Информационные технологии

© Чугреев В.Л.

ТЕХНИЧЕСКИЙ ДОЛГ В ПРОГРАММНЫХ ПРОЕКТАХ ИННОВАЦИОННОГО ТИПА

ЧУГРЕЕВ ВАЛЕРИЙ ЛЕОНИДОВИЧ
кандидат технических наук, старший научный сотрудник отдела проблем научно-технологического развития и экономики знаний
Федеральное государственное бюджетное учреждение науки
Институт социально-экономического развития территорий Российской академии наук
E-mail: chugreev10@mail.ru

В статье рассматривается вопрос целесообразности использования быстрых архитектурных решений в проектах инновационного типа. Основной тезис статьи в том, что осознанное использование концепции технического долга на ранних стадиях высокорискованных программных разработок позволяет получить экономию финансовых и трудовых ресурсов. Рефакторинг, т. е. улучшение и оптимизация программной архитектуры, является важной составляющей процесса разработки, но заниматься ею целесообразно после верификации бизнес-модели. На ранних этапах, предшествующих подтверждению бизнес-гипотез относительно рынка сбыта, глубокий рефакторинг может замедлить выпуск прототипа и увеличить бюджет проекта, еще до подтверждения его коммерческого потенциала. Данное утверждение справедливо с учетом некоторых оговорок. Участники процесса должны понимать концепцию технического долга и использовать его осмысленно. Даже на ранних этапах проекта, еще до верификации бизнес-модели можно существенно осложнить разработку, если совсем не уделять внимания внутреннему качеству кода. Если технический долг тормозит разработку проекта, если ставит под угрозу его запуск или привносит в систему нестабильность, то он неприемлем. Есть три области программной инженерии, в которых может накапливаться технический долг: на уровне модели предметной области, на уровне программной архитектуры, на уровне программного кода. Каждый из этих уровней несет в себе потенциальную возможность накопления технического долга, и каждую из них нужно контролировать. Наиболее трудоемок и затратен рефакторинг на первом и втором уровнях. Здесь в первую очередь нужно сокращать затраты (до подтверждения бизнес-модели), однако это не всегда возможно. Исключение составляют случаи поиска продукта/услуги, т. е. те случаи, когда проработка проекта, его уточнение происходит в процессе разработки.

Технический долг, программный проект, разработка программных систем, программная архитектура, рефакторинг, стартап.

Целью данного исследования является изучение оптимального соотношения между затратами на ранних этапах разработки программной системы и качеством ее архитектуры. Задача исследования заключается в том, чтобы идентифицировать области разработки, аккумулирующие технический долг, а также оценить его последствия до и после этапа верификации бизнес-модели.

В настоящее время исследования, относящиеся к тематике технического долга, постулируют необходимость его скорейшей «выплаты» – проведения рефакторинга [6]. При этом редко принимаются во внимание экономические аспекты таких действий, их влияние на бюджет проекта и сроки его реализации. Не отрицая важность рефакторинга, а также пользу от его своевременного проведения, заметим, что необходимо также учитывать текущий этап развития проекта и его коммерческие перспективы. Научная новизна работы заключается в формулировании особенностей проектов инновационного типа на ранних этапах их развития в контексте разработки программных систем, а также в учете этих особенностей применительно к рефакторингу.

Прежде чем начать обсуждение, необходимо определиться с используемой терминологией.

Технический долг – метафора, обозначающая плохую структурную организацию системы, а также неаккуратный, трудночитаемый код. Авторство термина приписывается Уорду Каннингему, который сформулировал это понятие в виде метафоры финансового долга. Так же, как при финансовом займе, разработчики программного обеспечения (ПО) могут получить дополнительные средства сейчас в виде ускорения разработки ПО за счет использования неоптимальных технических решений, но неизбежно будут расплачиваться за них потом в виде трудо- и времязатрат при модификации ПО, т. е. платить своего рода проценты по долгу, погасить который можно путем рефакторинга [7].

Рефакторинг – это реструктуризация и оптимизация программной архитектуры, а также уточнение и модификация кода без изменения результатов его работы. В процессе рефакторинга не затрагивается поведение программы, работа идет с ее внутренней структурой и содержанием.

