Выбор метрики размера проекта в модели оценки трудоемкости разработки программ

Титов А. И.
Петербургский государственный университет путей сообщения Императора Александра I
Санкт-Петербург, Россия
titovvvv@rambler.ru

Аннотация. **В статье рассматриваются различные подходы к оценке трудоемкости разработки программного обеспечения (ПО). Анализируется зависимость оценки трудоемкости разработки ПО от размера проекта. Описываются основные виды существующих метрик и возможность их применения. Предлагаются показатели для сравнения метрик.**

Ключевые слова: **управление проектами, оценка трудоемкости, разработка ПО, метод функциональных точек, метод ucp, uml.**

Введение

В сфере разработки программного обеспечения управление расписанием проекта приобретает особую важность, поскольку большая часть стоимости разработки складывается, как правило, из непосредственных трудозатрат исполнителей [1]. Также создаваемые системы могут иметь высокую сложность, а исходные требования могут изменяться на разных этапах жизненного цикла [2, 3]. Эти факторы осложняют планирование и повышают риск неуспешного завершения проекта. Чтобы определить, реалистичны ли цели проекта, а также повысить точность планирования, используется оценка трудоемкости разработки программного обеспечения.

Выбор метода оценки должен быть обоснован, в первую очередь, решаемой задачей. Для одних проектов сначала необходимо оценить объем функциональности, а затем, исходя из полученной оценки, определить сроки реализации и объемы работ. В других возможна противоположная ситуация: сначала определяются бюджеты и временны́е рамки, после чего оценивается объем реализуемых функций. Также на выбор метода оценки влияют следующие факторы [4]:

- размер проекта;
- стиль разработки;
- стадия разработки;
- возможная точность.

Сегодня разработано множество методов оценки трудоемкости, некоторые из них универсальны, другие придуманы специально для управления определенным портфелем проектов. Из наиболее распространенных методов оценки можно выделить несколько категорий:

- экспертная оценка;
- регрессионные модели;
- функциональные точки;
- нейронные сети;
- комбинированные методы.

Наиболее перспективны комбинированные методы, поскольку при совмещении подходов может возникнуть принципиально новое решение, позволяющее получить более высокую точность или расширить область применения. Так, при объединении нейросетевого подхода к оценке трудозатрат с регрессионными моделями возможно создание универсальной модели с большим числом показателей, которые могут являться входными параметрами. Поскольку для разных категорий методов оценки существуют принципиально разные способы учета фактора размера проекта, возникает потребность в выборе метрики размера проекта.

Среди прочих факторов, влияющих на оценку, размер проекта является наиболее важным показателем [4]. Хотя оценки размера недостаточно для понимания общей сложности разрабатываемого продукта, существует явная зависимость между размером проекта и его трудоемкостью. На рис. 1 показана зависимость роста объема работ от увеличения размера проекта, рассчитанная по модели COCOMO. В качестве исходных данных для построения зависимости взяты проекты, использованные при разработке модели [5, 6]. На полученном графике можно увидеть зависимость объема работ от размера проекта, причем с ростом размера проекта общая динамика становится более ярко выраженной.

Рис. 1. Зависимость роста объема работ от увеличения размера проекта

Чтобы определить, по каким показателям можно сравнивать метрики размера ПО, необходимо рассмотреть основные группы метрик подробнее.

Число строк кода

Количество строк кода (Sourcelinesofcode) – это размерно-ориентированная метрика программного обеспечения, в которой объем ПО рассчитывается исходя из количества строк в тексте исходного кода.

Эта методика возникла в 1950-е годы. Основным носителем информации в те времена была перфокарта, причем на одной перфокарте кодировалась одна строка исходного кода. Поскольку каждая строка кода являлась отдельным физическим носителем, можно было подсчитать число этих объектов и определить трудоемкость и производительность труда программистов.

Среди методик учета числа строк есть две основные:

• по числу физических строк (LOC) – определяется как общее число строк исходного кода, включая комментарии и пустые строки;

• по числу логических строк кода (LLOC) – определяется как общее количество команд и зависит от используемого языка программирования. Если язык поддерживает размещение нескольких команд в одной строке, то одна физическая строка должна быть учтена как несколько логических, если она содержит более одной команды языка.

