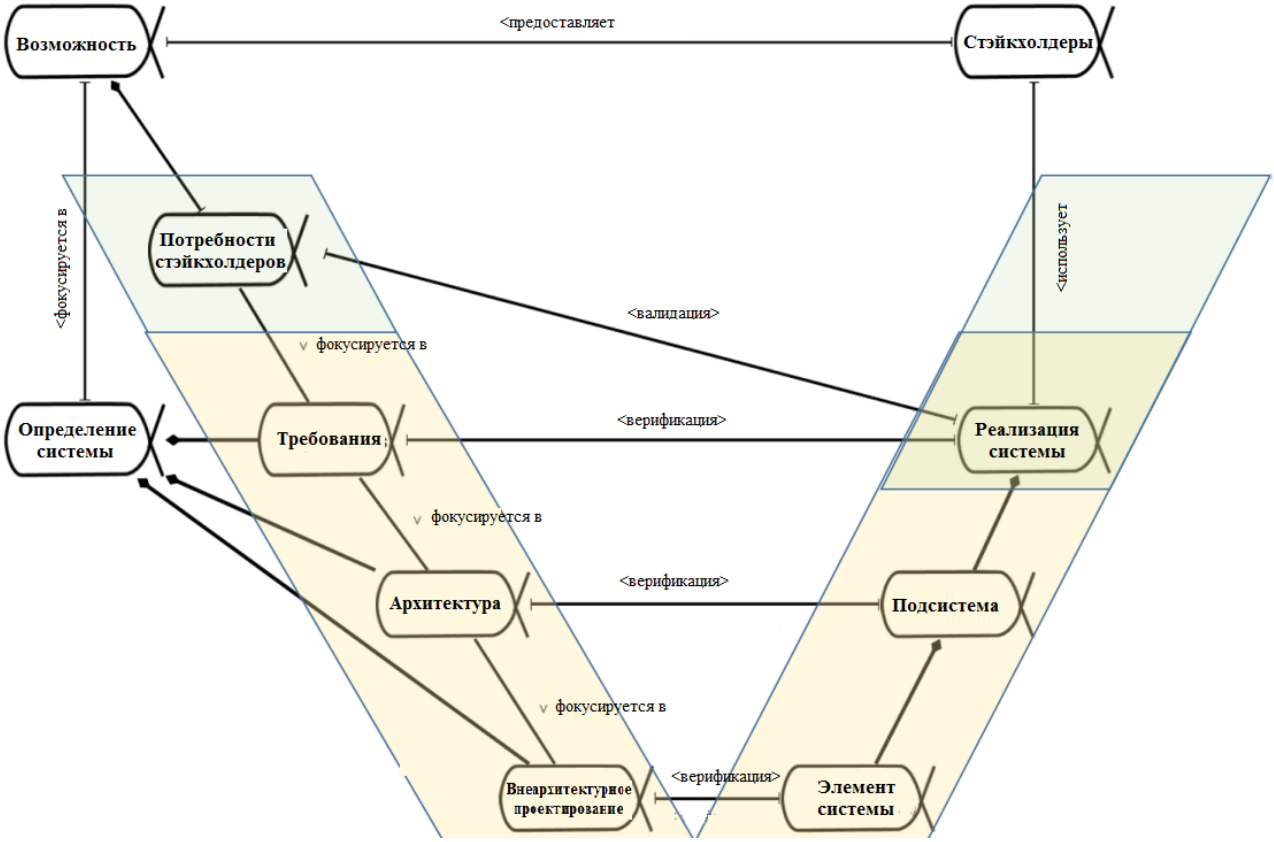


Рисунок 17 – V-диаграмма в терминах Essence [14]

2.5.Контрольные вопросы

1. Назначение стандарта Essence.
2. Перечислите и охарактеризуйте составные части ядра Essence.
3. Что такое альфа ядра Essence? Перечислите стандартные альфы ядра.
4. Опишите графическую нотацию изображения альф и суб-альф ядра Essence.
5. Какие области деятельности определяет ядро Essence?
6. Какие компетенции определяет ядро Essence? Перечислите и охарактеризуйте уровни владения компетенциями.
7. Перечислите основные элементы языка Essence и связи между ними.
8. Опишите модификации Essence для системной инженерии.

3. Метод ARCADIA и инструментальное средство Capella

3.1. Общее описание метода

ARCADIA (ARChitecture Analysis & Design Integrated Approach) – метод архитектурного проектирования систем и программного обеспечения на основе моделей (рис. 18). Capella – инструментарий для графического моделирования и архитектурного проектирования, реализующий рекомендации и принципы метода ARCADIA. Capella использует расширенный вариант языка SysML [15, 16, 17].

С помощью ARCADIA/Capella можно провести моделирование архитектуры будущей системы и на ранних этапах отследить возможность выполнения функциональных и нефункциональных требований. Метод ARCADIA представляет систему в виде набора связанных между собой моделей на разных уровнях абстракции и позволяет анализировать её с разных точек зрения, рассматривая каждый значимый аспект проектирования.

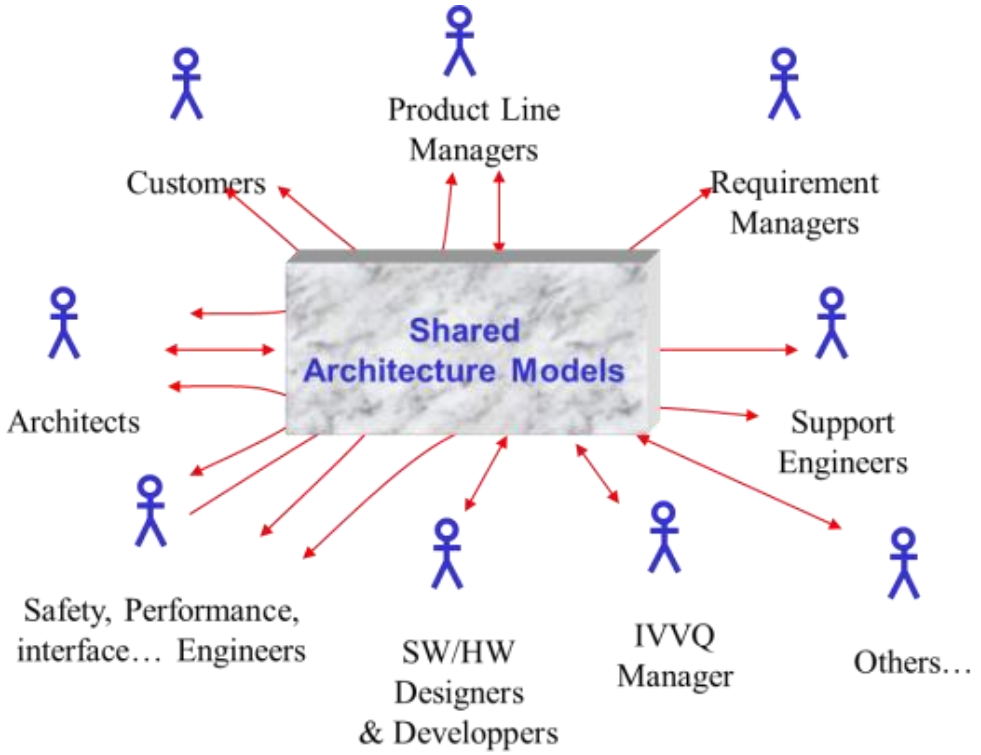


Рисунок 18 – Принцип проектирования на основе моделей с помощью ARCADIA/Capella

Метод ARCADIA определяет следующую последовательность шагов (рис. 19):

– определение потребностей пользователя;
– формализация требований к системе;
– разработка логической архитектуры;
– разработка физической архитектуры;
– формализация требований к компонентам.

