СУПРОВІД ТА ЕВОЛЮЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

УДК 004.052.42

И.Б. Туркин, А.В. Сыромятников,
Ю.А. Кузнецова
Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ»

АНАЛИЗ ЭВОЛЮЦИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ОТКРЫТЫМ КОДОМ

Выполнен анализ проблем обеспечения качества проектов, специфичных для программной инженерии. Рассмотрены методы оценки внутреннего качества программного обеспечения на основе формальных показателей исходного кода, обосновано использование метрик исходного кода как отправной точки для оценки качества проекта. Выполнен практический анализ внутреннего качества ПО в динамике на основе ПО 7-Zip.

Виконано аналіз проблем забезпечення внутрішньої якості проектів, специфічних для програмної інженерії. Розглянуто методи оцінки внутрішньої якості програмного забезпечення на основі формальних показників вихідного коду, обґрунтовано використання метрик вихідного коду у якості відправної точки для оцінки якості проекту. Виконано практичний аналіз внутрішньої якості ПЗ на основі ПЗ 7-Zip.

The analysis of the software quality assurance problems is conducted. Internal software quality estimation methods based on formal source code measures are discussed, the use of source code metrics as a baseline for project quality estimation is grounded. Practical internal quality dynamics analysis is performed on"7-Zip" software.

Ключевые слова: *программный проект, программное обеспечение, качество, метрика, шкала, центральные моменты, версия, 7-Zip.*

Введение

Согласно результатам исследований компании Standish Group Chaos, из всех программных проектов, завершённых в 2010 году [1], только 32% программных проектов являются успешными, 44% – являются спорными (имеющими перерасход средств, превышение бюджета, неполную реализацию требований, другие недостатки), а 24% – являются провальными. Если сравнить с первыми результатами подобных исследований, проведенных ещё в 1994 году [2], наблюдается улучшение – на тот момент 16% проектов были успешными, 53% – спорными и 31% – неудачными. Однако текущая статистика все ещё оставляет много места для улучшения.

Успех любого проекта определяется его способностью удовлетворить потребности потребителя, а потому обеспечение высокого уровня качества является необходимой задачей любого производства, в том числе программной инженерии.

Недостаточное качество создаваемого ПО вынуждает многие IT-организации до 70 % бюджета информационной системы резервировать на этап сопровождения [3], при этом до 60% всех модификаций ПО выполняется для устранения ошибок, а только оставшиеся 40% – для коррекции ПО в рамках изменяющегося бизнес-процесса, совершенствования тех или иных показателей качества ПО, либо для предотвращения потенциальных проблем [4].

Качество ПО – понятие комплексное. Стандарты выделяют качество процессов разработки, внутреннее и внешнее качество программного продукта, качество программного продукта на стадии использования [5]. Для каждого из компонентов качества можно назвать набор метрик, определяющих качество программного продукта. Полученная структура называется моделью качества программного обеспечения [6, 7]. Метрика программного обеспечения – это мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций, а также метод её подсчета. Метрики позволяют получить численные значения каждого свойства программного обеспечения или его спецификаций.

Особый интерес представляют метрики сложности программного обеспечения.

Сложность является важным фактором, от которого зависят прочие параметры качества ПО, такие как точность, корректность, надёжность, удобство сопровождения. Существование методов и алгоритмов автоматического расчёта метрик сложности ПО с помощью программных средств позволяет получить комплексный формальный отчёт о качестве ПО за очень малое время. Это позволяет производить объективный мониторинг уровня качества ПО в течение всего жизненного цикла проекта, вносить коррективы в план проекта, а также своевременно принимать решение о необходимости проведения рефакторинга [8].

Анализ проблемы обеспечения качества в инженерии программного обеспечения

Обзор исследований и публикаций в области инженерии ПО за 2008–2012 гг., представленный в [9], позволяет утверждать, что практически всё четырехкратное увеличение общего количества публикаций за анализируемый период объясняется концентрацией научных интересов в сфере процессов разработки и управления ПО, включая модели жизненного цикла, анализ требований, тестирование, гибкие технологии разработки, зрелость процессов разработки, качество и измерение качества.

Проблемы авторского права в ПИ. Первоначально создание программного обеспечения для компьютеров было, в первую очередь, академическим занятием. Для специалистов в области компьютерной науки каждая программа представляла собой результат научного исследования. Это означает, что исходный текст программы был обязательно доступен всему научному сообществу, поскольку любой научный результат должен быть верифицируемым, то есть подтверждаться другими исследователями и быть открытым для критики [10].

В связи с бурным развитием информационных технологий программное обеспечение стало объектом продажи, поэтому на него автоматически распространяются уже не только законы научной разработки, но и свойства материальных предметов, которыми можно торговать, обмениваться, право владения и пользования которыми стоит охранять законодательно. Так программное обеспечение попало в разряд интеллектуальной собственности: сейчас исходный текст программы рассматривается как произведение, объект применения авторского права. Выбор программного обеспечения сейчас огромен, но большинство программных решений являются проприетарными[1], что ощутимо сказывается на возможностях его применения.

Основной характеристикой проприетарных лицензий является то, что издатель ПО в лицензии даёт разрешение её получателю использовать одну или несколько копий программы, но при этом сам остаётся правообладателем всех этих копий [11]. Одно из следствий такого подхода заключается в том, что практически все права на ПО остаются за издателем, а пользователю передаётся лишь очень ограниченный набор строго очерченных прав. Для проприетарных лицензий типично перечисление большого количества условий, запрещающих определённые варианты использования ПО, даже тех, которые без этого запрета были бы разрешены законом об авторском праве. Так, например, лицензия на Microsoft Windows включает большой список запрещённых вариантов использования, таких как обратная разработка, одновременная работа с системой нескольких пользователей и распространение тестов её рабочих характеристик.

Наиболее значительным следствием применения проприетарной лицензии является то, что конечный пользователь обязан принять её, так как по закону владельцем ПО является не он, а издатель программы. В случае отказа принять лицензию пользователь вообще не имеет права работать с программой.

Более всего ограничения проприетарного ПО сказываются на разработчиках программного обеспечения, так как проприетарные лицензии не только не предоставляют доступа к исходному коду ПО, но и запрещают повторно использовать в других продуктах даже готовые модули и библиотеки.