Также имеются производные от основных методик, которые в зависимости от задачи могут содержать дополнительную информацию по следующим показателям:

• число пустых строк;

• число строк, содержащих комментарии;

• процент комментариев (отношение строк кода к строкам комментария, производная метрика стилистики);

• среднее число строк для функций (классов, файлов);

• среднее число строк, содержащих исходный код для функций (классов, файлов);

• среднее число строк для модулей.

Наибольшее распространение эта метрика получила в языках программирования, в которых одна строка кода реализует строго одну команду. В современных высокоуровневых языках одну и ту же функциональность можно описать различным числом строк кода, поэтому для них данная метрика может слабо коррелировать с реальными трудозатратами. Кроме того, при использовании современных средств разработки ПО часто используется генерация кода для определенных действий, что может еще сильнее усложнить итоговую взаимосвязь между числом строк и трудоемкостью.

При этом у измерений в строках кода есть ряд преимуществ. Например, данные по количеству строк в завершенных проектах или модулях программ могут быть легко собраны при помощи служебных средств интегрированных сред разработки (IDE) или специальных программ.

Метрики Холстеда

Метрики, основанные на анализе числа строк и синтаксических элементов исходного кода программы, были предложены М. Холстедом (MauriceHalstead) в 1977 г. [7]. Метрики Холстеда (Halstead complexity measures) частично позволяют учесть возможность реализации одной и той же функциональности разным числом строк и операторов. Наиболее частым сценарием использования этих метрик является оценка сложности промежуточных продуктов разработки, однако набор метрик также содержит оценки размера.

Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

• NUOprtr (Number of Unique Operators) – число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);

• NUOprnd (Number of Unique Operands) – число уникальных операндов программы (словарь операндов);

• Noprtr (Number of Operators) – общее число операторов в программе;

• Noprnd (Number of Operands) – общее число операндов в программе.

На основе этих характеристик вычисляют различные метрики размера, сложности и качества программ, такие как:

• словарь программы (Halstead Program Vocabulary):

$$HPVoc = NUOprtr + NUOprnd;$$

• длина программы (Halstead Program Length):

$$HPLen = Noprtr + Noprnd;$$

• объем программы (Halstead Program Volume):

$$HPVol = HPLen \cdot \log_2 HPVoc;$$

• сложность программы (Halstead Difficulty):

$$HDiff = \frac{NUOprtr}{2} \cdot \frac{NOprnd}{NUOprnd};$$

• усилия на разработку программы (Halstead Effort):

$$HEff = HDiff \cdot HPVol.$$

Хотя метрики Холстеда позволяют использовать дополнительные возможности по анализу исходного кода, которые отсутствуют в метрике числа строк кода, с точки зрения оценки размера проекта большая часть задач остается нерешенной. Оценка общего числа операторов и их операндов до завершения этапа разработки программы может оказаться еще более сложной задачей, чем оценка числа строк кода.

Функциональные точки

Анализ функциональных точек (Function points) – это метод измерения размера программного обеспечения с точки зрения пользователей системы [8]. Метод был разработан Аланом Альбрехтом (Alan Albrecht) в середине 1970-х годов, впервые опубликован в 1979 г. Широкое распространение эта методика получила в середине 1980-х годов, после того как была сформирована организация IFPUG, занимающаяся развитием метода, а также публикующая новые версии [9]. На текущий момент актуальна версия метода 4.3.

Данный метод предназначен для оценки объема программного продукта по функциональной модели, т. е. оценивается объем функций разрабатываемой системы. Основная цель этого метода – в декомпозиции системы таким образом, чтобы обеспечить приемлемую сложность анализа. Базовой единицей измерения, на которой основывается данный метод, является функциональная точка. Оценка размера в функциональных точках базируется на количестве и сложности следующих элементов:

• внешних входных элементов – всех элементов, предназначенных для ввода или управления данными, которые поступают в систему;

• внешних выходных элементов – всех элементов для ввода и управления данными, которые выходят за внешние границы системы;

• внешних запросов – комбинации входных и выходных элементов, в которых входной элемент сопоставляется с простой выходной формой;

• внутренних логических файлов – каждого логического файла (группы данных), который создается или используется в системе;

• внешних интерфейсных файлов – каждого файла под управлением другой системы, с которым взаимодействует измеряемая программа.