Сначала описывается окружение и поведение системы как чёрного ящика, описывается функциональность и формализуются нефункциональные требования. Затем подбираются функции и формируется логическая архитектура, удовлетворяющая предъявленным требованиям. После этого логические компоненты отображаются на физические. Заключительным этапом является создание требований для разработчиков по каждому из разработанных компонентов. При этом на каждом этапе блоки и элементы с верхнего уровня абстракции отображаются на нижние, и на всём пути отслеживаются функциональные и нефункциональные ограничения.

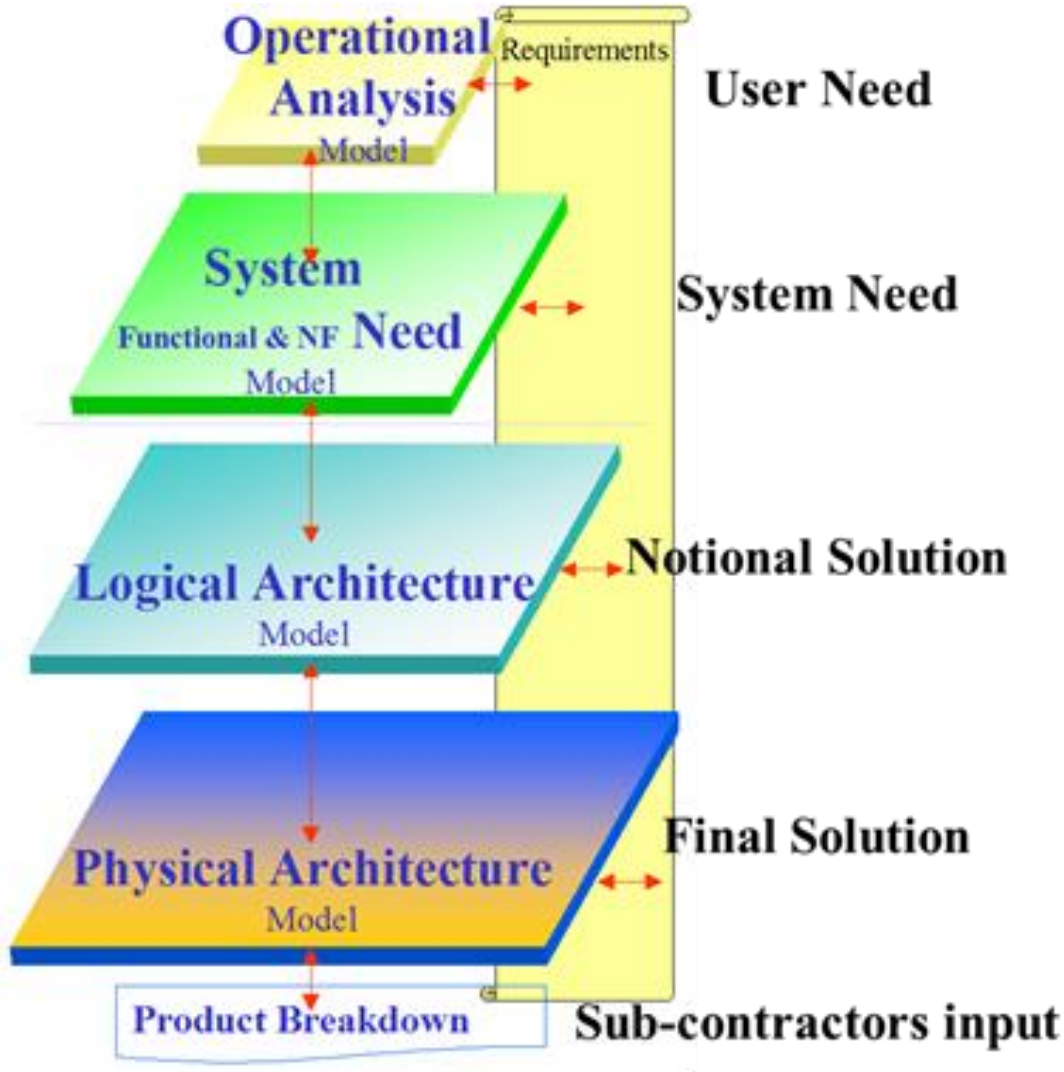


Рисунок 19 – Процесс проектирования в рамках ARCADIA/Capella

3.2. Анализ окружения и требований заказчика (Operational Analysis)

На первом этапе проектирования в соответствии с методом ARCADIA (Operational Analysis) описываются потребности заказчика и функционирование окружения, в которое будет встроена разрабатываемая система. Важно помнить: самой системы на этом этапе ещё нет, она вводится на следующем этапе – System Need Analysis. Поэтому описывать требования к системе не нужно.

На этом этапе может быть описано:

– поведение существующих систем, на которые можно опираться при дальнейшей разработке;

– функционирование процессов и окружения некоторой автоматизируемой области до применения автоматизации;

– описание взаимодействия сущностей и акторов между собой, которое будет впоследствии реализовано системой (при этом системы как бы нет).

Можно также заменить разрабатываемую систему некоторой сущностью (entity), в которую система входит. Однако это контрпродуктивно, так как описание поведения разрабатываемой системы и её взаимодействие с окружением всё равно необходимо будет производить на этапе System Need Analysis. Если сделать что-то подобное на этапе Operational Analysis, это будет двойной работой.

3.3. Описание требований к системе (System Need Analysis)

На втором этапе проектирования по ARCADIA впервые появляется система. Её поведение описывается как поведение «чёрного ящика» при помощи системных функций, никоим образом не определяющих внутреннее устройство системы, только поведение с точки зрения внешнего пользователя. Описывается «что», а не «как». На самом деле проектирование средствами Capella можно начинать именно с этого шага. Один из предложенных самими разработчиками маршрутов проектирования стартует именно с System Need Analysis.

На данном этапе необходимо полностью скрыть все особенности реализации системы и постараться абстрагироваться от уже имеющихся вариантов решений. Например, если предполагается реализовать систему в виде сервера и обращающегося к нему удалённого устройства – эти подробности реализации должны быть скрыты, а описывается просто «система», неким «волшебным» образом выполняющая все требования. Описание на этом этапе каких-либо особенностей реализации системы является ошибкой, это должно быть сделано на следующих этапах. Фактически описываются просто желаемые реакции на внешние воздействия, такие, какими мы хотим их видеть.