Программные проекты подразумевают разработку программного обеспечения. Проекты инновационного типа в условиях рыночной экономики – это, как правило, стартапы. Стартап – это временная структура, которая занимается поиском воспроизводимой, прибыльной, рентабельной бизнес-модели [2]. Идея стартапа заключается в запуске пробного бизнес-проекта, который имеет по мнению основателей некоторые шансы стать успешным. Ключевой вопрос здесь: какие именно шансы? Ответить на него изначально невозможно. Нужно пробовать, запускать проект, продвигать продукт на рынке и по итогам некоторого разумного периода (который зависит от наличия финансовых средств и терпения инвесторов) принимать решение о целесообразности продолжения этого проекта.

Стартап – это не обязательно отдельная бизнес-структура, стартапы вполне могут запускать в рамках какой-то организации, которая готова инвестировать свои средства в потенциально перспективные, но рискованные технологии/продукты. Под это дело может быть организован отдел или рабочая группа, выделен бюджет и обозначены сроки. Но каким бы ни был стартап – это всегда риск, это возможность получить выигрыш, но не его гарантия. Изначально невозможно просчитать рынок и его реакцию. Перспективы стартапа, какими бы радужными они ни выглядели для основателей

или инвесторов, – это всегда предположения, и совпадут ли они с реальностью, покажет только жизнь. Здесь как нельзя точнее уместна поговорка «пока не попробуешь, не узнаешь».

Несколько примеров дорогостоящих провалов.

Cuil – поисковая система и одноименная компания, основанная выходцами из корпорации Google. Создана на средства венчурных фондов, общий объем инвестиций – $33 млн, в 2010 году прекратила существование [9].

Google Reader – RSS-агрегатор компании Google, проект был запущен в 2005 году и прекратил свое существование в 2013 году. Затраты на него неизвестны, но очевидно, что они немалые. И если даже для самой корпорации они не были болезненными, то в любом случае это репутационные издержки и разочарование тех пользователей, которые пользовались данными системами. Известно, что более 150000 пользователей Google Reader подписали петицию против закрытия этой системы [4]. Однако их мнение не перевесило чашу весов экономической целесообразности или, точнее, нецелесообразности продолжения проекта.

В итоге экономические показатели и их перспективы – главное мерило любого стартапа. Каким бы замечательным ни был проект, если его бизнес-модель нежизнеспособна, его закрывают. «Печальная для венчурных инвесторов статистика: 11 из каждых 12 стартапов проваливается, так и не окупив вложенных денег» [5]. И если для крупных компаний инвестиционные потери хоть и неприятны, но терпимы, то для средних и мелких – такие потери могут оказаться разорительными.

Конечно, инициаторы стартапа стараются просчитать проект, учесть его рыночный потенциал и т. д., но в самой природе инновационных проектов заложена неопределенность. Изначально это новинка, и нужна ли она людям – большой вопрос. Здесь нужно сделать уточнение. Продукт (или услуга) сам по себе может и не быть новинкой, но новым может быть рынок его распространения. Взять для примера социальную сеть ВКонтакте. Изначально воспринимаемая как клон Facebook она со временем приобрела индивидуальные черты и захватала лидирующие позиции в русскоязычном сегменте рынка социальных сетей.

Практически невозможно спрогнозировать успех или неудачу стартапа, здесь слишком много неизвестных. Однако есть методики минимизации потерь. Одной из таких методик является «Бережливый стартап» (англ. Lean Startup). Она была сформулирована Эриком Рисом и изложена в его книге «Бизнес с нуля. Метод Lean Startup для быстрого тестирования идей и выбора бизнес-модели» [3]. В основе этого подхода лежат такие концепции, как бережливое производство, развитие клиентов и гибкая методология разработки [1].

Смысл бережливого стартапа заключается в том, чтобы минимизировать затраты на запуск проекта и развитие/поддержание не востребованного клиентами функционала. Каждая из заложенных в проект возможностей, потенциально интересных пользователям (как правило, это то, что образует УТП – уникальное торговое предложение), максимально быстро проверяется на актуальность и востребованность (действительно ли это нужно пользователям?), и по результатам проверки принимается решение о дальнейшем сопровождении данной функциональной возможности.

В случае программных проектов речь идет о том, что может программа. Сюда относится и функционал, и интерфейсная часть. Если какие-то условные кнопки пользователи не нажимают или нажимают, но редко, значит, с ними что-то не

так. В парадигме бережливого стартапа нужно разобраться, нужны ли вообще эти кнопки? Кнопки – это очень упрощенное, если не сказать примитивное, обозначение в некоторых случаях сложнейшего комплекса интерфейсных элементов, за которыми стоят функциональные возможности программной системы. Если пользователи ими не пользуются, то какой смысл в их дальнейшем сопровождении? И наоборот – те интерфейсные элементы, а значит, и функциональные возможности, которые чаще используются, следует развивать и улучшать.