Общая схема взаимодействия функциональных типов представлена на рис. 2.

Анализ функциональных точек включает в себя следующие этапы [10]:

1) определения типа оценки;

2) определения области оценки и границ продукта;

3) подсчета функциональных точек, связанных с данными;

4) подсчета функциональных точек, связанных с транзакциями;

5) определения суммарного количества не выровненных функциональных точек (UFP);

6) определения значения коэффициента выравнивания (FAV);

7) расчета количества выровненных функциональных точек (AFP).

На двух первых этапах анализируется предмет оценки, определяются границы продукта, выявляется разрабатываемая функциональность. На этапах 3–5 происходят общий подсчет и суммирование функциональных точек без учета коэффициента выравнивания. При выполнении 6-го пункта в методе расчета вводятся общесистемные требования, которые накладывают различные ограничения и увеличивают сложность разработки. Сложность этих требований оцениваются коэффициентом выравнивания (FAV), который зависит от 14 общих системных характеристик (total degree of influence, TDI) и вычисляется по формуле

$$FAV = (0{,}01 \cdot \sum TDI) + 0{,}65.$$

В 7-м пункте оценка подсчитывается в выровненных функциональных точках, причем вариант подсчета зависит от изначального выбора типа оценки. Например, базовая оценка, учитывающая только непосредственно разработку продукта, вычисляется как произведение количества точек и коэффициента выравнивания:

$$AFP = UFP \cdot FAV.$$

При необходимости может быть выполнен еще один этап вычисления – преобразование величины AFP или UFP в число строк кода. Коэффициенты преобразования выбираются в зависимости от языка программирования. Пример коэффициентов преобразования, по данным ряда исследований [4, 11] приведен в табл. 1.

Таблица 1

Коэффициенты преобразования функциональных точек в число тысяч строк кода (SLOC)

Язык	Число команд в функциональной точке		
	Минимум	Среднее значение	Максимум
C	60	128	170
C++	40	55	140
C#	40	55	80
Java	40	55	80
SQL	7	13	15
Макроассемблер	130	213	300

В целом измерение размера проекта функциональными точками является более актуальной метрикой, чем измерение числом строк кода. Ключевое преимущество этого подхода – оценка основана на требованиях к продукту, что позволяет оценить трудоемкость на самых ранних этапах работы над проектом, сразу после выявления необходимых требований. На последующих этапах работы оценку можно уточнить, поэтому этот метод можно применять при использовании гибких методологий разработки ПО. Также надо отметить, что декомпозиция системы, выполняемая на начальных этапах расчета, может в дальнейшем стать основой для документирования функциональности системы, а значит, снизить трудозатраты проектной команды.

Однако использование функциональных точек в качестве единиц измерения имеет ряд существенных недостатков. Для вычисления функциональных точек необходимо детально изучить спецификацию требований и подсчитать все входные и выходные элементы, файлы, транзакции, что может быть весьма трудоемко. При этом полученная оценка весьма субъективна, поскольку некоторые бизнес-процессы могут иметь высокую алгоритмическую сложность, но обладать достаточно простыми внешними вводами и выводами. По мере развития методологий разработки программного

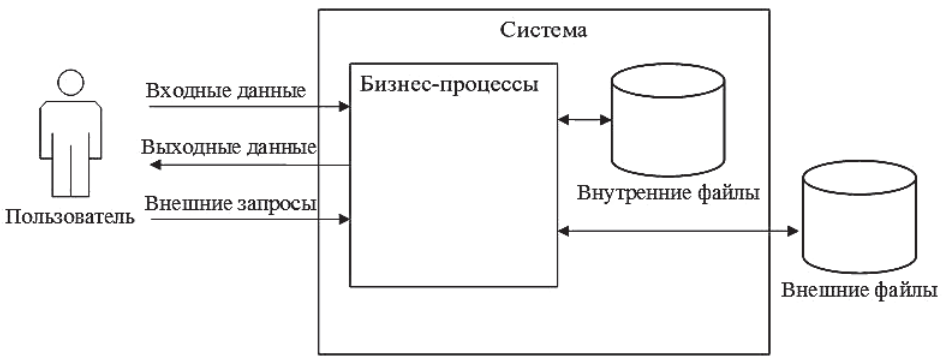

Рис. 2. Общая схема взаимодействия функциональных типов

обеспечения эксперты предложили новые методы оценки, основанные на методе функциональных точек.