3.4. Разработка логической архитектуры (Logical Architecture)

На этом этапе требуется произвести детализацию введённых на предыдущем этапе абстрактных системных функций при помощи логических функций. Они покажут, как система будет обеспечивать необходимую функциональность. Также на этом этапе производится разбиение системы на модули (драйверы, подсистемы).

Описание предполагаемого внутреннего устройства системы и выделение отдельных самостоятельных блоков, над которыми будут работать разработчики – сложная и ответственная задача, поэтому относиться к ней надо со всей серьёзностью. От правильной организации и внутреннего устройства, от удобных и понятных иерархий зависит очень многое и при написании ПО, и при создании любых сложных систем.

Готовых решений, как разбивать систему на блоки, не существует. В руководстве от разработчиков Capella предлагается несколько вариантов:

– объединить компоненты с множеством связей в отдельные подсистемы с тем, чтобы связи между подсистемами были минимальны и их было легко описать; сведения о связях берутся из опыта предыдущих проектов, где использовались подобные компоненты;

– можно начать от высокоуровневых функциональных требований и организовать подсистемы вокруг основных функций системы;

– ещё один вариант – делить систему в соответствии с компетенциями существующих команд разработчиков.

Разбиение на физические платформы и способы реализации на данном этапе проводить не нужно.

3.5. Разработка физической архитектуры (Physical Architecture)

На этом этапе производится отображение логических функций на физические блоки. Решаются следующие вопросы:

– как система будет разрабатываться;

– что будет реализовано программно, а что аппаратно;

– описываются спецификации интерфейсов и т. д.

3.6. Контрольные вопросы

1. Назначение метода ARCADIA.
2. Опишите возможности среды моделирования Capella.
3. Последовательность шагов проектирования в ARCADIA.
4. Задачи этапа анализа окружения и требований в ARCADIA.
5. Задачи этапа описания требований в ARCADIA.
6. Задачи этапа разработки логической архитектуры в ARCADIA.
7. Задачи этапа разработки физической архитектуры в ARCADIA.

4. Лабораторный практикум 1. Знакомство со стандартом Essence

Задачи практикума:

– получение базового представления о понятиях, которыми оперируют методы программной и системной инженерии;

– изучение общей методологической основы для описания методов системной и программной инженерии – стандарта Essence;

– приобретение навыков анализа состояния проектов в области программной и системной инженерии (лабораторные работы 1.1–1.3);

– освоение инструментов описания методов программной и системной инженерии (лабораторные работы 1.4–1.5).

4.1. Описание лабораторных работ

В лабораторных работах первого практикума производится последовательное изучение на конкретном примере (дипломный проект студента) возможностей, предоставляемых стандартом Essence. В первых трёх лабораторных работах изучается в основном ядро программной инженерии, предлагаемое Essence, в четвёртой работе упор делается на изучении языка Essence, пятая работа направлена на углублённое понимание всего стандарта в комплексе.

4.2. Требования к сквозным темам индивидуальных заданий

В качестве материала для анализа в лабораторных работах 1.1–1.4 следует использовать дипломный проект, выполненный на предыдущих этапах обучения (далее – исследуемый проект). Если, по мнению студента, это невозможно, проблема решается в индивидуальном порядке.

4.3. Общие требования к выполнению лабораторных работ и оформлению отчетов

1. В отчетах по лабораторным работам 1.1–1.3 должны быть приведены карточки (detail cards) [18] с заполненными чек-листами, или должен быть иным образом произведён анализ состояний рассматриваемых альф по всем пунктам чек-листов.
2. Должно быть чётко обозначено, на каком этапе находится проект. Иначе говоря, в каком состоянии альфы (Alpha state) находится каждая рассматриваемая альфа.
3. Отчёты должны быть подготовлены в виде законченных документов, оформленных в соответствии с принятыми нормами (титульный лист, основное содержание лабораторной, выводы).
4. На титульном листе отчетов лабораторных работ 1.1–1.3 обязательно должна быть указана тема бакалаврской работы или той работы, которая взята за основу для анализа.
5. Отчеты о выполненных лабораторных работах принимаются в распечатанном или электронном виде.
6. Отчёты должны быть по возможности краткими, но исчерпывающими.
7. В процессе защиты отчета могут задаваться вопросы касательно рассматриваемого проекта и стандарта Essence, в частности – по каждому из вопросов карточек. Студент должен владеть терминологией и знать основу подхода Essence.
8. Отчёты последовательно идущих лабораторных работ не должны противоречить друг другу.
9. Отчёты разных студентов, даже из разных групп, не должны быть одинаковыми.

4.4. Задания

4.4.1. Лабораторная работа 1.1. Анализ области интересов «Пользователь»

Цель: приобретение навыков анализа систем и проектов в аспекте их взаимосвязи с пользователями (Customer area of concern).

Содержание отчёта:

1. Краткое описание исследуемого проекта в свободной форме (примерно половина страницы). Должно позволить сформировать общее представление о проекте, его цели и особенностях для человека «не в теме».
2. Перечисление и краткое описание возможностей (Opportunity). Определение Opportunity дано в стандарте.
3. Анализ проекта в рамках альфы «Возможность» (Opportunity).*
4. Перечисление и краткое описание стейкхолдеров. Должны быть перечислены все заинтересованные стороны.
5. Анализ проекта в рамках альфы «Заинтересованные стороны» (Stakeholders).*
6. Перечисление областей деятельности (Activity Spaces) и компетенций, относящихся к области работы с клиентом и/или маркетинга (Customer area of concern).
7. Выводы.

* Описывается в соответствии с пунктом 1 общих требований.

4.4.2. Лабораторная работа 1.2. Анализ области интересов «Решение»

Цель: приобретение навыков анализа систем и проектов в аспекте разрабатываемого решения (Solution area of concern).

Содержание отчёта:

1. Схема и краткое описание архитектуры разрабатываемой системы в произвольной форме.
2. Перечисление и краткое описание требований к проекту. Требования должны давать исчерпывающее представление о том, что за система разрабатывается.
3. Анализ проекта в рамках альфы «Требования» (Requirements).*
4. Перечисление и краткое описание ключевых принятых архитектурных решений.
5. Анализ проекта в рамках альфы «Программная система/Система» (Software System).*
6. Перечисление областей деятельности (Activity Spaces) и компетенций, относящихся к области разработки (Solution area of concern).
7. Выводы.

* Описывается в соответствии с пунктом 1 общих требований.

4.4.3. Лабораторная работа 1.3. Анализ области интересов «Менеджмент»

Цель: приобретение навыков анализа систем и проектов в аспекте организации рабочего процесса (Endeavor area of concern).

Содержание отчёта:

1. Перечисление и краткое описание основных работ и этапов.
2. Анализ проекта в рамках альфы «Работа» (Work).*
3. Перечисление и краткое описание всех участников рабочего процесса и их ключевых компетенций.
4. Анализ проекта в рамках альфы «Команда» (Team).*
5. Перечисление и краткое описание выбранных методов и инструментов, которые использовались в работе.
6. Анализ проекта в рамках альфы «Технология работы» (Way of Working).*
7. Перечисление областей деятельности (Activity Spaces) и компетенций, относящихся к области менеджмента (Endeavor area of concern).
8. Выводы.

