УДК 004.413:338.5

МОДЕЛИ, МЕТОДЫ И СРЕДСТВА ОЦЕНКИ СТОИМОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Н.А. Сидоров, Д.В. Баценко, Ю.Н. Василенко, Ю.В. Щебетин

Национальный авиационный университет
03058, Киев, проспект космонавта Комарова, 1,
тел.: 406 7396; e-mail: sna@nau.edu.ua

В работе рассматриваются модели, методы и средства оценки стоимости программного обеспечения, приводятся результаты эксперимента по оценке стоимости, рассматриваются особенности калибровки параметров стоимости; делаются выводы о целесообразности применения рассмотренных моделей, методов и средств в разработке ПО.

This article reviews models, methods and tools for software cost estimation also there are given results of the experiment in cost estimation, the peculiarities of calibration are studied and the conclusions are drawn on the expediency of application of reviewed models, methods and tools.

Развитие моделей, методов и средств оценки стоимости программного обеспечения (ПО) достигло уровня практического применения. Однако из-за отсутствия информации, средств и специалистов они не используются при разработке ПО в Украине.

В работе приводятся результаты аналитического обзора литературы с учетом опыта авторов по указанной теме. Работа состоит из четырёх частей. В первой рассматриваются единицы размера ПО, которые используются в моделях, методах и средствах оценки стоимости ПО, во второй – модели и методы оценки, в третьей – средства оценки, в четвёртой – на примере показывается использование средств для оценки ПО.

1. Единицы размера ПО

В оценке стоимости ПО используют две единицы размера: строка кода Line of Code (LOC) [1] и функциональная точка Function Point (FP) [2].

Line of Code – это строка исходного кода ПО (исключаются пустые строки, комментарии и специфические операторы). К преимуществам использования LOC, как единицы размера ПО, относят простоту, а недостатками являются следующие: размер проекта в LOC может быть определён только после его завершения; LOC зависит от языка программирования; LOC не учитывает качество кода. Производительность (S) программиста с использованием LOC подсчитывается по следующей формуле $S = n/m$, где n – количество строк кода, написанных программистом (LOC); m – время работы программиста (в человеко-часах). Видно, что чем больше строк кода, тем выше производительность разработчика. Однако, очевидно можно реализовать одну и ту же функцию, написав меньшее количество строк кода. Единица размера LOC не отражает функциональные свойства кода. Поэтому, если разработчик стремится оптимизировать процесс разработки, с целью уменьшения трудозатрат на реализацию проекта, то при использовании LOC как основной единицы размера проекта под уменьшением трудозатрат подразумевается уменьшение количества строк кода в программе, при этом не оценивается его функциональность.

Существуют также проблемы с применением LOC и в проектах, использующих несколько языков программирования. Например, 10.000 LOC языка C++ очевидно нельзя сравнивать с 10.000 LOC языка COBOL, а в случае применения автоматизированных или основанных на шаблонах, визуальных средств разработки подсчёт LOC тем менее эффективен, чем больше кода создаётся автоматически.

Function Point была введена как альтернатива LOC [2]. Методика анализа функциональных точек была разработана А. Дж. Альбректом (A. J. Albrecht) для компании IBM в середине 70-х годов прошлого столетия, когда возникла потребность в подходе к оценке затрат труда на разработку ПО, который бы не зависел от языка и среды разработки. С 1986 года продвижение методики и разработку соответствующего стандарта продолжает International Function Point User Group (IFPUG). Эта организация разработала руководство по практике применения расчёта функциональных точек Function Point Counting Practices Manual (FPCPM) и последняя версия этого документа (4.1) официально признана ISO как стандарт оценки размера ПО [3].

Методика анализа FP основывается на концепции разграничения взаимодействия. Сущность ее состоит в том, что программа разделяется на классы компонентов по формату и типу логических операций. В основе этого деления лежит предположение, что область взаимодействия программы разделяется на внутреннюю – взаимодействие компонентов приложения, и внешнюю – взаимодействие с другими приложениями.