Точки свойств

Метод точек свойств (Feature points) представляет собой расширение метода функциональных точек, которое учитывает не только требования к системе, но и особенности ее реализации. Главное отличие от метода функциональных точек заключается в том, что получаемая оценка корректируется с учетом алгоритмической сложности реализации. К исходным типам элементов добавляется еще один – алгоритмы. В методе точек свойств алгоритмы определяются как набор правил, определяющих какую-либо существенную вычислительную задачу.

Благодаря учету алгоритмической сложности точки свойств лучше адаптированы к оценке систем с высокой сложностью алгоритмов, например [12]:

• системного ПО, операционных систем или компиляторов;

• систем, функционирующих в реальном времени;

• инженерных приложений, таких как системы автоматизированного проектирования (САПР) или математическое ПО;

• систем искусственного интеллекта;

• систем поддержки телекоммуникаций.

Объектные точки

В современных методологиях разработки объектно-ориентированное программирование (ООП) занимает особое место, фактически является наиболее широко распространенной методологией. Поскольку в изначальном варианте метода функциональных точек не было предусмотрено применения объектно-ориентированного подхода, был разработан адаптированный вариант, оперирующий терминами ООП. Его принципиальным отличием от других вариаций метода функциональных точек является то, что он не расширяет стандартный набор типов элементов, а использует совершенно иные [13]:

• формы (Screens definitions);

• отчеты (User reports);

• модули (3GL Modules).

Фактически в данной метрике каждому уникальному классу или объекту назначается одна объектная точка (Objectpoints). В целом оценка производится примерно по тем же этапам, что и функциональные точки:

1) подсчет количества форм, отчетов и компонентов;

2) классификация каждого экземпляра объекта по уровню сложности;

3) определение веса для объектов;

4) суммирование взвешенных объектов;

5) определение процента повторного использования кода;

6) определение уровня продуктивности;

7) подсчет значения длительности работы в человеко-месяцах.

По сравнению с методом функциональных точек ощутимые различия есть при использовании факторов преобразования на этапах 5–7. Во-первых, подсчитывается процент повторного использования, исходя из него количество объектных точек пересчитывается:

$$\mathrm{NOP}=\frac{\mathrm{OP}\cdot(100-r)}{100},$$

где OP – количество объектных точек; r – процент повторного использования кода.

В конце рассчитывается значение длительности работы (PM) в человеко-месяцах:

$$\mathrm{PM}=\frac{\mathrm{NOP}}{\mathrm{PROD}},$$

где PROD – оценка уровня продуктивности, определяемая исходя из опыта и возможностей разработчиков.

Данная метрика удобна для оценки проектов, которые ведутся по объектно-ориентированной методологии и имеют большое число компонентов с визуальной составляющей. Поскольку определение числа точек больше ориентировано на визуальные компоненты, в проектах с высокой алгоритмической сложностью подсчет объектов может затрудниться.

Метод UCP

Метод UCP (Usecasepoints) представляет собой оценку размера проектов на основе диаграмм UML (Unified Modeling Language) и методологии RUP (Rational Unified Process). Как и многие другие современные методы оценки, UCP базируется примерно на тех же принципах, что и метод функциональных точек. Главное различие заключается в замене единиц измерения с функциональных точек на варианты использования (Use Cases).

Оценка по методу UCP складывается из следующих этапов [14]:

1) *оценка акторов*. На этом шаге определяются все акторы системы (сущности, взаимодействующие с системой извне). После определения для каждого актора в соответствии с его оценкой устанавливается вес (табл. 2);

Таблица 2

Оценки акторов в модели UCP

Оценка актора	Описание	Вес
Простой	Внешняя система, взаимодействующая с помощью заранее описанного API (REST, SOAP, DLL)	1
Средний	Внешняя система, взаимодействующая с помощью более сложного или гибкого API либо с помощью сетевых протоколов (TCP/IP, FTP, HTTP) или СУБД	2
Сложный	Взаимодействие с пользователем через графический интерфейс	3

2) *нескорректированная оценка вариантов использования*. Рассчитывается исходя из количества транзакций (табл. 3).