* Описывается в соответствии с пунктом 1 общих требований.

4.4.4. Лабораторная работа 1.4. Описание метода с помощью языка Essence

Цель: приобретение навыков описания методов и практик средствами языка Essence.

Содержание отчета:

1. Описание в произвольной форме какого-либо метода, желательно реально применявшегося исполнителем в разработке (если таких нет, можно брать любой).
2. Описание выбранного метода средствами языка Essence. Описание должно быть не избыточным, но достаточным, чтобы без пункта 1 получить некоторое базовое представление о методе. Обычно достаточно 2-3 диаграмм с краткими текстовыми пояснениями. Возможно добавление карточек в предложенном стандартом формате, подробно описывающих каждый из элементов. Желательно наличие диаграмм, раскрывающих временной аспект (последовательности действий и т. д.).
3. Выводы.

Отдельные требования:

1. В рамках одной учебной группы выбранные для описания методы не должны повторяться.
2. Нельзя брать те методы, которые приведены в примерах в самом стандарте, в библиотеках на сайте Ивара Якобсона и в презентациях А.И. Левенчука (scrum, user story).

4.4.5. Лабораторная работа 1.5. Предложение по улучшению стандарта

Цель: приобретение навыков описания и модификации методик разработки, закрепление понимания архитектуры метода Essence.

Содержание отчета:

1. Описание проблем и недостатков стандарта Essence.
2. Описание предлагаемых расширений или изменений, которые могут включать изменения как языка, так и ядра.
3. Добавить примеры использования модифицированного варианта с демонстрацией его преимуществ, желательно со сравнением «было»/«стало».
4. Выводы.

5. Лабораторный практикум 2. Анализ проектных альтернатив в реализации КФС

Задачи практикума:

– изучение высокоуровневого (системного) проектирования на практическом примере;

– освоение навыков составления технических заданий;

– получение представления об исследовании пространства проектных решений (design space exploration);

– получение представления о множественности проектных альтернатив и расширение знаний о возможных способах реализации вычислительных задач (разные модели вычислений, платформы, альтернативные архитектуры, алгоритмы и т. д., см. рис. 20);

– сравнение разных способов реализации на конкретных практических примерах.

5.1. Описание лабораторных работ

В рамках выполнения лабораторных работ требуется:

– провести архитектурное (системное) проектирование для достаточно сложной и комплексной задачи;

– произвести исследование пространства проектных решений с оценкой жизнеспособности альтернатив через реализацию прототипов отдельной (относительно небольшой) части системы.

Лабораторные работы имеют одну сквозную тему, которая выбирается на первых занятиях. Каждая лабораторная работа является этапом одного большого проекта.

Темы лабораторных работ отражают типичный маршрут проектирования, включая:

– составление технического задания;

– исследование пространства проектных решений;

– выявление проблемных мест;

– реализацию прототипов частей системы на разных платформах с применением разных моделей вычислений, алгоритмов и/или технологических стеков для оценки их эффективности и адекватности задаче; один из оцениваемых вариантов обязательно реализуется средствами языка C/C++ на учебном стенде SDK-1.1M [20] или аналогичном микроконтроллерном устройстве.

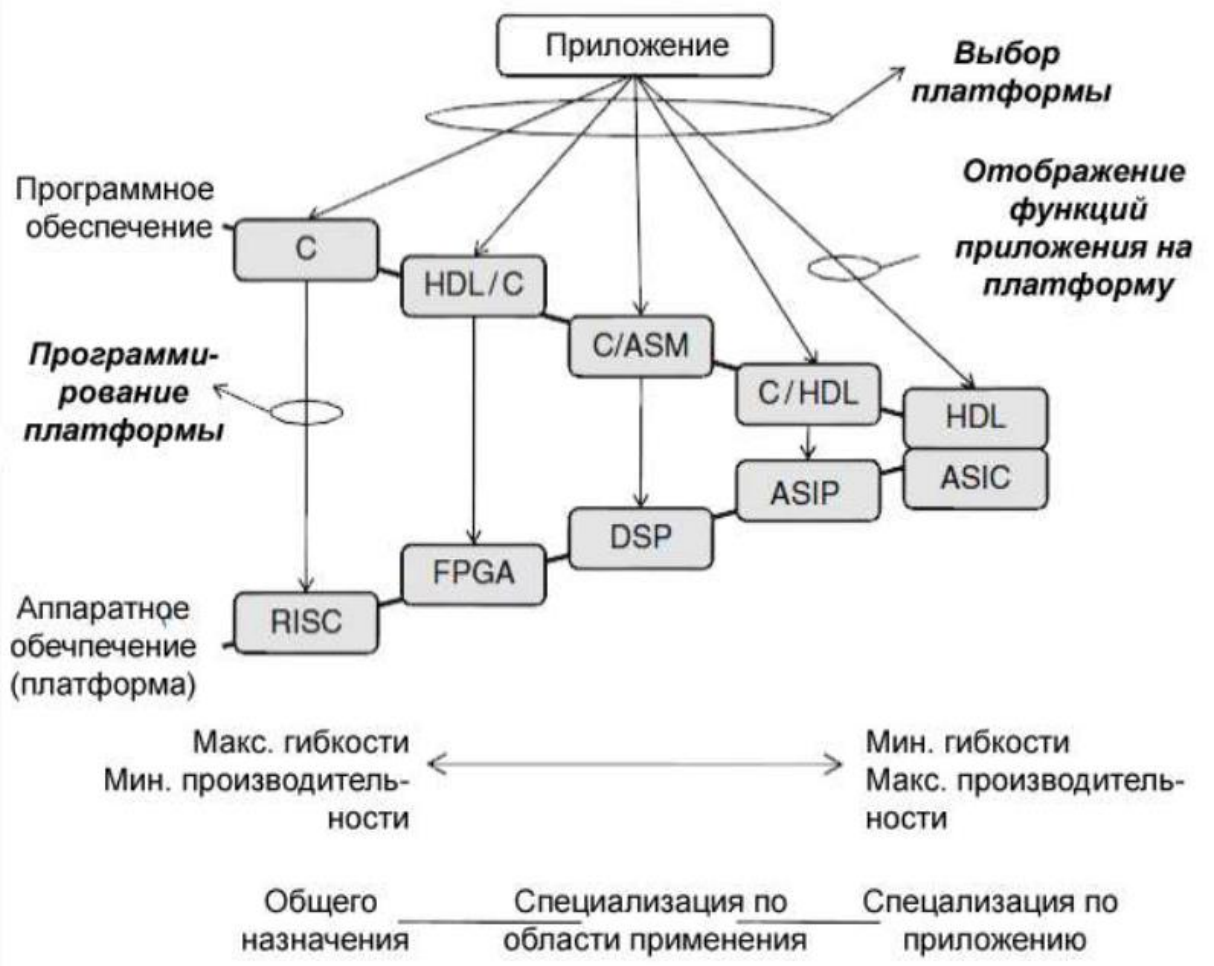


Рисунок 20 – Пример выбора отображений функциональности на архитектуру [19]

5.2. Требования к сквозным темам индивидуальных заданий

В качестве задания должна быть выбрана конкурентоспособная и практически реализуемая прикладная задача (продукт), которая требует разработки решения средствами вычислительной техники.