Такое положение вещей доставляет слишком много неудобств IT-сообществу, а потому у него существует альтернатива – свободное программное обеспечение.

Свободное программное обеспечение (СПО) – широкий спектр программных решений, в которых права пользователя («свободы») на неограниченные установку, запуск, а также свободное использование, изучение, распространение и изменение (совершенствование) программ защищены юридически авторскими правами при помощи

1. Английское proprietary значит «собственнический; частный», от латинского proprius – «владение, собственность». Не все программы с открытым кодом, распространяемые бесплатно – не проприетарны. Проприетарность говорит прежде всего о том, что разработка и производство ПО целиком принадлежат и контролируются неким собственником.

свободных лицензий [12]. На практике, для СПО исходные коды программного обеспечения должны быть доступны пользователям, наряду с исполняемыми файлами и лицензиями.

Движение СПО зародилось в 1983 году, когда Ричард Столлман сформулировал идею о необходимости дать программную свободу пользователям. В 1985 году Столлман основал Фонд свободного программного обеспечения, чтобы обеспечить организационную структуру для продвижения своей идеи.

Идея продиктована стремлением сохранить первоначальную модель научного сотрудничества между разработчиками ПО, для чего необходимо обеспечить доступность исходных текстов программ, написанных разработчиками, для чтения и критики всему научному сообществу с сохранением авторства произведений. Для этого Ричард Столлман сформулировал понятие свободное программное обеспечения, в котором отразились принципы открытой разработки программ в научном сообществе, сложившемся в американских университетах в 1970-е годы. Согласно этим принципам программу можно свободно:

– **использовать** с любой целью («нулевая свобода»);

– **изучать**, как программа работает, и адаптировать её для своих целей («первая свобода»);

– **распространять** копии программы («вторая свобода»);

– **улучшать** и **публиковать** свою улучшенную версию – («третья свобода»).

Возможность исправления ошибок и улучшения программ – самая важная особенность свободного и открытого программного обеспечения, что просто невозможно для пользователей закрытых частных программ.

Понятию свободного программного обеспечения (*free software*) близко понятие открытого программного обеспечения (*open source software*).

Открытое программное обеспечение (англ. open source software) – программное обеспечение с открытым исходным кодом [13]. Исходный код таких программ доступен для **просмотра**, **изучения** и **изменения**, что позволяет пользователю принять участие в доработке самой программы, использовать код для создания новых программ и исправления в них ошибок – через заимствование исходного кода, если это позволяет лицензия, или через изучение использованных алгоритмов, структур данных, технологий, методик и интерфейсов.

Термин *open source* был создан в 1998 году Эриком Реймондом и Брюсом Перенсом, которые считали, что термин *free software* (*свободное ПО*) в английском языке неоднозначен и смущает многих коммерческих предпринимателей.

Подавляющее большинство открытых программ является одновременно свободными. Отличие между движениями открытого ПО и свободного ПО заключается в основном в приоритетах. Сторонники термина «*open source*» делают упор на эффективность открытых исходников как метода разработки, модернизации и сопровождения программ. Сторонники термина «*free software*» считают, что именно права на свободное распространение, модификацию и изучение программ являются главным достоинством свободного открытого ПО.

Открытое ПО удобно для использования в образовательных и исследовательских целях, так как позволяет изучать подходы и методы, технику и стиль написания кода другими программистами, а также исследовать развитие и изменение программного решения на протяжении всего его времени разработки.

Качество открытого программного обеспечения. Широко распространенным является мнение о том, что качество свободного ПО уступает качеству несвободного, – как минимум потому, что большинство проприетарных программ являются платными, и у разработчиков есть более сильная мотивация для написания качественного ПО.

Исходный код несвободного ПО закрыт, поэтому нельзя объективно оценить его качество. Остаётся только возможность сравнивать качество исходного кода открытого ПО с другим открытым ПО или другими версиями этого же продукта. Сравнение версий одного и того же продукта даёт возможность объективно оценить его качество на протяжении каждого этапа разработки и проследить его эволюцию [14].

Качество программного обеспечения – это степень соответствия его предъявленным требованиям. Качество является важнейшей характеристикой ПО, а потому обеспечение качества предполагает активности по обеспечению качества всех компонентов на всех этапах жизненного цикла разработки ПО [15]. Каждый этап жизненного цикла ПО

характеризуется разным набором активностей, а потому к каждому может применим свой набор характеристик качества (табл. 1).

Так как речь идёт об изучении характеристик качества уже разработанного ПО, не имеет смысла рассматривать качество процессов разработки – ПО разрабатывается третьими лицами, поэтому отсутствует возможность оценить качество процессов. Однако доступность нескольких версий исходного кода предоставляет широкие возможности для изучения эволюции проекта и оценки параметров его качества в динамике.

Таблица

Применение метрик ПО на этапах жизненного цикла ПО

Этап жизненного цикла	Составляющие качества
Формирование требований, планирование проекта	Трудоемкость проекта, Риск проекта
Проектирование	Сложность Мобильность
Реализация	Сложности Корректность Надежность Производительность
Тестирование	Корректность Ресурсоемкость Отказоустойчивость
Внедрение	Удобство использования Ресурсоемкость Отказоустойчивость
Эксплуатация и сопровождение	Сложность Сопровождаемость

Метрика программного обеспечения [16] – это мера, позволяющая получить численное значение некоторого свойства программного обеспечения или его спецификаций.

Среди всех метрик стадии разработки ПО особый интерес представляют метрики сложности ПО. Во-первых, они лучше прочих поддаются формальной оценке. А во-вторых, именно сложность является основополагающим фактором качества ПО, так как, чем ниже сложность, тем проще программисту ориентироваться в программе и писать новый код, ниже вероятность занесения ошибки и выше шанс обнаружить уже существующую.

Несмотря на преимущества метрического подхода к оценке качества ПО, формальные метрики сами по себе не могут быть использованы в качестве основы для вынесения решения о качестве продукта [17]. Существует множество параметров, которые не могут быть оценены формально, например, производительность труда программиста или совершенство архитектуры ПО. С другой стороны, метрики могут быть использованы в качестве хорошего дополнения к *прочим* методам оценки качества ПО. Наиболее эффективным способом их применения может являться рассмотрение изменения показателей метрик в ходе эволюции программного проекта, так как это позволяет наблюдать за динамикой изменения качества и своевременно реагировать на его изменение [18].