Таблица 3

Оценка вариантов использования в модели UCP

Оценка варианта использования	Количество транзакций	Вес
Простой	До 3	5
Средний	От 4 до 7	10
Сложный	Более 7	15

Существуют также альтернативные критерии для оценки сложности, например, количество классов, реализуемых в рамках одного варианта использования, либо количество объектов в базе данных, изменяемых в рамках одного варианта использования;

3) *оценка технических факторов*. Используется для определения сложности архитектуры приложения и степени влияния нефункциональных требований. Каждый фактор оценивается по шкале от 0 (фактор не значим) до 5 (фактор оказывает существенное влияние), после чего умножается на вес фактора. Описания всех 13 факторов, используемых в модели, приведены в табл. 4;

Таблица 4

Оценка технических факторов

Фактор	Описание	Вес
T1	Распределенность системы	2,0
T2	Время отклика/производительность	1,0
T3	Увеличение продуктивности пользователя	1,0
T4	Сложность внутренней обработки	1,0
T5	Повторная используемость кода	1,0
T6	Удобство установки	0,5
T7	Удобство использования	0,5
T8	Переносимость на другие платформы	2,0
T9	Поддержка системы	1,0
T10	Параллельные вычисления	1,0
T11	Функции безопасности	1,0
T12	Доступ для третьих лиц	1,0
T13	Необходимость обучения пользователя	1,0

4) *оценка внешних факторов*. Используется для определения коэффициента влияния организационных рисков на разработку. Вычисления производятся по аналогии с техническими факторами. Описания внешних факторов приведены в табл. 5;

Таблица 5

Оценка внешних факторов

Фактор	Описание	Вес
E1	Знание предметной области	1,5
E2	Опыт разработки приложений	0,5
E3	Навык использования ООП	1,0
E4	Уровень ведущего аналитика	0,5
E5	Мотивация проектной команды	1,0
E6	Неизменяемость требований	2,0
E7	Частичная занятость сотрудников	–1,0
E8	Сложность языка разработки	–1,0

5) *окончательный подсчет*. Оценивается общее число вариантов с учетом прочих факторов по формуле

$$UCP = (UUCW + UAW) \cdot TCF \cdot ECF,$$

где UUCW – нескорректированная оценка вариантов использования; UAW – оценка акторов; TCF – оценка технических факторов; ECF – оценка внешних факторов.

При использовании метода UCP как метрики размера проекта вычисления на этом заканчиваются, при использовании в качестве модели оценки трудоемкости требуется перевести полученный результат из количества вариантов использования в трудозатраты в человеко-месяцах [15]. В отличие от вычисления количества элементов, перевод оценки в человеко-месяцы может быть довольно трудоемкой и вариативной задачей, поскольку рекомендуемые коэффициенты перевода (20–28 часов на один элемент) не всегда могут соответствовать реальным трудозатратам. Выбор количества часов на один элемент должен зависеть от степени абстракции при создании диаграмм и опыта разработки схожих модулей.

Выбор показателей для сравнения

После проведения анализа основных метрик размера проекта были определены основные показатели, по которым можно провести сравнение. Поскольку основной целью сравнения является выбор метрики размера проекта, наиболее подходящей для комбинированных методов оценки трудоемкости разработки, предпочтение было отдано более универсальным показателям, которые характеризуют метрики с точки зрения управления жизненным циклом проекта.

1. Возможность оценки на ранних этапах разработки (П1). Оценка размера проекта имеет наибольшую важность именно на ранних этапах разработки, поскольку является базовой информацией для построения календарного графика работ. Оценивание на поздних этапах проекта, например, после завершения этапа разработки и начала этапа внедрения, не дает информации, которая может существенно повлиять на ход завершения проекта. При этом оценка размера на ранних этапах должна иметь возможность постепенного дополнения в процессе сбора и анализа требований к продукту, а также непосредственно при разработке.

2. Возможность оценки бизнес-аналитиком (П2). Приблизительная оценка размера проекта может потребоваться на самых ранних этапах проекта, например, при заключении контракта. Если оценка требуется до детализации требований к продукту и формирования проектной команды разработчиков, то в этой ситуации оценку может выполнить бизнес-аналитик. В этом случае на самом базовом уровне оценивается объем необходимой функциональности и количество связей с внешними системами. Полученная оценка должна быть выполнена таким образом, чтобы ее можно было детализировать на этапах анализа требований и разработки.