Проектируемая система должна быть достаточно сложной (либо её проработка должна вестись на достаточно глубоком уровне).

Желательно, чтобы предполагалось наличие в результирующей системе элементов с относительно трудоёмкими вычислениями (типичные примеры – распознавание изображений, шифрование) и/или задачи обеспечения коммуникаций или управления с соответствующими жёсткими ограничениями (требования реального времени, пониженного энергопотребления и т. д.).

Примеры систем:

1. Система управления «умным домом» с видеонаблюдением и распознаванием изображений/речи. Система реализует некоторый набор функций «умного дома», одной из которых является выполнение определенных действий при распознавании лица хозяина/автомобильного номера, или по голосовой команде хозяина.

2. Беспилотный летательный аппарат, автономный робот или автопилот автомобиля, с распознаванием изображений с камер, обработкой данных с лидаров и т. д.

3. Комплексные системы типа «умный город», «Индустрия 4.0», «интернет вещей» и т. д., например – системы «умного» управления светофорами на общегородском уровне.

4. Система распределённых вычислений на базе управляющих устройств. Сеть из управляющих устройств, каждое из которых выполняет некоторую функцию управления в реальном времени, а свободное процессорное время тратит на выполнение частей каких-либо глобальных вычислений, распределяемых между всеми узлами.

5. Прикладная вычислительная система, реализованная с использованием собственного компилятора/транслятора (программируемые логические контроллеры и т. д.). Должна позволять описывать алгоритмы на предметно-ориентированном языке, удобном для решения прикладной задачи, и генерировать исполняемые программы для определённой аппаратуры.

6. Прикладная вычислительная система, реализующая высокопроизводительные вычисления средствами GPU, DSP-сопроцессоров или эффективно использующая параллельное программирование (например – некоторое ПО для работы с графикой). Система должна обеспечивать существенный прирост в производительности для некоторой «тяжеловесной» прикладной задачи.

7. Виртуальная машина (например, Java-машина), реализованная в рамках некоторой системы управления. Виртуальная машина должна позволять в определенных рамках «программировать» систему управления и реализовать прикладную функциональность, предоставляя удобные элементы для реализации необходимых функций и скрывая ненужные подробности.

5.3. Общие требования к выполнению лабораторных работ и оформлению отчетов

1. Перед началом выполнения работы необходимо согласовать тему с преподавателем.

2. Отчёты должны быть подготовлены в виде законченных документов, оформленных в соответствии с принятыми нормами (титульный лист, основное содержание лабораторной, выводы).

3. На титульном листе обязательно должна быть указана общая (сквозная) тема всех лабораторных работ.

4. Отчеты о выполненных лабораторных работах принимаются в распечатанном или электронном виде. Возможен вариант одного общего отчёта, дополняемого после каждого этапа.

5. По лабораторным работам составляются подробные отчёты в объеме, достаточном для подробного описания проделанной работы. Они должны включать исходные коды и техническую информацию, необходимую для понимания того, как работа была выполнена.

6. На определённых этапах в отчёты включаются модели, схемы, описания алгоритмов, рабочие прототипы.

7. Для лабораторных работ 2.3–2.5 должна быть проведена демонстрация работоспособности разрабатываемого решения (программы, устройства). В отдельных случаях возможна демонстрация в симуляторе.

5.4.Задания

5.4.1.Лабораторная работа 2.1. Разработка технического задания

Цель: приобретение навыков написания технических заданий.

В работе требуется подготовить описание, дающее базовое понятие о выбранной предметной области (краткий реферат).

Выполнить обзор рынка и существующих технических решений (не менее трёх) по выбранной теме. Должны быть описаны известные подходы, системы аналогичного или близкого назначения. Если прямых аналогов нет – системы, реализующие часть предполагаемой функциональности. Необходимо выделить и сравнить основные характеристики существующих решений, наиболее значимые параметры, «фичи», достоинства и недостатки. Материалы удобно представлять в табличном виде.

Основная цель выполнения обзора рынка – поиск незанятой «ниши», то есть такого набора характеристик продукта, который будет конкурентоспособным. Для этого кратко описывается будущий продукт, самые сильные стороны, предполагаемые преимущества перед существующими аналогами.

На основе проведённого исследования составить техническое задание. описываются требования к разрабатываемому продукту (функциональные, нефункциональные, при необходимости – организационные).

Готовность и адекватность требований проверить с помощью чек-листов альфы «Требования» (Reqirements) стандарта Essence (см. лабораторную работу 1.2). Достаточно привести карточки с заполненными чек-листами и название состояния соответствующей альфы, а также уметь обосновать их.

Требования к оформлению:

1. Техническое задание оформляется как отдельный, законченный, самостоятельный документ (как будто предыдущих разделов нет).

2. Требования оформляются списком, одно требование на строку.

3. Каждое отдельное требование – это отдельный самостоятельный пункт или аспект разрабатываемой системы, который может быть помечен как «выполнено» или «не выполнено». Одно требование не может быть разбито на

несколько отдельных требований. Если это возможно, значит, требование неверно.

4. Перечисление требований начинается с описания системы: что это за система, зачем она нужна, где применяется и т. д.

5. Функциональные требования должны максимально подробно описывать поведение системы, как чёрного ящика: то, что от неё ждёт пользователь.

6. Количество функциональных требований должно быть достаточным для понимания человеком «со стороны» сути проектируемой системы и всех значимых аспектов её функционирования.

7. Объём раздела функциональных требований: не менее страницы текста.

8. Требования могут описываться при помощи моделей инструментария Capella (см. лабораторную работу 3.2).

9. Нефункциональные требования должны включать в себя самые очевидные и важные параметры, не связанные напрямую с поведением системы или связанные косвенно (энергопотребление, надёжность, безопасность, требования реального времени, эргономика и т. д.). Пример – носимое портативное устройство. Очевидные требования для него – эргономичность, пониженное энергопотребление, компактный размер, небольшой вес.

10. Список нефункциональных требований должен быть как можно более широким, но не избыточным. Обычно достаточно нескольких штук.

11. Крайне желательно указывать приоритет требований.

12. Требования могут группироваться и структурироваться, например – при помощи маркированных/нумерованных списков с несколькими уровнями вложенности.

Примеры аспектов, которые должны рассматриваться в нефункциональных требованиях (для каждого продукта может быть свой набор):

– безопасность, защита от атак и «от дурака»;
– надёжность, отказоустойчивость, ремонтопригодность;
– повторное использование решений в других проектах;
– переносимость (например, программных решений на другие платформы);
– инструментальный аспект;
– экономический аспект;
– верифицируемость и пригодность к тестированию;
– трассируемость, возможность выявлять ошибки в ходе выполнения;
– простота доработки и обновления функциональности, включая удалённое обновление ПО;
– проблемы интеграции готовых блоков и ведения существующих проектов;
– ограничения по энергопотреблению;
– выполнение в реальном времени;
– физические параметры, вес, размеры;
– устойчивость к климатическим факторам;
– устойчивость к электромагнитным помехам;
– устойчивость к радиации;

– простота использования конечным пользователем;

– факторы, создающие комфорт конечному пользователю.