Из доступных публикаций можно извлечь много примеров практического применения метрик. Улучшение документированности программного проекта позволило снизить стоимость сопровождения с 50% до 38% от общей стоимости [19]. Управление сложностью привело к получению приемлемых показателей прочих характеристик качества ПО, таких как корректность, надёжность, производительность [20].

Метрики необходимы для упорядочения программных продуктов по выбранным свойствам, которые они характеризуют [21, 22]. В зависимости от особенностей рассматриваемых показателей качества применяются различные виды метрик и шкал для их измерения [23, 24].

Относительная шкала (ratio scale) – шкала, в которой есть определённая точка отсчёта и возможны отношения между значениями шкалы. Особенностью этой шкалы является наличие твёрдо фиксированного нуля, который означает полное отсутствие какого-либо свойства или признака. Шкала отношений является наиболее информативной шкалой, допускающей любые математические операции и использование разнообразных статистических методов. В *шкале отношений* измеряются такие физические показатели ПО, как:

- время выполнения программы;
- число маршрутов в программе;
- число таблиц в базе данных;
- объём программы и т.д.

Близкая к шкале отношений *интервальная шкала*, отличается от неё отсутствием нуля,

поэтому в интервальной может быть вычислена только разность, но не отношение.

Порядковая шкала позволяет ранжировать некоторые характеристики путём сравнения с опорными значениями. Для объекта измерения устанавливается приоритетность признаков. Различают абсолютные и относительные порядковые метрики, первые из которых показывают больше или меньше значение данного параметра программы по сравнению с опорным, а второй – во сколько раз больше или меньше. Математические преобразования с такими показателями более ограничены, чем у первого вида метрик.

Номинальная или категорийная шкала характеризует только наличие рассматриваемого свойства или признака у программы без учёта градации по численным значениям. Например: наличие у программы структурированности, гибкости, простоты освоения и т.д.

Изучение качества ПП на стадии разработки предполагает работы со следующими метриками:

- сложность;
- корректность;
- удобство использования;
- надежность;
- производительность;
- мобильность.

Из этих метрик именно сложность лучше всего поддаётся формальной оценке. Более того, управление сложностью является ключом к получению приемлемых значений прочих метрик, таких как корректность, надёжность, производительность. Чем грамотнее спроектировано ПО, тем ниже его сложность; чем ниже сложность, тем проще программисту ориентироваться в программе и писать новый код, ниже вероятность занесения ошибки и выше шанс обнаружить существующую. Снижение сложности ПО заметно снижает время его разработки и стоимость сопровождения. Опираясь на эти факты, Стив Макконелл назвал управление сложностью «главным техническим императивом разработки ПО» [25].

Метрики сложности программ принято разделять на три основные группы [26, 27]:

- метрики размера программ;
- метрики сложности потока управления программ;
- метрики сложности потока данных программ.

Метрики первой группы базируются на определении количественных характеристик, связанных с размером программы, и отличаются относительной простотой. К наиболее известным метрикам данной группы относятся число операторов программы, количество строк исходного текста, набор метрик Холстеда. Метрики этой группы ориентированы на анализ исходного текста программ. Поэтому они могут использоваться для оценки сложности промежуточных продуктов разработки.

Метрики второй группы базируются на анализе управляющего графа программы. Представителем данной группы является метрика Маккейба (1976 г. [28]) – показатель цикломатической сложности, который относится к группе показателей оценки сложности потока управления программой и вычисляется на основе графа управляющей логики программы (control flow graph).

Управляющий граф программы, который использует метрики данной группы, может быть построен на основе анализа алгоритмов модулей. Поэтому метрики второй группы могут применяться для оценки сложности промежуточных продуктов разработки.

Метрики третьей группы базируются на оценке использования, конфигурации и размещения данных в программе. В первую очередь это касается глобальных переменных. К данной группе относятся метрики Чепина.

Цель и задачи работы

Цель настоящей работы – совершенствование процессов разработки, сопровождения и реинжиниринга ПО на основе мониторинга количественных метрик – показателей внутреннего качества.

Для достижения цели необходимо решение следующих **задач**:

- определить набор рассчитываемых метрик сложности ПО;
- выполнить анализ полученных данных;
- сформулировать выводы об эволюции проекта с точки зрения качества ПО.

Результаты исследований

Объект исследований. 7-Zip – файловый архиватор с высокой степенью сжатия данных [29]. Поддерживает большое число форматов файлов, в том числе собственный формат 7z, который использует высокоэффективный алгоритм сжатия данных LZMA.

Разработка ведётся с 1999 года по сей день. 7-Zip является бесплатным ПО с открытым исходным кодом, большая часть которого распространяется на условиях свободной лицензии GNU LGPL. Исключением является код декомпрессора unRAR, который имеет ограничения.

Основной платформой является Windows, для которой существует две версии 7-Zip – консольное приложение и приложение с графическим интерфейсом пользователя. Кроме этого, консольная версия сообществом разработчиков была портирована для систем стандарта POSIX.

Языками разработки 7-Zip являются C++ и C. Исходный код 7-Zip начиная с версии 3.13 доступен для загрузки с хостинга проектов с открытым исходным кодом SourceForge.net [30]. По состоянию на ноябрь 2012 число загрузок 7-Zip превышает 281 миллион.

В ходе исследования будут рассмотрены 5 версий 7-Zip: 4.20, 4.40, 4.60, 9.04, 9.20. Скачок в нумерации версий объясняется сменой концепции именования.

Список изменений для каждой версии доступен на официальном сайте проекта [31]. Ключевые изменения между рассматриваемыми версиями таковы:

– 4.20: Основа для сравнения.

– 4.40: Добавлена поддержка форматов LZH, CHM/HXS, а также многотомных CAB-архивов. Добавлена поддержка больших страниц памяти.

– 4.60: Расширена поддержка многопоточности для отдельных алгоритмов архивации. В многопроцессорных системах теперь по умолчанию используется многопоточный режим работы. Улучшено шифрование .7z архивов. Уменьшено количество dll файлов.

– 9.04: Добавлена возможность изменения цельных .7z архивов. Реализован алгоритм сжатия LZMA2. Реализована поддержка формата XZ. Расширен список поддерживаемых файловых систем: NTFS, FAT, VHD и MBR.