3. Наличие инструментов для автоматизированной оценки (П3). В определенных ситуациях оценить размер необходимо не только для планирования, но и для характеристики уже выполненных работ. Поскольку подсчет оценки требует определенных трудозатрат, для его выполнения на поздних этапах проекта следует использовать метрики с возможностью автоматизации анализа и вычислений.

4. Учет алгоритмической сложности (П4). Оценка, основанная исключительно на объеме функциональности, может дать искаженные данные с точки зрения трудоемкости разработки. При реализации нескольких похожих по функ-

циональности компонентов трудозатраты могут значительно различаться в зависимости от алгоритмической сложности этих компонентов.

Сравнение метрик

Рассмотренные метрики были проанализированы и оценены по выбранным показателям (табл. 6).

Таблица 6

Сравнение метрик размера ПО

Метрика	П1	П2	П3	П4
SLOC			✓	
Halstead			✓	✓
Function Points	✓	✓		
Feature Points	✓			✓
Object Points	✓	✓		
UseCasePoints	✓	✓		✓

Как можно увидеть из результатов сравнения, среди выбранных метрик отсутствуют варианты, которые соответствовали бы всем показателям. Среди метрик, основанных на подсчете строк кода, наиболее удачна реализация метрик Холстеда, поскольку они позволяют оценить не только размер, но и алгоритмическую сложность. Для оценки на ранних этапах разработки следует использовать одну из адаптаций метода функциональных точек. Среди группы этих метрик метод UCP соответствует наибольшему числу показателей. Кроме соответствия выбранным показателем этот метод обладает и другими преимуществами: возможностью оценки в рамках объектно-ориентированного подхода и использованием нотации UML, которая достаточно широко распространена в IT-отрасли.

Чтобы подтвердить возможность использования метода UCP в оценке на ранних этапах разработки ПО, были спроектированы диаграммы UML для системы АСМАДС [16]. На начальных этапах проекта разработки отсутствовала

Рис. 3. Пример диаграммы вариантов использования для одного из компонентов системы АСМАДС

большая часть нефункциональных требований, а также требований к способу реализации, таких как платформа и язык программирования. Исходя из этого, приближенная оценка была получена при помощи диаграмм вариантов использования по первоначальным функциональным требованиям. Пример диаграммы, построенной для одного из компонентов системы, представлен на рис. 3.

При проектировании определены основные классы системы и разработаны диаграммы последовательностей, на основе которых уточнена оценка по диаграммам вариантов использования (рис. 4).

Таким образом, при использовании метода UCP предварительная оценка легла в основу дальнейшего проектирования системы и была уточнена на более поздних этапах разработки.

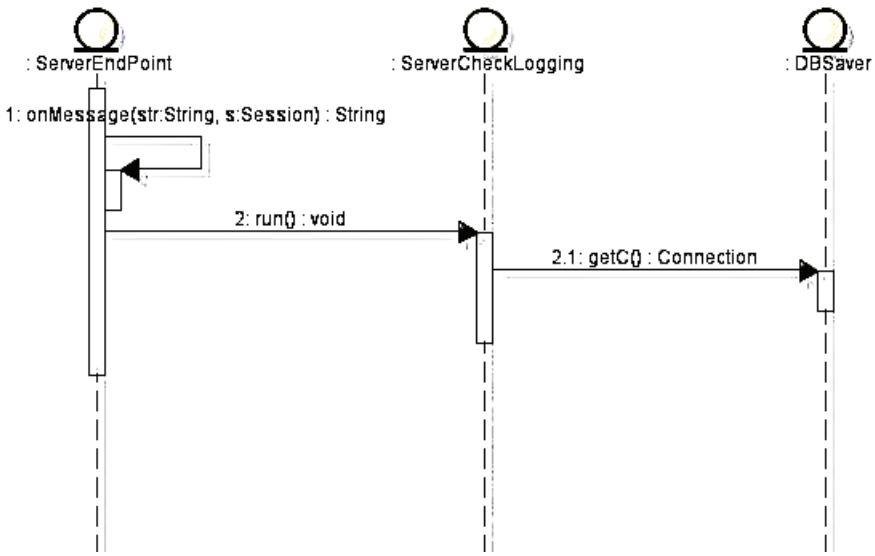

Рис. 4. Пример диаграммы последовательностей для варианта использования «Проверка авторизации»

Заключение

В условиях непрерывного развития методологий разработки ПО необходимо построить модели оценки проекта, соответствующие современным тенденциям разработки. Поскольку метрика размера проекта является важной частью многих моделей оценки трудоемкости, разработка новых моделей должна начинаться именно с выбора метрики.