В соответствии с принятым стандартом [3] используются пять классов компонентов, на которых основывается анализ:

- внутренний логический файл Internal Logical File (ILF) – группа логически связанных данных,

находящихся внутри границ приложения и поддерживаемых вводом извне;

- внешний интерфейсный файл External Interface File (EIF) – группа логически связанных данных, находящихся вне границ приложения и являющихся внутренним логическим файлом для другого приложения;

- внешний ввод External Input (EI) – транзакция, при выполнении которой данные пересекают границу приложения извне. Это могут быть как данные, получаемые от другого приложения, так и данные, вводимые в программу пользователем. Получаемые данные могут быть командами управления или статическими данными. В последнем случае может возникнуть необходимость обновить внутренний логический файл;

- внешний вывод External Output (EO) – транзакция, при выполнении которой данные пересекают границу приложения изнутри. Из ILF и EIF создаются файлы вывода или сообщения и отправляются другому приложению. Вывод также содержит производные данные, получаемые из ILF;

- внешний запрос External Inquiry (EQ) – транзакция, при выполнении которой происходит одновременный ввод и вывод. В результате информация возвращается из одного или более ILF и EIF. Вывод не содержит производных данных, а ILF не обновляются.

Классы компонентов оцениваются по сложности и относятся к категории высокого, среднего или низкого уровней сложности. Для транзакций (EI, EO, EQ) уровень определяется по количеству файлов, на которые ссылается транзакция File Types Referenced (FTR) и количеству типов элементов данных Data Element Types (DET). Для ILF и EIF имеют значение типы элементов записей Record Element Types (RET) и DET. Типы элементов записей это подгруппа элементов данных в ILF или EIF. Типы элементов данных – это уникальное не рекурсивное поле подмножества ILF или EIF. Уровни сложности и соответствующие им значения FTR и DET описаны в FPCPM [3].

Например, для EI с количеством FTR от 3 и более и DET от 5 до 15 уровень сложности определяется как высокий. Далее компоненты распределяются по «весовым категориям» в зависимости от уровня их сложности. Например, ILF средней сложности имеет значение 10, а EQ высокой сложности значение 6. После этого, производится подсчёт нескорректированных функциональных точек Unadjusted Function Point (UFP) по соответствующей формуле [1]:

$$UFP = \sum_{i=1}^{5}\sum_{j=1}^{3} N_{ij} W_{ij},$$

где N_{ij} и W_{ij} соответственно количество экземпляров класса i сложности j и его весовое значение. Результат расчёта может быть скорректирован с помощью фактора регулирования стоимости Value Adjustment Factor (VAF)[4]. При расчёте VAF учитываются четырнадцать общих характеристиках системы General System Characteristic (GSC), которые оценивают общую функциональность разрабатываемого приложения. Эти характеристики отражают возможность повторного использования кода, производительность, возможность распределённой обработки, и другие свойства приложения [2]. Каждой GSC присваивается значение от 0 до 5. После того как учтены все четырнадцать общих характеристик системы рассчитывается фактор регулирования стоимости по следующей формуле [4]:

$$VAF = 0.65 + [(\sum_{i=1}^{14} C_i) / 100],$$

где C_i – степень влияния i-ой GSC. Последним, подсчитывается количество полных функциональных точек [4]: FP = UAF * VAF.

Существуют приложения, в оценке которых использование стандартных функциональных точек не эффективно. Эти приложения следующие [5]: управление процессом в реальном времени, математические вычисления, симуляция, системные приложения, инженерные приложения, встроенные системы. Перечисленные приложения отличаются высокой интенсивностью вычислений, часто основанных на алгоритмах повышенной сложности. Для решения задач расчёта размера указанных приложений в 1986 году организацией Software Productivity Research (SPR) была разработана методика анализа характеристических точек ПО (feature points) [5]. Сущность ее состоит в том, что оценивается количество алгоритмов в программе и незначительно модифицируется степень значимости (weighting values) для расчёта FP. Эта методика считается экспериментальной [5].

2. Методы и модели оценки стоимости ПО

Методы и модели оценки стоимости ПО можно разделить на две группы: неалгоритмические методы и алгоритмические модели. К неалгоритмическим методам относятся Price-to-win, оценка по Паркинсону, экспертная оценка, оценка по аналогии. К алгоритмическим моделям относятся SLIM и COCOMO.

Неалгоритмические методы. Сущность неалгоритмических методов состоит в том, что при оценке стоимости ПО используются определенные схемы и принципы, а не математические формулы. К этим методам относятся следующие:

Price-to-win [1]. Метод основывается на принципе «клиент всегда прав». Суть метода состоит в том, что независимо от предполагаемых реальных затрат на разработку проекта, оценка стоимости ПО корректируется в соответствии с пожеланиями заказчика. Price-to-win фактически является политикой проведения переговоров с

клиентом, поэтому часто применяется компаниями, не имеющими средств для качественной оценки проектов. Применение метода может иметь для разработчика следующие негативные последствия: нехватка ресурсов для выполнения проекта, невыполнение сроков сдачи проекта и как результат – потеря контракта или банкротство [1].