– 9.20: Добавлена поддержка формата MSLZ. Реализована работа с большими (>8Гб) файлами в TAR архивах.

Помимо этого, в каждой новой версии были дополнительно оптимизированы алгоритмы и исправлены обнаруженные ошибки.

В 7-Zip исходный код каждого поддерживаемого файла вынесен в отдельный модуль и представлен отдельным проектом. В ходе данной работы в качестве объекта исследования выбрано ядро архивации собственного формата архиватора – 7z.

Инструментарий. CppDepend – программное обеспечения для статического анализа исходного кода приложений, написанных на языке C++ [32]. CppDepend является одним из продуктов семейства NDepend, в которое входят также пакеты NDepend для анализа .NET-кода и JArchitect для решений на языке Java. CppDepend является проприетарным программным обеспечением. Его разработку ведет компания SMACCHIA.COM S.A.R.L.

Первая версия CppDepend была выпущена в сентябре 2009 г. Она по умолчанию поддерживала работу с проектами формата .vcproj, а также поставлялась с отдельным инструментом ProjectMaker для поддержки других типов проектов. Инструмент позволял анализировать код, выполнял подсчёт метрик исходного кода, а также уже имел язык Code Query Linq (CQLinq) для гибкого создания запросов по метрикам исходного кода. Кроме этого, были реализована функции построения графа и матрицы зависимостей. По результатам анализа создавался HTML-отчёт.

Вторая версия вышла в апреле 2010 и характеризовалась расширением существующего функционала, оптимизацией используемых алгоритмов и большим количеством исправлений. Была добавлена возможность интеграции с Microsoft Visual Studio, что позволило выполнять анализ исходного кода непосредственно из интегрированной среды разработки. Значительно возросло быстродействие продукта (в 3 раза повышена скорость обработки CQL-запросов, на порядок возросла скорость вычислений для матрицы зависимостей). Улучшены функциональность и интерфейс работы с редактором CQL-запросов и графа зависимостей.

Третья (текущая) версия CppDepend была выпущена в июле 2012. Используется новый парсер C++ кода, обновлен язык запросов CQL. Кроме того, выпущена версия для операционных систем на базе Linux.

При написании инструмента использованы парсеры C++ doxygen и Clang. Ядро системы, язык запросов к исходному коду и интерфейс пользователя написаны на базе технологий .NET. CppDepend поддерживает операционные системы Windows и, начиная с версии 3, Linux.

CppDepend – многофункциональный инструмент для анализа базы исходных кодов проекта. Он выполняет статический анализ исходного кода и извлекает из него информацию, позволяющую эффективно оценить архитектуру системы, качество исходного кода, оперативно находить и устранять проблемные области. Данный инструмент обладает высокой гибкостью и позволяет пользователю самостоятельно задавать правила оценки каждого отдельного проекта.

Ключевыми возможностями CppDepend являются:

– средства оценки связности компонентов системы. В CppDepend реализована функция построения графа зависимостей между компонентами системы на уровне проектов, пространств имён, классов, методов. Эта информация может быть представлена в виде графа и матрицы зависимостей. Данная функциональность позволяет сравнивать фактическое состояние системы с запланированной архитектурой и вносить изменения в систему при регистрации отклонений;

– подсчёт ряда метрик исходного кода. Для каждого элемента системы, будь то метод, класс, пространство имён или решение целиком, подсчитываются метрики, которые могут быть использованы инструментом для дальнейшего

анализа системы, либо экспортированы для использования с внешними средствами;

– гибкий язык запросов CQL позволяет извлекать информацию непосредственно из исходного кода. Также с помощью CQL можно задать правила для определения проблемных участков системы (например, находить классы со слабой сопряженностью или методы со слишком высокой сложностью).

Обработка результатов измерений. CppDepend поддерживает вычисление 60 метрик исходного кода [33]. Ниже приведены метрики, представляющие интерес для настоящего исследования.

Предварительная обработка данных. В качестве основных статистических характеристик распределения метрик для классов в составе сборки использованы центральные моменты:

– математическое ожидание – среднее значение случайной величины: $x = \frac{1}{n}\sum_{i=1}^{n} x_i$;

– дисперсия – показатель рассеивания значений случайной величины относительно её математического ожидания.

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 ;$$

– показатель асимметричности, характеризует случаи, когда какие-нибудь причины благоприятствуют более частому появлению значений, которые выше или, наоборот, ниже среднего. При левосторонней, или положительной, асимметрии в распределении чаще встречаются более низкие значения признака, а при правосторонней, или отрицательной – более высокие:

$$A = \frac{\sum_i (x_i - \bar{x})^3}{n \cdot \sigma^3} .$$

– показатель эксцесса (*E*) характеризует меру остроты пика распределения случайной величины и определяется по формуле:

$$E = \frac{\sum (x_i - \bar{x})^4}{n \cdot \sigma^4} - 3 .$$

– минимальное значение метрики;

– максимальное значение метрики;

Следующей важной характеристикой распределения метрик классов являются **гистограммы**, которые позволяют оценить характер законов распределения. Это необходимо для выбора корректных методов их дальнейшей обработки, т.к. случайные величины с различной функцией распределения требуют различных подходов к их анализу. Из рисунков ниже видно, что лучшей аппроксимацией законов распределения метрик проекта является гамма-распределение.

Количество строк кода (Lines of Code, LOC) – общее число строк кода в данном классе. Абстрактные методы, перечисления имеют LOC = 0. При расчёте LOC учитываются только конкретные исполняемые строки кода. Объявления пространств имён, типов, методов, полей не учитываются как строки кода, т.к. не являются исполняемыми командами. Как видно из рис. 1, около 90% всех типов проекта содержат не более 100 строк кода.