Изначально можно продумать возможности системы и реализовать их в том или ином объеме, необходимом для первичного запуска программного продукта, но какие из этих возможностей окажутся востребованными – покажет лишь время. С точки зрения концепции бережливого стартапа есть гипотезы, т. е. предположения о том, что именно нужно/интересно будущим потребителям продукта. После запуска проекта необходимо как можно быстрее проверить эти гипотезы. Причем проверка не сводится к одному лишь техническому мониторингу: какими интерфейсными элементами пользователи пользуются чаще, а какими реже? Проверке также подвергается готовность пользователей платить за эти возможности. Самая критичная и трезвая оценка для любого стартапа: сколько людей готовы платить за предоставляемый продукт?

Обычно это долевая оценка: отношение числа пользователей, попробовавших продукт (например, минимальную бесплатную или демонстрационную версию), к числу пользователей, оплативших полную версию. В любом случае – это реальные, живые деньги, заработанные на первых этапах стратапа. Понятно, что поначалу это совсем небольшие вырученные средства, не играющие существенной роли, но само отношение позволяет прогнозировать отдачу от проекта в перспективе, с учетом более серьезного рекламного бюджета и маркетинговых усилий. На начальном этапе инициаторы стартапа пытаются понять, «стоит ли игра свеч». Если проект покажет достаточно высокую конверсию осведомленных клиентов в реальные (купившие ПО), то он будет развиваться дальше. Если нет, то проект будет свернут, а вложенные в него средства спишут как потери, как плату за риск.

Возвращаясь к теме технического долга. Каким образом те или иные технические решения, принимаемые на стадии разработки программной системы, могут повлиять на стоимость проекта? Самым прямым: качество разрабатываемого программного продукта прямо пропорционально его стоимости. При этом нужно различать внешнее и внутреннее качество ПО.

Внешнее качество выражается в том, как выглядит программа и как она работает для пользователя (корректно или некорректно, быстро или медленно). Это все то, что пользователь может заметить и оценить. Внутреннее качество связано с архитектурными решениями, принятыми в процессе проектирования программной системы. Упрощенно говоря, это организация программного кода, которая подразумевает деление ответственности между различными кодовыми частями системы. Если речь идет об объектно-ориентированных языках, то ответственность делится между классами и их методами. К внутреннему качеству также относится читаемость кода – название классов, методов, переменных, единый стиль именования, аккуратное форматирование и др.

Конечный пользователь может оценить только внешнее качество, внутреннее ему не доступно. Если он и сталкивается с ним, то только опосредованно. Высокое внутреннее качество, как правило, выражается в стабильной, быстрой

и корректной работе программы. Низкое – наоборот. При этом зависимость не является гарантированной и точно прогнозируемой закономерностью. Это скорее корреляция. Бывает так, что эстетически приятная и аккуратная программа имеет неряшливую кодовую базу. С одной стороны – она работает и выглядит привлекательно, с другой стороны, дальнейшее сопровождение и доработка такой программы – крайне трудоемкий процесс, требующий много времени и сил. Важно не только то, как работает программа сейчас, но и то, как ее легко/просто дорабатывать, модифицировать.

Плохо организованный код подразумевает значительные трудности по его модификации. Любое внесение изменений в такой код чревато самыми непредсказуемыми последствиями. Неряшливая кодовая база, неграмотная архитектура, дублирование кода – все это технический долг, который существенно тормозит дальнейшее развитие проекта. Да, но в случае стартапа дальнейшее развитие проекта возможно только после прохождения первичного этапа, первичной проверки жизнеспособности заложенной в проект бизнес-модели.

В этом случае стремление к высокому внутреннему качеству на этапах, предшествующих верификации бизнес-модели, скорее всего, обернется увеличением сроков разработки и, как следствие, ее стоимости. Может сложиться парадоксальная ситуация – программист, ведомый самыми лучшими побуждениями, стремящийся к хорошо продуманному и организованному коду, увеличивает стоимость проекта, т. е. своими действиями наносит экономический ущерб. Понятно, что это справедливо только в том случае, если проект окажется провальным. Но вспомним ранее приведенную статистику – таковых большинство. Это изначально высокорискованные инвестиционные вложения.