Кроме этого, требования могут содержать такие разделы, как: описание режимов работы, ограничения на применение, ограничения на реализацию (применяемое аппаратное обеспечение, программные и инструментальные средства), перспективные возможности с учетом ограничений (например, предусмотреть возможность введения нового режима работы).

Содержание отчёта:

1. Титульный лист.

2. Краткое описание предметной области и задачи – так, чтобы она была понятна человеку, который не знаком с контекстом.

3. Описание и сравнение существующих решений, предпочтительно в табличном виде, не менее 3-х штук.

4. Краткое описание концепции предполагаемого продукта.

5. Техническое задание как отдельный документ, включающий в себя описание требований (функциональных, нефункциональных, при необходимости – организационных и т. д.).

6. Анализ качества проработки требований к системе средствами Essence. Проверяется с помощью чек-листов альфы «Требования» (Requirements) (см. лабораторную работу 1.2).

5.4.2. Лабораторная работа 2.2. Предварительное исследование пространства проектных решений

Цель: получение базового представления о таком понятии, как исследование пространства проектных решений, и изучение существующих возможных проектных альтернатив при проектировании КФС.

На основе полученного в предыдущей лабораторной работе технического задания требуется произвести «исследование пространства проектных решений» (design space exploration). Составляется список узловых точек проектирования – мест, в которых может быть выбрано несколько альтернатив, которые могут касаться структурной организации, элементной базы, используемых технологических стеков.

Например, реализацией какой-либо функции или отдельного блока может быть:

– программная реализация на ПК общего назначения, сервере или микроконтроллере;

– bare-metal программа или программа, функционирующая с помощью ОС, в том числе ОС реального времени (RTOS);

– аппаратная реализация на ПЛИС или ASIC;

– смешанная аппаратно-программная реализация;

– программная реализация на soft-core микропроцессоре (процессор на ПЛИС);

– распределённая вычислительная система, облачные, туманные вычисления;

– реализация с применением нейронных сетей;

– реализация на виртуальной машине;

– реализация с автоматическим синтезом различных решений и т. д.

Допустимы варианты с использованием разных моделей вычислений (Model of Computation). Примеры: конечный автомат, лямбда-исчисления, сети Петри и Кана. Допустимы разные варианты как описания, так и выполнения. Возможно описание решение одной задачи с помощью разных алгоритмов, реализация разных протоколов, разных принципов взаимодействия.

Поиск должен быть максимально широким и захватывать максимальное количество различных решений. Информация может быть представлена графически или в текстовом формате.

Для каждого решения производятся предварительные количественные и качественные оценки, позволяющие выявить оптимальные решения.

В отчёте должно быть не менее 15 разных вариантов (например, 5 точек по 3 варианта).

Представление проектных альтернатив можно выполнить средствами Capella (лабораторные работы 3.3–3.4). В таком случае, для демонстрации проектных альтернатив достаточно вставить скриншоты с различными вариантами логического разбиения (или различными вариантами физической реализации).

Необходимо оценить готовность системы (архитектуры системы) средствами Essence. Проверяется с помощью чек-листов альфы «Программная система/Система» (Software System) (см. лабораторную работу 1.2).

Содержание отчёта:

1. Титульный лист с названием сквозной темы (если отчёт не «подшивается» к предыдущему).

2. Перечисление точек ветвления в маршруте проектирования (можно в графическом виде: например, дерево решений; можно средствами Capella, см. лабораторные работы 3.3–3.4) и описание возможных путей решения в каждой из точек.

3. Сравнение решений, критерии: насколько будут выполняться выдвинутые в лабораторной работе 2.1 требования, примерная оценка трудозатрат, стоимости изготовления и проектирования, производительность, энергопотребление, надёжность и т. д.

4. Анализ проработки решения средствами Essence – проверяется с помощью чек-листов альфы «Программная система/Система» (Software System) (см. лабораторную работу 1.2).

5.4.3. Лабораторная работа 2.3. Использование учебного стенда SDK-1.1M в качестве платформы для одного из вариантов реализации (или реализации части функциональности)

Цели:

– приобретение практических навыков программирования микроконтроллеров;

– получение представления о возможностях и сложностях создания соответствующей проектной альтернативы при создании КФС.

Совместно с преподавателем выбирается некоторая часть разрабатываемой системы, работу которой требуется детализировать для более детального исследования пространства проектных решений (например, написать небольшую программу, позволяющую приблизительно оценить характеристики и работу всей будущей системы).

В лабораторной работе требуется составить подробное и понятное техническое задание на данную часть работы. Для реализации используется учебный стенд SDK-1.1M [20]. Стенд представляет собой функционально законченное устройство на базе микроконтроллера семейства STM32, имеет набор периферийных устройств, может связываться с «внешним миром» через Ethernet и USB, имеет слот для SD-карт и разъёмы для установки плат расширения для Arduino.

Содержание отчёта:

1. Титульный лист с названием сквозной темы (если отчёт не «подшивается» к предыдущему).
2. Краткое техническое задание на выполняемую часть работы.
3. Предварительная оценка трудозатрат на выполнение работы, сравнение с фактическими трудозатратами.
4. Функциональное описание реализуемого компонента.
5. Описание архитектуры компонента (желательно – графическое, дающее представление об устройстве и взаимодействии компонентов ПО, пример блок-схема).
6. Исходный код компонента.
7. Протокол тестирования.
8. Отчёт о результатах анализа реализации.
9. Описание возникших в ходе работы проблем.

5.4.4. Лабораторные работы 2.4/2.5. Создание альтернативных реализаций

Цель: изучение возможных проектных альтернатив при создании КФС.

Требуется создать альтернативные реализации той же части проекта, что и в предыдущей лабораторной работе (если это невозможно, задание на лабораторные работы 2.4/2.5 следует согласовать с преподавателем), и произвести их сравнение. Общей целью всех лабораторных работ, в частности 2.2–2.5, является выявление оптимальных вариантов реализаций для данного технического задания при существующих ограничениях.

В качестве альтернативных реализаций некоторого программного решения могут быть выбраны какие-либо варианты из намеченных в лабораторной работе 2.2.

Примеры (дублирует/дополняет список в лабораторной работе 2.2):

– программная реализация;

– аппаратные IP-блоки на языках описания аппаратуры Verilog, VHDL, SystemVerilog и т. д.;

– нейронные сети;

– модели в Simulink (MATLAB), LabVIEW, Ptolemy и им подобных средах;

– программа на Assembler (возможен вариант с использованием asm-вставок в код на языке C);

– эффективное использование аппаратных ускорителей (сопроцессоров) – DSP, GPU;

– многоядерная/многопроцессорная реализация;

– распределённая вычислительная система;

– использование собственного компилятора/транслятора/генератора шаблонов;

– «экзотические» языки (Lisp, Smalltalk и т. д.);

– использование виртуальных машин;

– использование операционных систем реального времени и т. д.