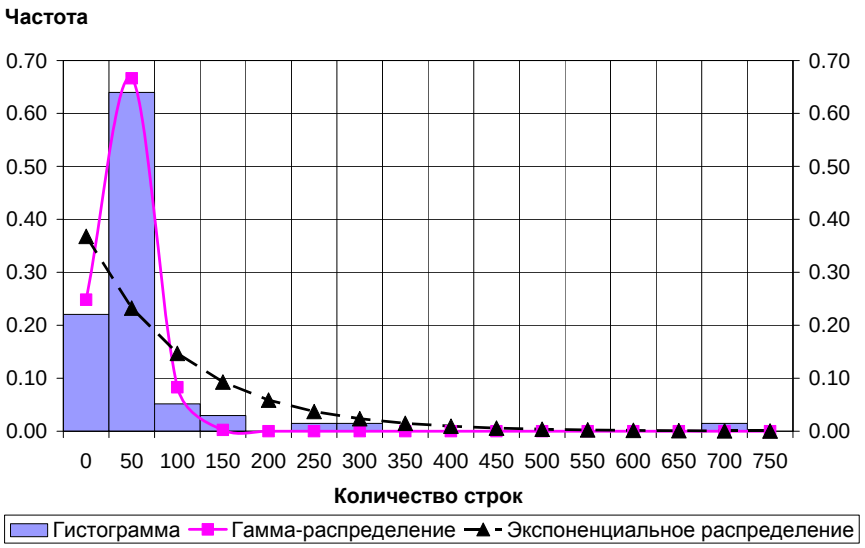

Рис. 1 Гистограмма количества строк кода в классе (версия 7-Zip 4.20 Core)

Общее количество строк кода (рис. 2) и количество классов (рис. 3) после перехода к версии 7-Zip 4.40 Core изменяется незначительно, что очередной раз подтверждает зрелость проекта.

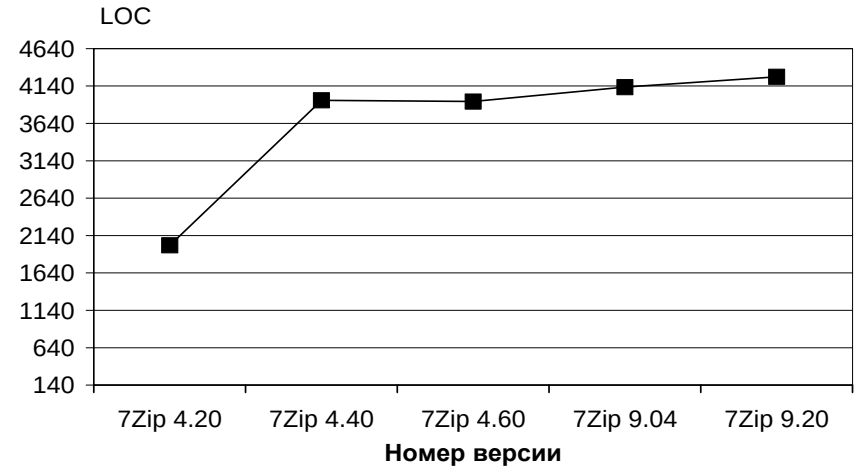

Рис. 2 Зависимость строк кода (LOC) от номера версии

Количество строк комментариев (Lines of Comment) и *процент комментариев (Percentage Comment).*

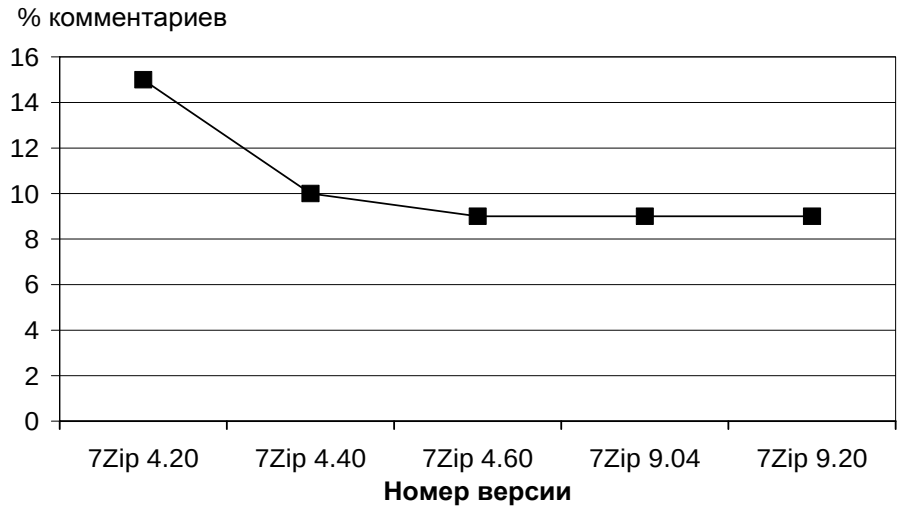

Рис. 3 Зависимость процента комментариев от номера версии

Считается, что код, содержащий менее 20% комментариев, должен быть документирован лучше. С другой стороны, чрезмерное количество комментариев (>40%) также может затруднить чтение кода.

На снижение процента комментариев в более поздних версиях 7-Zip 4.20 Core влияет, по-видимому, хорошая отработанность проекта.

Количество типов в проекте (Number of Type, NbTypes). Типами являются абстрактные и конкретные классы, структуры, перечисления.

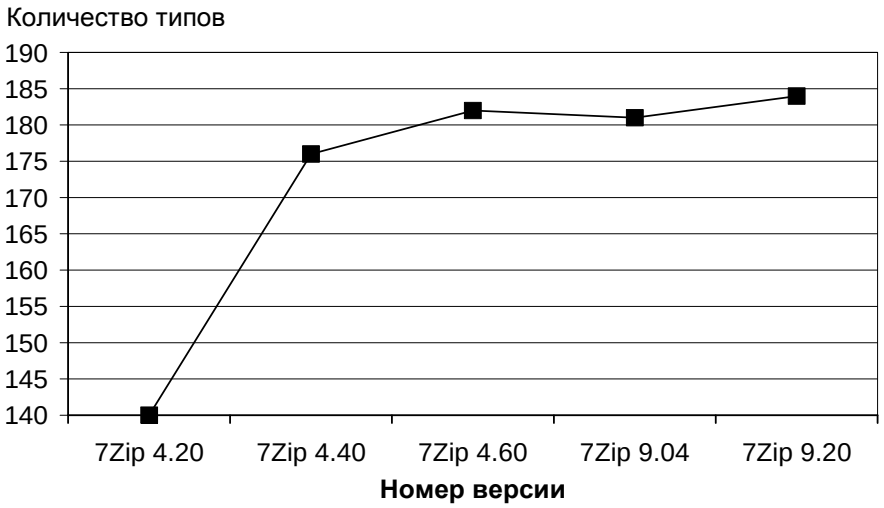

Рис. 4 Зависимость количества типов от номера версии

Входящая связанность (Afferent coupling, Ca). Количество типов вне данного пространства имен, зависящих от типов данного пространства имен. Высокая входящая связность указывает на то, что данное пространство имен имеет множество обязанностей.