Как бы хорошо ни была спроектирована программа, если она никому не нужна, если пользователи не готовы платить за нее, то все средства, затраченные на ее разработку, окажутся выброшенными на ветер. Задача стартапа – максимально быстрая проверка гипотез и поиск работающей бизнес-модели. Если можно существенно сэкономить на сроках разработки за счет технического долга, то это нужно делать. Как это может повлиять на дальнейшую разработку? Здесь есть как минимум два сценария развития событий.

Сценарий № 1. В процессе изучения пользователей, их опыта работы с системой появляется понимание того, что в действительности необходимо. И зачастую то, что необходимо, коренным образом отличается от того, что есть. Инициаторы стартапа, его руководители делают так называемый «разворот» – кардинальное изменение функциональных возможностей или существенное изменение бизнес-модели. Это предполагает значительное изменение программного кода, в некоторых случаях – даже полное его переписывание. Технический долг здесь просто обнуляется, потому что фактически система пишется заново или переписывается большая ее часть.

Сценарий № 2. Тот редкий случай, когда гипотезы подтвердились и бизнес-модель оказалась реалистичной. В этом случае инициаторы стартапа решают его развивать дальше. Здесь было бы разумно как можно раньше «выплатить» технический долг и в дальнейшем развивать систему, поддерживая высокое внутренне качество. Если долг слишком высок, возможно, стоит переписать систему заново. На данном этапе, имея точные проверенные факты о работоспособности бизнес-модели и рентабельности вложений, можно привлекать дополнительные заемные средства, которые можно пустить в том

числе и на повторную разработку. Условный инвестор готов оплатить повторную разработку, но только в том случае, если он уверен в перспективах проекта.

Таким образом, проблема накопленного технического долга на первоначальном этапе не является критичной. Конечно, это справедливо с учетом некоторых оговорок. Рассмотрим их.

1. Участники процесса должны понимать концепцию технического долга и использовать его осмысленно. Если руководство проекта осознает опасность технического долга, если планирует его как временное решение, то сможет выделить необходимые ресурсы для своевременного погашения или обнуления долга (в случае переписывания системы).

2. Даже на ранних этапах проекта, еще до верификации бизнес-модели можно существенно осложнить разработку, если совсем не уделять внимания внутреннему качеству кода. Технический долг допустим, но все хорошо в меру. И мера эта определяется возможностью быстрого и беспроблемного запуска проекта. Если технический долг тормозит разработку проекта, если ставит под угрозу его запуск или привносит в систему нестабильность, то он неприемлем.

3. Есть три области программной инженерии, в которых может накапливаться технический долг:

1) На уровне модели предметной области. Здесь речь идет о неоптимальном или некорректном отражении предметной области (сущностей и связей, из которых она состоит) на программную систему. Понятно, что если модель выбрана неудачно, то и ее программная реализация будет такой же. Эта проблема имеет две составляющие: недостаточное понимание предметной области разработчиком системы и некорректное отражение модели предметной области на программную реализацию.

Первая составляющая имеет своей причиной незнание моделируемых процессов и сущностей предметной области. Например, программисту нужно разработать систему учета банковских операций, в которой он не то что не является специалистом, а вообще не имеет о ней никакого представления. Конечно, разработчик так или иначе будет изучать предметную область (он вынужден это делать) и, вероятно, консультироваться со специалистами, но вряд ли это избавит его от ошибок. Подготовка грамотного специалиста может занимать куда больше времени, чем имеется в распоряжении у программиста в рамках проекта.

Вторая составляющая связана с воплощением модели в программном коде. Здесь также возможны искажения и неточности, причиной которых, как правило, являются технические ограничения и особенности используемых технологий. Сначала реальные объекты и процессы предметной области отражаются в виде иерархии классов и сценариев взаимодействия объектов (если используется объектно-ориентированный подход), затем этот конгломерат идей переносится на реляционную структуру базы данных, чтобы обеспечить хранение данных. На каждом из этих этапов могут быть свои сложности и возможности для накопления технического и/или концептуального долга. Под концептуальным долгом здесь понимаются ошибки и неточности, обусловленные недостаточно глубоким пониманием предметной области.

2) На уровне программной архитектуры. Это проблема неоптимального разделения ответственности между обособленными частями программного кода – обычно между классами. Конечно, программная архитектура включает в себя модель предметной области (МПО), выраженной в программном коде, но МПО – только часть архитектуры.

Если разработка программной системы идет, например, по паттерну MVC (Model-View-Controller – модель-представление-контроллер), то архитектура будет включать в себя классы контроллеров, вспомогательных моделей для передачи данных в представления, сами представления. Помимо этого в программной архитектуре наверняка будут присутствовать и другие вспомогательные классы, обеспечивающие инфраструктуру кэширования, доступа к базе данных и др.