Оценка по Паркинсону [1]. Метод основывается на принципе: «Объем работы возрастает в той мере, в какой это необходимо, чтобы занять время, выделенное на ее выполнение» [6]. Принцип, позднее названный «законом» [1], был впервые высказан С.Н. Паркинсоном и описывал природу взаимодействия бюрократической системы в административных институтах, отображая процесс неэффективного использования ресурсов [6]. В применении к разработке программных проектов, закон Паркинсона используется в виде следующей схемы: чтобы повысить производительность труда разработчика, необходимо уменьшить время, отведённое на разработку [1].

Экспертная оценка [7]. Метод основывается на принципе экспертной оценки и применяется в проектах использующих новые технологии, новые процессы или решающих инновационные задачи. К процессу оценки привлекаются инженеры-разработчики, которые сами оценивают курируемую ими часть проекта. После этого созывается собрание, на котором результаты отдельных оценок интегрируются в единую, целостную систему. Предположения, на которых основывалась оценка отдельных экспертов, заносятся в протокол и открыто обсуждаются. При опросе экспертов используются Дельфийская или расширенная Дельфийская методика, ориентированная на приведение экспертов к консенсусу [1]. В результате достигается баланс оценки при интеграции отдельных компонентов в общую систему. Далее следует очередная стадия покомпонентной оценки, и по мере увеличения количества итераций точность оценки увеличивается.

Оценка по аналогии [8]. Являясь разновидностью экспертной оценки, часто выделяется в отдельный метод. Метод основывается на принципе аналогии [8]. Оценка по аналогии, как и алгоритмические модели, использует эмпирические данные о характеристиках завершённых проектов. Ключевое различие состоит в том, что алгоритмические модели используют эти данные косвенным образом, например, для калибровки параметров моделей, а метод оценки по аналогии с помощью эмпирических данных позволяет отобрать схожие проекты. Схема оценки основанная на указанном принципе состоит из нескольких этапов. На первом этапе осуществляется сбор данных по разрабатываемому проекту. В рамках ЖЦ ПО оптимальными формами для этого являются анализ требований и проектирование. На основе экспертной оценки производится отбор характеристик, по которым будут сравниваться проекты. Выбор характеристик зависит от типа приложения, среды разработки и набора известных параметров приложения. Следующий этап включает в себя поиск и анализ проектов «аналогичных» по выбранным характеристикам разрабатываемому. Результатом данного этапа является, как правило, несколько проектов имеющих наименьшие различия в численных значениях характеристик оценки. Для отбора проектов, наиболее близких разрабатываемому, может использоваться метод измерения Евклидова расстояния в n- мерном пространстве. Каждой характеристике присваивается значение веса (множитель), определяющее значимость характеристики для проекта. В упрощённом варианте вес равен единице, т. е. все характеристики проекта считаются равнозначными по важности. Далее проекты и их соответствующие характеристики отображаются в n – мерном пространстве как точки (n равно количеству переменных, для каждой переменной используется своё измерение), после чего вычисляется Евклидово расстояние между соответствующими точками [8]:

$$d(a,b) := \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 \ldots + (a_n - b_n)^2} = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2},$$

где a и b – точки в пространстве, $a_1 \ldots a_i$ и $b_1 \ldots b_i$ – координаты точек в соответствующих плоскостях.

Проекты, имеющие наибольшее сходство будут расположены ближе всего, т. е. Евклидово расстояние у них будет наименьшим. Последним этапом является экспертная оценка разрабатываемого проекта, в которой значения, взятые из аналогичного проекта используются как базис оценки.

Модели оценки стоимости ПО. Модель оценки стоимости ПО представляет собой одну или несколько функций, которые описывают зависимость между характеристиками проекта и затратами на его реализацию. Модели разделяют по типу используемых функций на линейные, мультипликативные, степенные и использованию исторических данных на эмпирические и аналитические. Наиболее часто реализуемыми и хорошо документированными моделями являются модель Путнэма (степенная, аналитическая) и модель COCOMO (степенная, эмпирическая).