По результатам сравнения ключевых метрик предложены показатели, на основе которых можно выбрать метрики в зависимости от требований к разрабатываемой модели проекта. Также по выбранным показателям проведено сравнение для выбора метрики, наиболее подходящей для комбинированных методов оценки трудоемкости. Метод UCP, наиболее соответствующий выбранным показателям, применен для оценки трудоемкости системы АСМАДС, чтобы подтвердить возможность его использования для оценки размера проекта.

Дальнейшие исследования целесообразно продолжить в направлении:

• расширения числа показателей для сравнения метрик;

• разработки новых метрик размера проекта, позволяющих производить оценку на разных итерациях проекта в гибких методологиях разработки;

• улучшения метода UCP для повышения точности оценки при использовании разных типов диаграмм UML.

Литература

1. Кульдин С. П. Генетический подход к проблеме оценке сроков и трудоемкости разработки программного обеспечения с заданными требованиями к качеству / С. П. Кульдин // Прикладная информатика. – 2010. – № 5 (29). – С. 30–42.

2. A Guide to the Project Management Body of Knowledge (PMBOK Guide). – 5th ed. (Руководство к своду знаний по управлению проектами (Руководство PMBOK. Russian)). 2014. – 589 с.

3. Липаев В. В. Проектирование и производство сложных заказных программных продуктов / В. В. Липаев. – М.: СИНГТЕГ, 2011. – 399 с.

4. Макконелл С. Сколько стоит программный проект / С. Макконелл. – М.: Русская редакция; СПб.: Питер, 2007. – 304 с.

5. Boehm B. Software engineering economics / B. Boehm. – New Jersey: Prentice-Hall, 1981. – 42 p.

6. Boehm B. Software Cost Estimation with Cocomo II. New Jersey, Prentice-Hall. 2000. 544 p.

7. Halstead M. H. Elements of Software Science / M. H. Halstead. – Amsterdam: Elsevier North-Holland, Inc. 1977. – 127 p.

8. Albrecht J. Software function, source lines of codes, and development effort prediction: a software science validation / J. Albrecht, J. E. Gaffney // IEEE Trans Software Eng. SE-9. – 1983. – P. 639–648.

9. IFPUG. Function Point Counting Practices Manual Release 4.0. Int. Function Point Users Group. – Westerville, Ohio, 1994. – 370 p.

10. Архипенков С. Я. Лекции по управлению программными проектами / С. Я. Архипенков. – М., 2009. – 127 с.

11. Тютюнников Н. Н. Оценка размера создаваемого программного средства с использованием функциональных точек / Н. Н. Тютюнников // Перспективы развития информационных технологий. – 2014. – № 18. – URL : http://cyberleninka.ru/article/n/otsenka-razmera-sozdavaemogo-programmnogo-sredstva-s-ispolzovaniem-funktsionalnyh-tochek.

12. Фатрелл Р. Т. Управление программными проектами. Достижение оптимального качества при минимуме затрат / Фатрелл Р. Т., Шафер Д. Ф., Шафер Л. И.; пер. с англ. – М.: СПб.: Киев: Вильямс, 2004. – 1136 с.

13. Minkiewicz A. Measuring Object-Oriented Software With Predictive Object Point / A. Minkiewicz // ASM'97 – Appl. Software Meas. – Atlanta, 1997. – P. 225–254.

14. Banerjee G. Use case points – an estimation approach / G. Banerjee. – URL : http://www2.fiit.stuba.sk/~bielik/courses/msi-slov/reporty/use_case_points.pdf.

15. Cohn M. Estimating with Use Case Points / M. Cohn // Methods Tools. – 2005. – Vol. 13, № 3. – P. 3–13.

16. Брынь М. Я. Программный комплекс для мониторинга деформаций особо опасных объектов / М. Я. Брынь, А. Д. Хомоненко, В. П. Бубнов, А. А. Никитчин, С. А. Сергеев, П. А. Новиков, А. И. Титов // Проблемы информационной безопасности. Компьютерные системы. – 2014. – № 1. – С. 36–41.