Содержание отчёта:

1. Титульный лист с названием сквозной темы (если отчёт не «подшивается» к предыдущему).
2. Техническое задание.
3. Собственная оценка трудозатрат на написание программы, сделанная до начала работы, сравнение с реальными трудозатратами.
4. Краткий реферат (не больше половины страницы) об используемом языке (программирования, описания аппаратуры, моделирования и т. д.) и инструментальных средствах.
5. Функциональное описание реализуемого компонента.
6. Описание архитектуры компонента (желательно графическое, дающее представление об устройстве и взаимодействии компонентов ПО, например блок-схема).
7. Листинг программы или иное описание реализации.
8. Сравнение с предыдущими решениями (производительность, энергопотребление, надёжность и т. д.).
9. Сравнение эффективности различных проектных альтернатив.

5.4.5. Лабораторная работа 2.6. Выбор, описание и обоснование архитектуры

Цель: получение практических навыков архитектурного описания сложных систем.

Требуется принять решения относительно всех обозначенных в лабораторной работе 2.2 проектных альтернатив и подготовить подробное описание результирующей архитектуры системы, включающее данные решения, но не исчерпывающееся ими.

Привести обоснование всех принятых решений.

Привести описание архитектуры системы, желательно в графическом виде. Может быть выполнено средствами Capella (лабораторная работа 3.4).

Оценить готовность системы (архитектуры системы) средствами Essence. Проверяется с помощью чек-листов альфы «Программная система/Система» (Software System) (см. лабораторную работу 1.2).

Содержание отчёта:

1. Титульный лист с названием сквозной темы (если отчёт не «подшивается» к предыдущему).
2. Описание архитектуры системы, предпочтительно в графическом виде, может выполняться средствами Capella (лабораторная работа 3.4).
3. Краткое обоснование всех основных архитектурных решений.
4. Анализ проработки системы средствами Essence. Проверяется с помощью чек-листов альфы «Программная система/Система» (Software System) (см. лабораторную работу 1.2).

6. Лабораторный практикум 3. Спецификация архитектуры КФС средствами Capella

Задачи практикума:

– изучение метода ARCADIA и реализующего его принципы инструмента Capella;

– знакомство с типовым маршрутом высокоуровневого (системного) проектирования КФС на практических примерах.

6.1. Описание лабораторных работ

Выполнение лабораторных работ предполагает проработку архитектуры системы (высокоуровневое, системное проектирование) средствами Capella. Лабораторные работы предполагают более глубокую проработку материалов из лабораторного практикума 2 и позволяют более наглядно представить вещи, о которых идёт речь в лабораторных работах 2.1, 2.2, 2.6.

Лабораторные работы имеют общую сквозную тему, которая повторяет тему лабораторного практикума 2.

Темы конкретных работ отражают проработку архитектуры системы в соответствии с этапами метода ARCADIA.

6.2. Требования к сквозным темам индивидуальных заданий

Тема лабораторных работ повторяет сквозную тему, выбранную для лабораторного практикума 2.

6.3. Общие требования к выполнению лабораторных работ и оформлению отчетов

1. Оформление отчетов выполняется с использованием материалов проектирования в среде Capella. Проект в Capella предъявляется преподавателю.
2. При защите лабораторных работ должны быть предоставлены отчёты для лабораторного практикума 2 (как минимум, отчёт для лабораторной работы 2.1).

6.4.Задания

6.4.1.Лабораторная работа 3.1. Анализ окружения и требований заказчика

В работе требуется произвести первый этап проектирования в соответствии с методом ARCADIA, Operational analysis: описать потребности заказчика и функционирование окружения проектируемой системы.

Для выполнения лабораторной работы достаточно проработать все основные диаграммы этапа Operational analysis.

Минимальный набор диаграмм:

– диаграмма, описывающая “capabilities” – возможности и функции окружения (Operational Capabilities diagram);

– диаграмма, описывающая операционные сущности (Operational Entity Breakdown diagram);

– диаграммы, описывающие поведение окружения (Operational Entity Scenario, Operational Architecture);

– диаграмма взаимодействия операционных активностей (Operational Activity Interaction diagram);

– дополнительное задание: спецификация операционного процесса.

6.4.2.Лабораторная работа 3.2. Описание требований к системе

В работе требуется произвести второй этап проектирования в соответствии с методом ARCADIA, System Need Analysis: описать требования к системе, как к чёрному ящику.

Для выполнения лабораторной работы достаточно проработать все основные диаграммы этапа System Need Analysis.

Минимальный набор диаграмм:

– любая диаграмма по выбору, описывающая миссию и system capabilities (в UML и SysML аналог system capabilities – use cases);

– иерархия системных акторов и их связь с системой (Contextual System Actors diagram);

– диаграммы, описывающие взаимодействие системы с окружением (System Architecture diagram, Exchange Scenario);

– диаграммы взаимодействия системных функций (Functional Dataflow Diagram);

– дополнительное задание: описание хотя бы одного операционного процесса, включающего системные и внешние функции.

6.4.3.Лабораторная работа 3.3. Создание логической архитектуры

В работе требуется произвести третий этап проектирования в соответствии с методом ARCADIA, Logical Architecture: произвести логическое разбиение системы.

Кроме того, в работе требуется предложить несколько вариантов внутреннего устройства системы. Это можно сделать следующим образом:

1. Сохранить «эталонный» вариант разбиения системы.

2. Подумать, какие могут быть альтернативные способы организации системы, и либо отразить это в отчёте, либо создать разные проекты/диаграммы, иллюстрирующие разные ветки проектирования системы.

3. Создать несколько альтернативных вариантов и либо сохранить их как отдельные проекты, либо сделать скриншоты, дающие понять суть изменений.

Отчёт к лабораторной работе должен содержать скриншоты или текстовое описание принятых решений.

Минимальный набор диаграмм проекта:

– разбивка на логические блоки (Logical Component Breakdown diagram);

– диаграммы, позволяющие сопоставить между собой системную и логическую архитектуры (System Architecture diagram, Logical Architecture diagram);

– при наличии сложного поведения и взаимодействия компонентов – сценарии взаимодействия (Exchange Scenario diagram).

Один из вариантов оформления: сделать «эталонный» проект, а в отчёте описать при помощи скриншотов альтернативные варианты, выделив и явно показав отличия.

6.4.4. Лабораторная работа 3.4. Создание физической архитектуры

В работе требуется произвести четвертый этап проектирования в соответствии с методом ARCADIA, Physical Architecture: описать физическую архитектуру системы.

Кроме того, в работе требуется предложить несколько вариантов физической реализации системы. Это можно сделать следующим образом:

1. Подумать, какие могут быть альтернативные способы физической реализации системы.

2. По аналогии с предыдущим шагом описать несколько вариантов, и либо отразить это в отчёте, либо создать разные проекты/диаграммы, иллюстрирующие разные ветки проектирования системы.

Отчёт к лабораторной работе должен содержать скриншоты или текстовое описание принятых решений.

Минимальный набор диаграмм:

– разбивка на физические компоненты (Physical Component Breakdown diagram);

– диаграммы, позволяющие сопоставить между собой физическую и логическую архитектуры (Physical Architecture diagram, Logical Architecture diagram);

– при необходимости – прочие диаграммы.