3) На уровне кода. Мы уже упоминали об этом выше, это все, что относится к читаемости кода: названия классов, методов, переменных, единый стиль именования, аккуратное форматирование и др.

Каждый из этих уровней несет в себе потенциальную возможность накопления технического долга, и каждую из них нужно контролировать. Самый простой из них – это 3-й уровень. Здесь опытный, квалифицированный программист при наличии достаточной культуры кодирования может вообще не накапливать долг. Здесь можно сразу писать аккуратный и грамотный код, что, однако, затруднительно в случае 1-го и 2-го уровня. Какой бы высокой квалификации и культуры кодирования ни был программист, проработка предметной области и архитектуры – это длительный, итеративный процесс. Эрик Эванс [8] отмечает, что ранние модели предметной области – это, как правило, наивные модели. И лишь в процессе программной разработки, многочисленных архитектурных уточнений, а также консультаций со специалистами в предметной области модель приобретает зрелость.

Наиболее трудоемок и затратен рефакторинг на 1-ом и 2-ом уровнях. Его по возможности стоит сокращать и избегать (до подтверждения бизнес-модели), однако это не всегда возможно. Исключение составляют случаи поиска продукта/услуги. Бизнес-идея или идея продукта может быть изначально аморфной, не проработанной в полной мере. Как известно, детали могут играть решающую роль в успехе или провале проекта. В случае разработки программного проекта эти детали выражаются в проработке сценариев взаимодействия пользователя с системой. Эти сценарии не очевидны на начальном этапе разработки продукта/услуги, кроме того, они во многом зависят от технических особенностей используемых технологий. Бизнесмен, желающий реализовать ту или иную идею, не может, да и не должен знать все.

Программная часть – это епархия специалистов по разработке ПО, они, в свою очередь, могут не знать каких-то особенностей планируемых бизнес-процессов. Поэтому предварительное согласование проекта оставляет множество белых пятен, которые могут быть устранены только в процессе дальнейшей работы, в процессе взаимодействия представителей бизнеса и технических специалистов, не говоря уже о последующем тестировании продукта на реальных пользователях.

В том случае, когда задача четко формализована, а изменение/уточнение требований в процессе разработки сведено к минимуму, есть возможность сэкономить бюджет проекта за счет осознанного технического долга. Технический риск, связанный с недостаточно проработанной архитектурой на ранних этапах развития программного проекта, является вполне допустимым, при условии, что этот риск имеет разумные пределы и контролируется участниками процесса.

ЛИТЕРАТУРА

1. Бережливый стартап [Электронный ресурс] // Википедия. – Режим доступа : https://ru.wikipedia.org/wiki/Бережливый_стартап
2. Бланк, С. Стартап. Настольная книга основателя [Текст] / С. Бланк, Б. Дорф. – М. : Альпина Паблишер, 2014.
3. Рис, Э. Бизнес с нуля: Метод Lean Startup для быстрого тестирования идей и выбора бизнес-модели [Текст] / Э. Рис. – М. : Альпина Паблишер, 2012.
4. Сайт петиций [Электронный ресурс]. – Режим доступа : https://www.change.org/p/google-keep-google-reader-running
5. Старт-даун: 10 идей для стартапов, которые никогда не срабатывают [Электронный ресурс] // Компьютерра. – Режим доступа : http://www.computerra.ru/business/54600/start-daun-10-idey-dlya-startapov-kotoryie-nikogda-ne-srabatyivayut
6. Фаулер, М. Рефакторинг: улучшение существующего кода [Текст] / М. Фаулер. – СПб. : Символ Плюс, 2003.
7. Чугреев, В. Л. Развитие IT-методологии на основе интердисциплинарного подхода [Текст] / В. Л. Чугреев // Актуальные вопросы в научной работе и образовательной деятельности: сб. науч. трудов по материалам Международной научно-практической конф., г. Тамбов, 31 января 2013 года : в 13 ч. – Ч. 7 – Тамбов : Изд-во ТРОО «Бизнес-Наука-Общество», 2013. – С. 154–155.
8. Эванс, Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем [Текст] / Э. Эванс. – М. : Вильямс, 2011.
9. Cuil [Электронный ресурс] // Википедия. – Режим доступа : http://ru.wikipedia.org/wiki/Cuil

ИНФОРМАЦИЯ ОБ АВТОРЕ

Чугреев Валерий Леонидович – кандидат технических наук, старший научный сотрудник отдела проблем научно-технологического развития и экономики знаний. Федеральное государственное бюджетное учреждение науки Институт социально-экономического развития территорий Российской академии наук. Россия, 160014, г. Вологда, ул. Горького, д. 56а. E-mail: chugreev10@mail.ru. Тел.: (8172) 59-78-10.