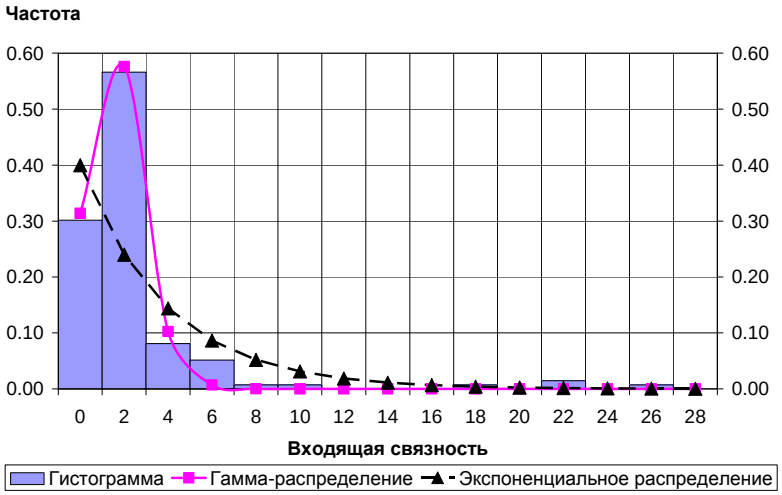

Рис. 5 Гистограмма входящей связанности (версия 7-Zip 4.20 Core)

Исходящая связанность (Efferent coupling, Ce). Количество типов данного пространства имён, зависящих от типов вне данного пространства имён. Высокая исходящая связность указывает, что данное пространство имён сильно зависимо от других (рис. 6).

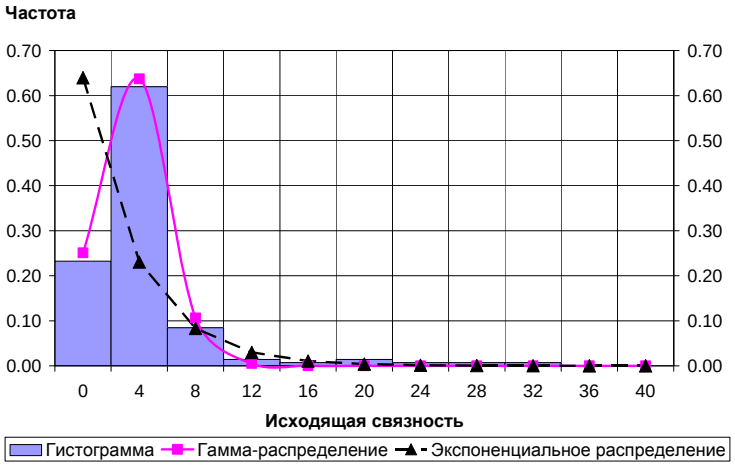

Рис. 6 Гистограмма исходящей связанности (версия 7-Zip 4.20 Core)

Цикломатическая сложность (Cyclomatic Complexity, CC). Цикломатическая сложность – структурная мера сложности, которая оценивает число возможных путей в потоке управления программы. При вычислении цикломатической сложности используется граф потока управления программы: узлы графа соответствуют неделимым группам команд программы и ориентированным рёбрам, каждый из которых соединяет два узла и соответствует двум командам, вторая из которых может быть выполнена сразу после первой. Цикломатическая сложность может также быть применена для отдельных функций, модулей, методов или классов в пределах программы.

Гистограмма цикломатической сложности классов отличается от рассмотренных выше тем, что достаточно хорошее приближение достигается при использовании однопараметрического экспоненциального распределение, что объясняется большим количеством абстрактных классов в проекте.

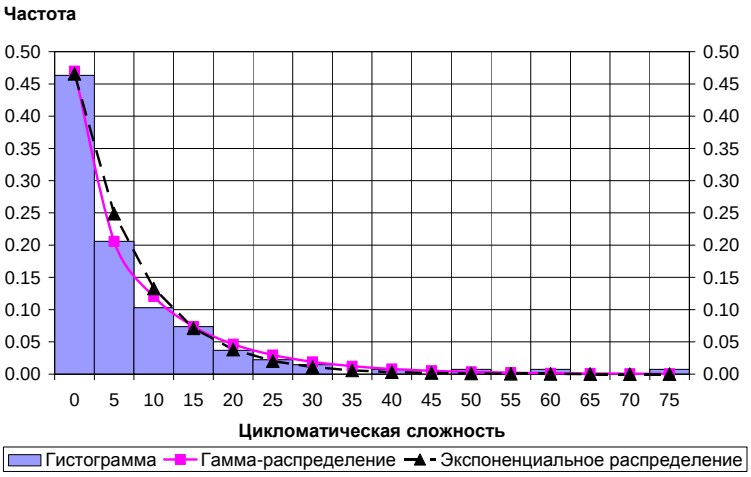

Рис. 7 Гистограмма цикломатической сложности классов (версия 7-Zip 4.20 Core)

В основе метрики *TypeRank*, характеризующей важность класса в проекте, находится алгоритм PageRank – один из алгоритмов ссылочного ранжирования. Алгоритм применяется к коллекции документов, связанных гиперссылками (таких, как веб-страницы из всемирной паутины), и назначает каждому из них некоторое численное значение, измеряющее его «важность» или «авторитетность» среди остальных документов. По своей сути алгоритм может применяться не только к веб-страницам, но и к любому набору объектов, связанных между собой взаимными ссылками, то есть к любому графу. Название «PageRank» является торговой маркой компании Google Inc. Алгоритм запатентован в США в 2001 году. Как следует из рисунка, 85% классов имеют примерно равную важность в проекте, что в целом свидетельствует о хорошо продуманной его архитектуре.