Один из вариантов оформления – сделать «эталонный» проект, а в отчёте описать при помощи скриншотов альтернативные варианты, выделив и явно показав отличия.

Рекомендуемая литература

1. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. Санта-Клара, Калифорния, 2008.
2. Непейвода Н.Н., Скопин И.Н. Основания программирования. М.: Институт компьютерных исследований, 2003.
3. Брукс Ф. Мифический человеко-месяц, или Как создаются программные системы. М.: Символ-Плюс, 2010.
4. Керниган Б., Ритчи Д. Язык программирования Си. М.: Издательский дом «Вильямс», 2009.
5. Керниган Б., Пайк Р. Практика программирования. М.: Издательский дом «Вильямс», 2004.
6. Рэймонд Э.С. Искусство программирования для UNIX. М.: Издательский дом «Вильямс», 2005.
7. Andy D. Pimentel. Exploring Exploration: A Tutorial Introduction to Embedded Systems Design Space Exploration.
8. Тимофеев В. Как писать программы без ошибок. URL: http://www.pic24.ru/doku.php/osa/articles/encoding_without_errors (дата обращения: 14.01.2022).
9. Быковский С.В., Горбачев Я.Г., Ключев А.О., Пенской А.В., Платунов А.Е. Сопряжённое проектирование встраиваемых систем (Hardware/Software Co-Design). Часть 1: Учебное пособие – СПб: Университет ИТМО, 2016. URL: https://books.ifmo.ru/book/1861/sopryazh%D1%91nnoe_proektirovanie_vstraivaemyh_sistem_(Hardware/Software_Co-Design)._chast_1:_uchebnoe_posobie.htm (дата обращения: 14.01.2022).
10. Быковский С.В., Горбачев Я.Г., Ключев А.О., Пенской А.В., Платунов А.Е. Сопряжённое проектирование встраиваемых систем (Hardware/Software Co-Design). Часть 2: Учебное пособие – СПб: Университет ИТМО, 2016. URL: https://books.ifmo.ru/book/1862/sopryazh%D1%91nnoe_proektirovanie_vstraivaemyh_sistem_(Hardware/Software_Co-Design)._chast_2:_uchebnoe_posobie.htm (дата обращения: 14.01.2022).
11. Платунов А.Е, Постников Н.П. Высокоуровневое проектирование встраиваемых систем (часть 1). – СПб.: НИУ ИТМО, 2011. URL: https://books.ifmo.ru/book/682/vysokourovnevoe_proektirovanie_vstraivaemyh_sistem.htm (дата обращения: 14.01.2022).
12. Платунов А.Е, Постников Н.П. Высокоуровневое проектирование встраиваемых систем (часть 2). – СПб.: НИУ ИТМО, 2013. URL: https://books.ifmo.ru/book/910/vysokourovnevoe_proektirovanie_vstraivaemyh_sistem(chast_2).htm (дата обращения: 14.01.2022).
13. Ключев А.О., Кустарев П.В., Ковязина Д.Р., Петров Е.В. Программное обеспечение встроенных вычислительных систем – Санкт-Петербург, 2009. URL: https://books.ifmo.ru/book/460/programmnoe_obespechenie_vstroennyh_vychislitelnyh_sistem.htm (дата обращения: 14.01.2022).

Список использованных источников

1. E. A. Lee, S. A. Seshia, Introduction to Embedded Systems - A Cyber-Physical Systems Approach, Second Edition. MIT Press, 2017.
2. A 21st Century Cyber-Physical Systems Education. National Academies of Sciences, Engineering, and Medicine. 2016. Washington, DC: The National Academies Press.
3. Платунов А.Е., Пинкевич В.Ю. Создание киберфизических систем: проблемы подготовки ИТ-специалистов // Control Engineering Россия - 2021. - № 3(93). - С. 64-70.
4. Платунов А.Е. Встраиваемые системы управления // Control Engineering Россия - 2013. - Т. 43. - № 1. - С. 16 - 24.
5. Hermann Kopetz. Real-Time Systems. Design Principles for Distributed Embedded Applications. Springer US, 2011.
6. Cyber-Physical Systems. URL: https://ptolemy.berkeley.edu/projects/cps/ (дата обращения: 01.05.2021).
7. Association for Computing Machinery. Curricula Recommendations. URL: https://www.acm.org/education/curricula-recommendations (дата обращения: 01.05.2021).
8. Computing Curricula 2020: Paradigms for Global Computing Education. Association for Computing Machinery, IEEE Computer Society. 2020 December 31.
9. Перекатов, В.И. Эталоны компьютерной науки в американском высшем образовании (1968-2001). Ч.1, 2 // Высшее образование сегодня: Реформы. Нововведения. Опыт = Higher education today. - М. - 2003. - №1 - С. 20-34 : ил.; 2003. - №2 - С.42-55 : ил.
10. P. Marwedel, T. Mitra, M. E. Grimheden, H. A. Andrade, “Survey on Education for Cyber-Physical Systems,” in IEEE Design & Test, vol. 37, no. 6, pp. 56-70, Dec. 2020, doi: 10.1109/MDAT.2020.3009613.
11. M. E. Grimheden, M. Törngren. Towards curriculum guidelines for Cyber-Physical Systems. Proceedings of the WESE'14: Workshop on Embedded and Cyber-Physical Systems Education. October 2014, Article No.: 7. p. 1–6.
12. Essence – Kernel and Language for Software Engineering. URL: https://www.omg.org/spec/Essence/ (дата обращения: 14.01.2022).
13. Левенчук А.И. Системное мышление. Учебник. — Изд-во «Издательские решения». — 2018. — 398 с.
14. Anatoly Levenchuk, Towards a Systems Engineering Essence – INCOSE Russian chapter product, 2015.
15. Model-Based Systems Engineering. Capella MBSE Tool. URL: https://www.eclipse.org/capella/ (дата обращения: 14.01.2022).

16. Introduction to Arcadia with Capella. URL: https://esd.sutd.edu.sg/40014-capella-tutorial/index.html (дата обращения: 14.01.2022).
17. Eclipse Capella - Open Source MBSE Solution. URL: https://www.youtube.com/c/EclipseCapella (дата обращения: 14.01.2022).
18. Ivar Jacobson. Essence kernel. Alpha State Cards. URL: https://www.ivarjacobson.com/publications/cards/alpha-state-cards-pdf-version (дата обращения: 14.01.2022).
19. Schaumont P.R. A Practical Introduction to Hardware/Software Codesign. 2010.
20. А.О. Ключев, В.Ю. Пинкевич, А.Е. Платунов, В.А. Ключев. Стенд-конструктор SDK-1.1M. Организация и программирование микроконтроллеров. – СПб: Университет ИТМО, 2022. – 79 с.

Горбачев Ярослав Георгиевич
Платунов Алексей Евгеньевич
Пинкевич Василий Юрьевич
Кольчурин Максим Вячеславович

Киберфизические системы. Методы высокоуровневого проектирования

Учебное пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49, литер А