Chugreev V.L.

TECHNICAL DEBT IN SOFTWARE PROJECTS OF INNOVATIVE TYPE

The article discusses the feasibility of using rapid architectural solutions in innovative projects. The main thesis of the article is that the conscious use of the technical debt concept at the early stages of high-risk software helps save financial and human resources. Refactoring, i.e. improvement and optimization of software architecture is an important component of the development process. However, it is better to verify the business model first. At the early stages, prior to the confirmation of the business hypotheses regarding the market, deep refactoring can slow down the prototype production and increase the project budget, even before confirmation of its commercial potential. Some limitations should be taken into consideration. The process participants should understand the concept of technical debt and use it intelligently. Even at the early stages of the project, prior to the business model verification, one can complicate the development significantly, if he/she neglects the internal quality of the code. If technical debt inhibits the project development, compromises its launch or makes the system unstable, it is unacceptable. There are three areas of software engineering, which can accumulate technical debt: at the level of the domain model, software architecture and software code. Each level can accumulate technical debt, thus they should be monitored. Refactoring is most time-consuming and costly at the first and second levels. One should cut costs (to confirm the business model) in the first place but this is not always

possible. The cases of searching for a product/service can be an exception, as the elaboration of the project and its specification occurs in the process of development.

Technical debt, software project, software system development, software architecture, refactoring, startup.

REFERENCES

1. Berezhlivyi startap [Lean Startup]. *Vikipediya* [Wikipedia]. Available at: https://ru.wikipedia.org/wiki/Berezhlivyi_startap
2. Blank S., Dorf B. *Startap. Nastol'naya kniga osnovatelya* [Handbook of the Founder]. Moscow : Al'pina Pablisher, 2014.
3. Ries E. *Biznes s nulya: Metod Lean Startup dlya bystrogo testirovaniya idei i vybora biznes-modeli* [The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses]. Moscow : Al'pina Pablisher, 2012.
4. *Sait petitsii* [Petitions Website]. Available at: https://www.change.org/p/google-keep-google-reader-running
5. Start-daun: 10 idei dlya startapov, kotorye nikogda ne srabatyvayut [Start-Down: 10 Ideas for Startups that Never Work]. *Komp'yuterra* [Computerra]. Available at: http://www.computerra.ru/business/54600/start-daun-10-idey-dlya-startapov-kotoryie-nikogda-ne-srabatyivayut
6. Fowler M. *Refaktoring: uluchshenie sushchestvuyushchego koda* [Refactoring: Improving the Design of Existing Code]. Saint Petersburg : Simvol Plyus, 2003.
7. Chugreev V. L. Razvitie IT-metodologii na osnove interdistsiplinarnogo podkhoda [Development of IT-Based Methodologies on the Basis of Interdisciplinary Approach]. *Aktual'nye voprosy v nauchnoi rabote i obrazovatel'noi deyatel'nosti: sb. nauch. trudov po materialam Mezhdunarodnoi nauchno-prakticheskoi konf., g. Tambov, 31 yanvarya 2013 goda : v 13 ch.* [Current Issues in Research and Educational Activity: Collection of Scientific Papers According to the Materials of the International Research-to-Practice in Tambov, January 31 2013: in 13 parts]. Tambov : Izd-vo TROO "Biznes-Nauka-Obshchestvo", 2013, pp. 154–155.
8. Evans E. *Predmetno-orientirovannoe proektirovanie (DDD): strukturizatsiya slozhnykh programmnykh sistem* [Domain-Driven Design: Tackling Complexity in the Heart of Software]. Moscow : Vil'yams, 2011.
9. Cuil. *Wikipedia*. Available at: http://ru.wikipedia.org/wiki/Cuil

INFORMATION ABOUT THE AUTHOR

Chugreev Valerii Leonidovich – Ph.D. in Technical Sciences, Senior Research Associate at the Department of Scientific and Technological Development and Knowledge Economy. Federal State Budgetary Institution of Science Institute of SocioEconomic Development of Territories of Russian Academy of Science. 56A, Gorky Street, Vologda, 160014, Russia. E-mail: chugreev10@mail.ru. Phone: +7(8172) 59-78-10.