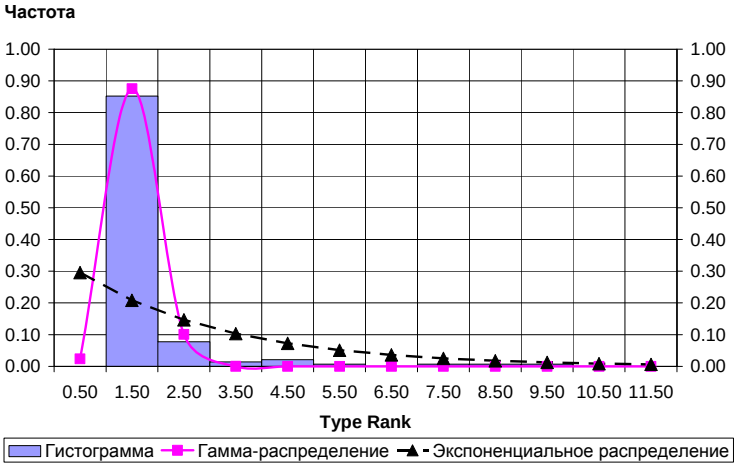

Рис. 8 Гистограмма *TypeRank* (версия 7-Zip 4.20 Core)

Определение **корреляции** – статистической взаимосвязи двух или нескольких случайных величин (либо величин, которые можно с некоторой допустимой степенью точности считать таковыми) [34]. При этом изменения значений одной или нескольких из этих величин сопутствуют систематическому изменению значений другой или других величин. Математической мерой корреляции двух случайных величин служит коэффициент корреляции.

Метод вычисления коэффициента корреляции зависит от вида шкалы, к которой относятся переменные. Так, для измерения переменных с интервальной и количественной шкалами необходимо использовать коэффициент корреляции Пирсона. Если, по меньшей мере, одна из двух переменных имеет порядковую шкалу, либо не является нормально распределённой, необходимо использовать ранговую корреляцию Спирмена.

Коэффициент корреляции рангов Спирмена [35] относится к непараметрическим показателям связи между переменными, измеренными в ранговой шкале. При расчёте этого коэффициента не требуется никаких предположений о характере распределений признаков в генеральной совокупности. Этот коэффициент определяет степень тесноты связи порядковых признаков, которые в этом случае представляют собой ранги сравниваемых величин.

Величина коэффициента корреляции Спирмена также лежит в интервале [-1; 1]. Он, как и коэффициент Пирсона, может быть положительным и отрицательным, характеризуя направленность связи между двумя признаками, измеренными в ранговой шкале.

Общая формула для вычисления коэффициента корреляции рангов Спирмена:

$$P = 1 - \frac{6 \times \sum(D^2)}{n \times \left(n^2 - 1\right)},$$

где n – количество ранжируемых признаков (пар значений);

D – разность между рангами по двум переменным для каждой пары значений случайных величин;

$\sum(D^2)$ – сумма квадратов разностей рангов.

Результаты анализа ранговой корреляции по Спирмену дают основания говорить о наличии сильных положительных зависимостей не только между всеми перечисленными метрика, но и нижеперечисленными:

– недостаточная сопряженность методов (Lack of Cohesion Of Methods, LCOM). Принцип одиночной ответственности предполагает, что для изменения класса не должно существовать более одной причины. Такой класс называется сопряженным. Высокое значение LCOM обычно говорит о недостаточном уровне сопряженности класса;

– ассоциация между классами (Association Between Class, ABC). Метрика Ассоциация между классами для отдельного класса представляет число членов других типов, которые он напрямую использует из тела своих методов;

– глубина в дереве наследования (Depth in Inheritance Tree, DIT). Указывает на длину цепочки классов-родителей данного класса. Классы с DIT > 6 могут быть сложны для сопровождения. С другой стороны, это не строгое правило, т.к. иногда классы должны быть унаследованы от классов другого слоя, имеющих высокий показатель DIT. Например, средняя глубина в дереве наследования у классов платформы .NET, которые расширяют System.Windows.Forms.Control составляет 5.3.

В заключение несколько общих выводов об архитектуре проекта, полученных на основе анализа сборки в целом.

Относительная когезия $H = \left(R + 1\right)/N$, характеризующая внутреннюю связность классов в проекте (рис. 9) находится в рекомендуемом диапазоне $1.5 \le H \le 4.0$, где:

– R – количество внутренних связей;

– N – количество классов в сборке.

Для классов проекта характерна высокая исходящая связность, что является признаком нестабильность из-за сильной зависимости от классов вне своего пространства имен.

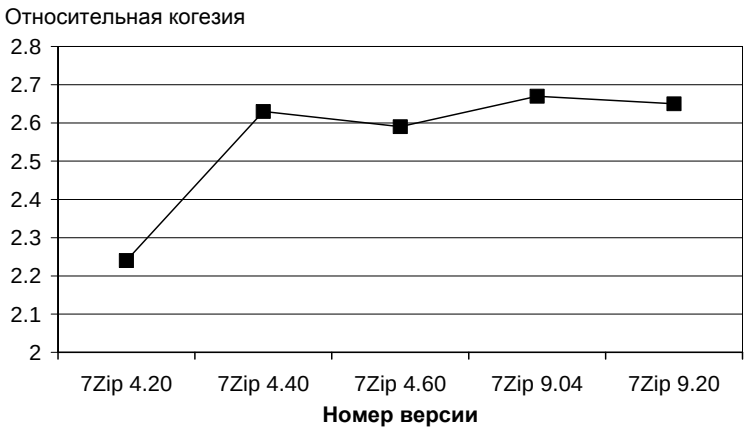

Рис. 9 Зависимость относительной когезии от номера версии

Выводы

Методы мониторинга и анализа объективных количественных показателей внутреннего качества ПО могут быть использованы на этапе разработки для объективной оценки качества и выбора проектных решений, а на этапе сопровождения для определения эффективности проводимых действий по рефакторингу и реинжинирингу программного продукта.

Для обработки полученных данных использованы классические методы математической статистики, включая расчёт центральных моментов, анализ законов распределения с помощью гистограмм, определение коэффициентов корреляции рангов Спирмена. Выполненный анализ позволил сделать качественные выводы о динамике развития проекта 7-Zip.

Список использованных источников

1. Chaos summary for 2010, The Standish Group, 2010, 27 p.

2. T23E-T10E Standish group report [электронный ресурс] // – режим доступа: [http://www.spinroot.com], 1995.

3. *Malhotra R., Chug A.* Software Maintainability Prediction using Machine Learning Algorithms. Software Engineering : An International Journal (SEIJ), Vol. 2, No. 2, September 2012. pp 19-36.