Модель Путнэма (SLIM). Наиболее распространенная модель аналитической группы. Создания для проектов, объемом больше 70 000 строк кода, модель основывается на утверждении, что затраты на разработку ПО распределяются согласно кривым Нордена-Рэйли, которые являются графиками функции, представляющей распределение рабочей силы по времени [9]. Общий вид подобной функции – $v = v_0 \cdot \left(1 - e^{-t^2/2t_p^2}\right)$, где v – полученное значение; t – время; v_0 и t_p – параметры, определяющие функцию. Для большого значения t, кривая стремится к параметру v_0, который называется cost scale factor parameter. Функция возрастает наиболее быстро при $t = t_p$ [9]. Основной причиной такого поведения модели являлось то, что изначально исследования Нордена базировались не на теоретической основе, а на наблюдениях за проектами, причем, в

основном за проектами не связанными с ПО (машиностроение, строительство). Поэтому нет научного подтверждения тому, что программные проекты требуют такого же распределения рабочей силы, наоборот, зачастую количество человеко-часов, требуемых проектом, может резко изменится, сделав оценку непригодной к использованию. После ряда эмпирических наблюдений, Путнэм выразил рабочее уравнение модели в форме: $Size = C \cdot E^{1/3} \cdot t^{4/3}$, где *Size* – размер кода в LOC, *C* – технологический фактор; *E* – общая стоимость проекта в человеко-годах; *t* – ожидаемое время реализации проекта. Технологический фактор включает в себя характеристику проекта в следующих аспектах: методы управления и понимание процесса, качество используемых методов инженерии ПО, уровень используемых языков программирования, уровень развития среды, навыки и опыт команды разработчиков, сложность приложения.

Уравнение для E, выглядит как $E = D_0 \times t_d^3$, где D_0 – коэффициент, выражающий количество необходимой работы (значения от 8 до 12, означает ПО полностью новое, с большим количеством связей; значения до 27 – требуется переработка существующего кода). Связывая два уравнения, получаем уравнение $E = \left(D_0^{4/7} \quad C^{9/7} \right) \quad S^{9/7}$ и $t_d = \left(D_0^{-1/7} \cdot E^{-3/7} \right) \cdot S^{-3/7}$, которые показывают, что затраты пропорциональны размеру кода в степени $9/7 \approx 1/286$. Это достаточно близко к модели Б. Боэма, где данный фактор лежит в пределах от 1.05 до 1.20 [10].

В 1991 году Путнэмом была представлена альтернативная реализация модели, выполненная по заказу Quantitative Software Management (QSM) Inc. и примененная в комплексе SLIM Estimate для оценки стоимости ПО [14]. Полное уравнение в этой реализации выглядит как: $E = 12^5 \quad B \quad \left(SLOC/P \right)^3 \quad \left(1/Schedule^4 \right)$. Если на общее время реализации проекта ограничения не накладываются, то возможно использование упрощенного уравнения $E = 56.4 \quad B \quad \left(SLOC/P \right)^{9/7}$. Здесь *B* – фактор специальных навыков; *P* – фактор продуктивности; Schedule – время разработки по графику (в месяцах). Уравнение может быть использовано, если предполагаемые затраты больше 20 человеко-месяцев.

Использование приведенных уравнений требует значения параметра *P*. Для его определения используется специальная таблица, содержащая значения параметра *P*, зависящие от среды применения разрабатываемого приложения.

Модель COCOMO. Семейство моделей COCOMO было создано в 1981 году на основе базы данных о проектах консалтинговой фирмы TRW [10].

COCOMO представляет собой три модели, ориентированные на использование в трех фазах жизненного цикла ПО: базовая (Basic) применяется на этапе выработки спецификаций; требований расширенная (Intermediate) – после определения требований к ПО; Advanced – углубленная используется после окончания проектирования ПО. В общем виде, уравнение моделей имеет вид $E = a \cdot S^b \times EAF$, где E – затраты труда на проект (в человеко-месяцах); S – размер кода (в KLOC); EAF – фактор уточнения затрат (effort adjustment factor). Параметры *a* и *b* зависят от вида разрабатываемого приложения, который может быть следующим:

- относительно простой проект, работа над которым ведется однородной командой разработчиков, требования носят рекомендательный характер, отсутствует заранее выработанная исчерпывающая спецификация (например, несложное прикладное ПО);
- проект средней сложности, работа над которым ведется смешанной командой разработчиков, требования к проекту определяются спецификацией, однако могут изменяться в процессе разработки проекта (например, ПО системы управления банковским терминалом);
- проект, который должен быть реализован в жестких рамках заданных требований (например, ПО системы управления полетами).

В базовой модели фактор EAF принимается равным единице. Для определения значения этого фактора в расширенной модели используется таблица, содержащая ряд параметров определяющих стоимость проекта. При использовании углубленной модели, вначале проводится оценка с использованием расширенной модели на уровне компонента, после чего каждый параметр стоимости оценивается для всех фаз ЖЦ ПО [10].

COCOMO II также является семейством моделей и представляет собой развитие базовой (Basic) модели COCOMO. COCOMO II включает три модели – создания приложений Application Composition Model (ACM), раннего этапа разработки Early Design Model (EDM) и пост-архитектурная Post Architecture Model (PAM).