Selecting Size of Project Metrics in Software Development Estimation Model

Titov A. I.
Petersburg State Transport University
Saint-Petersburg, Russia
titovvvv@rambler.ru

Abstract. **The article describes various approaches to the effort estimation of software development. It is also considered the dependence of bounds for the complexity of software development on the project size. It describes the main types of existing metrics and their application.The size metrics for comparing are described.**

Keywords: **projectmanagement, software development effort estimation, function points, use case points, uml.**

References

1. Kul'din S. P. Genetic approach to the problem of estimating the timing and complexity of software development with the specified quality requirements [Geneticheskijpodhod k problem ocenki srokov I trudoemkosti razrabotki programmnogo obespechenija s zadannymi trebovanijami k kachestvu], *Prikladnaja informatika [Appl. Inf.]*, 2010, no. 5 (29), pp. 30-42.

2. A Guide to the Project Management Body of Knowledge (PMBOK Guide). 5th ed. 2014, 589 p.

3. Lipayev V. V. *Proyektirovaniye i proizvodstvo slozhnykh zakaznykh programmnykh produktov* [Design and production of complex custom software], Moscow, SINGTEG, 2011, 399 p.

4. Makkonell S. *Skol'ko stoit programmnyj proekt* [Software Estimation: Demystifying the Black Art], Moscow, Russkaja redakcija; St. Petersburg, Piter, 2007, 304 p.

5. BoehmB. Softwareengineeringeconomics. New Jersey, Prentice-Hall. 1981. 42 p.

6. Boehm B. Software Cost Estimation with Cocomo II, New Jersey, Prentice-Hall, 2000, 544 p.

7. Halstead M. H. Elements of Software Science, Amsterdam: Elsevier North-Holland, Inc., 1977, 127 p.

8. Albrecht J., Gaffney J. E. Software function, source lines of codes, and development effort prediction: a software science validation, *IEEE Trans Software Eng. SE-9*, 1983, pp. 639-648.

9. IFPUG. Function Point Counting Practices Manual Release 4.0. Int.l Function Point Users Group, Westerville, Ohio, 1994, 370 p.

10. Arhipenkov S.Ja. *Lekcii po upravleniju programmnymi proektami* [Software project management lectures], Moscow, 2009, 127 p.

11. Tyutyunnikov N. N. Otsenka razmera sozdavayemogo programmnogo sredstva s ispolzovaniyem funktsionalnykh tochek [Estimate the size to create software tools using functional points], *Perspektivy razvitiya informatsionnykh tekhnologiy [Prospects of development of information technologies]*, 2014, no. 18 (available at: http://cyberleninka.ru/article/n/otsenka-razmera-sozdavaemogo-programmnogo-sredstva-s-ispolzovaniem-funktsionalnyh-tochek).

12. Fatrell R. T., Shafer D. F., Shafer L. I. *Upravlenie programmnymi proektami. Dostizhenie optimal'nogo kachestva pri minimum zatrat* [Quality software project management first edition], Moscow, St. Petersburg, Kiev, Vil'jams, 2004, 1136 p.

13. Minkiewicz A. Measuring Object-Oriented Software With Predictive Object Point, *ASM'97 – Appl. in Software Meas.*, Atlanta, 1997, pp. 225-254.

14. Banerjee G. Use case points – an estimation approach. (available at: http://www2.fiit.stuba.sk/~bielik/courses/msi-slov/reporty/use_case_points.pdf).

15. Cohn M. Estimating with Use Case Points, *Methods Tools*, 2005, Vol. 13, no. 3, pp. 3-13.

16. Bryn M.Ya., Khomonenko A. D., Bubnov V. P., Nikitchin A.A., Sergeyev S.A., Novikov P.A., Titov A. I. Programmnyy kompleks dlya monitoring deformatsiy osoboopasnykh obyektov [Software system for monitoring deformations especially dangerous objects], *Problemy informatsionnoy bezopasnosti. Kompyuternyye sistemy [Problems of information security. Computer systems]*, 2014, no. 1, pp. 36-41.