4. *Al-Hagery M. A. H.* Model-based factors to extract quality indications in software lines of code. International Journal of Computer Science & Information Technology (IJCSIT), Vol 3, No 2, April 2011. pp. 112-121.

5. ISO/IEC 25010:2011. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

6. *Бахтизин В.В., Неборский С.Н.*: Оценка качества интеллектуальных информационных систем, Белорусский государственный университет информатики и радиоэлектроники, 2012.

7. *Thapar S.S., Singh P., S. Rani*: Challenges to the Development of Standard Software Quality Model, International Journal of Computer Applications, 2012.

8. *Crt Gerlec, Marjan Hericko*: Evaluating Refactoring with a Quality Index, World Academy of Science, Engineering and Technology 39 2010.

9. *Abbas A. S., Jeberson W., Klinsega V.V.* A Literature Review and Classification of Selected Software Engineering Researches. International Journal of Engineering and Technology. Volume 2 No. 7, July, 2012. pp. 1256-1282.

10. Свободное программное обеспечение – Википедия. [электронный ресурс] // – режим доступа: [http://ru.wikipedia.org/wiki/Свободное_программное_обеспечение] от 4/12/2011.

11. Проприетарное программное обеспечение [электронный ресурс] // – режим доступа: [http://ru.wikipedia.org/] от 4/12/2011.

12. Свободное программное обеспечение [электронный ресурс] // – режим доступа: [http://ru.wikipedia.org/] от 4/12/2011.

13. Open source – Wikipedia, the free encyclopedia [электронный ресурс] // – режим доступа: [http://en.wikipedia.org/wiki/Open_source] от 4/12/2011.

14. *Barkmann H., Lincke R., Lowe W.*: Quantitative Evaluation of Software Quality Metrics in Open-Source Projects, Vaxjo University, 2008.

15. Space product assurance. Software metrication programme definition and implementation. ECSS Secretariat, Requirements & Standards Division, 2011.

16. Метрика программного обеспечения [электронный ресурс] // – режим доступа: [http://ru.wikipedia.org/] от 4/12/2011.

17. *Cem Kaner, Walter P. Bond.* Software Engineering Metrics: What Do They Measure and How Do We Know? 10TH International Software Metrics Symposium, Metrics 2004 KANER / BOND – 1. pp 1-12.

18. *Hsueh N., Chu P., Chu W.*: A quantitative approach for evaluating the quality of design patterns, The Journal of Systems and Software, 2007.

19. *Al-Hagery M. A. H.* Contributors to Reduce Maintainability Cost at the Software Implementation Phase/ International Journal of Software Engineering (IJSE), Volume (3), Issue (2). 2012., pp. 11-22.

20. Coverity scan open source report. [электронный ресурс] // – режим доступа: [http://scan.coverity.com] от 18/11/12.

21. *Sheikh U. F., Quadri S. M. K., Nesar A.*: Software Measurements And Metrics: Role In Effective Software Testing, International Journal of Engineering Science and Technology, 2001.

22. *Вовк О.Б.* Аналіз та оцінювання якості програмного продукту, Національний університет «Львівська політехніка», 2003.

23. Метрология Программного Обеспечения / Ответы на экзаменационные вопросы [электронный ресурс] // – режим доступа: [http://www.studfiles.ru] от 6/12/2011.

24. Концепция аналитической оценки характеристик качества программных компонен-

тов.– /Лаврищева Е.М., Рожнов А.М.– Проблеми програмування.–2004.–№3–4.– С.180–187.

25. *С. Макконелл*, Совершенный Код, 2е издание, Microsoft Press, 2010, 268с.

26. IBM - IBM Rational ClearCase V7.1 – Rational ClearCase – Software. [электронный ресурс] // – режим доступа: [http://www-01.ibm.com/software/awdtools/clearcase/] от 27/11/2012.

27. *Евсеев В. В.* Применение программных метрик кода на раннем этапе жизненного цикла программного обеспечения. Восточно-Европейский журнал передовых технологий № 1/2 (49), 2011. С 19-21.

28. *T. J. McCABE.* A Complexity Measure IEEE Transactions On Software Engineering, Vol. SE-2, NO.4, December 1976, pp 308-320.

29. 7-Zip. [электронный ресурс] // – режим доступа: [http://ru.wikipedia.org/wiki/7-Zip] от 29/11/12.

30. 7-Zip on SourceForge.net. [электронный ресурс] // – режим доступа: [http://sourceforge.net/projects/sevenzip/] от 29/11/12.

31. 7-Zip History. [электронный ресурс] // – режим доступа: [http://7-zip.org.ua/history.txt] от 29/11/12.

32. CppDepend Home page. [электронный ресурс] // – режим доступа: [http://www.cppdepend.com/] от 29/11/12.

33. C++ Metrics Definitions. [электронный ресурс] // – режим доступа: [http://www.cppdepend.com/Metrics.aspx] от 29/11/12.

34. Корреляция. [электронный ресурс] // – режим доступа: [http://ru.wikipedia.org/] от 15/11/12.

35. Коэффициент корреляции рангов Спирмена. [электронный ресурс] // – режим доступа: [http://cito-web.yspu.org/link1/metod/met125/node36.html] от 15/11/12.

36. Корреляция. [электронный ресурс] // – режим доступа: [http://ru.wikipedia.org/] от 15/11/12.

37. Коэффициент корреляции рангов Спирмена. [электронный ресурс] // – режим доступа: [http://cito-web.yspu.org/link1/metod/met125/node36.html] от 15/11/12.

Сведения об авторах:

Туркин Игорь Борисович – д.т.н., професор, зав. каф. інженерії програмного забезпечення Національного аерокосмічного університету ім. Н.Е. Жуковського «ХАІ».
e-mail: energy@d4.khai.edu.

Сыромятников Артем Владимирович – студент Національного аерокосмічного університету ім. Н.Е. Жуковського «ХАІ».
e-mail: artem.syromiatnikov@gmail.com.

Кузнецова Юлия Анатольевна – асистент кафедри інженерії програмного забезпечення Національного аерокосмічного університету ім. Н.Е. Жуковського «ХАІ».
e-mail: JK-Sv@yandex.ru.