ACM используется на раннем этапе реализации проекта, для того, чтобы оценить следующее: интерфейс пользователя, взаимодействие с системой, производительность. За начальный размер принимается количество экранов, отчетов и 3GL – компонентов. Если предположить, что в проекте будет использовано r % объектов из ранее созданных проектов, количество новых объектных точек в проекте Object Points (OP) можно рассчитать, как

$$OP = (\text{object points}) \times (100 - r)/100.$$

Тогда затраты можно вычислить по формуле:

$$E = OP \Big/ PROD,$$

где PROD – табличное значение [10].

EDM – это высокоуровневая модель, которой требуется сравнительно небольшое количество исходных параметров. Она предназначена для оценки целесообразности использования тех или иных аппаратных и программных средств в процессе разработки проекта. Для определения размера используется неприведенная функциональная точка (Unadjusted Function Point). Для ее преобразования в LOC используются данные из таблицы [10]**.** Уравнение модели раннего этапа разработки имеет вид $E = a \quad LOC \quad EAF$ [10], где a – константа 2.45. EAF определятся так же, как и в оригинальной модели COCOMO. Параметры для EDM получаются комбинированием параметров для пост-архитектурной модели.

PAM – наиболее детализированная модель, которая используется, когда проект полностью готов к разработке. Для оценки стоимости ПО с помощью PAM необходим пакет описания жизненного цикла проекта [20], который содержит подробную информацию о факторах стоимости и позволяет провести более точную оценку. PAM используется на этапе фактической разработки и поддержки проекта. Для оценки размеров могут использоваться как строки кода, так и функциональные точки с модификаторами, учитывающими повторное использование кода. Модель использует 17 факторов стоимости и 5 факторов, определяющих масштаб проекта (в модели COCOMO масштаб определялся параметрами вида приложения). Уравнение PAM имеет вид $E = a \quad LOC^{b} \quad EAF$,где a принято за 2.55, а $b = 1.01 + 0.01 \quad \Sigma W_i$, где W_i – параметры, отражающие свойства проекта, например, схожесть с ранее выполненными проектами, риск выбора архитектуры для реализации, понимание процесса разработки, сработанность команды разработчиков. Значения параметров являются табличными [11].

3. Средства оценки стоимости ПО

Широко известны средства оценки ПО, основанные на моделях SLIM и COCOMO [11 – 15].

SLIM Estimate компании QSM наиболее часто используемым программным средством для оценки стоимости ПО, в котором реализована модель Путнэма, является продукт. Средство входит в состав пакета прикладного ПО и предназначено для работы над проектом ПО на начальных стадиях жизненного цикла. В пакет, кроме средства оценки, также входят средства сбора и хранения данных о реализованных проектах (SLIM DataManager), анализа этих данных (SLIM Metrics), общего контроля над процессом разработки (SLIM Control). Данный пакет используется для оценки стоимости разрабатываемого ПО в следующих организациях: Alcatel Telecom, AT&T, Athens Group, Australian Department of Defence, BAE, Bell South Communications, Hewlett-Packard, IBM Rational Software, Lockheed Martin , Motorola Communications, Nokia, US Air Force Cost Analysis Agency. SLIM Estimate позволяет выполнять оценку стоимости разработки программного обеспечения различными способами: мастер быстрой оценки, оценка размера, оценка PI, оценка непредвиденных обстоятельств, оценка основанная на исторических факторах. Первым и наиболее часто используемым является использование мастера быстрой оценки (Quick Estimate Wizard). При этом используются следующие параметры: тип разрабатываемого приложения; максимально возможное время работы над проектом; бюджет проекта; ориентировочное общее количество строк; индекс продуктивности команды разработчиков; процент повторно используемого кода. Формируются таблицы и строятся диаграммы, отображающие общее количество задействованной рабочей силы и ее распределение по графику работ. Шаблон рабочей книги (workbook) проекта SLIM Estimate поддерживает около 50 различных форматов представления проведенной оценки ПО. Созданные рабочие книги могут служить шаблонами для оценки стоимости последующих проектов. По умолчанию, SLIM Estimate оценивает трудозатраты с 50 % вероятностью успешной реализации проекта, для изменения этого значения следует откорректировать значение вероятности с помощью мастера настройки вероятности [12]. Результатом оценки размера является общее количество строк кода, которое может создать команда разработчиков в данных условиях. Результат оценки индекса продуктивности представляет собой PI, необходимый для реализации проекта в заданных условиях. Оценка непредвиденных обстоятельств используется для генерации плана реализации с заданной вероятностью успешного завершения проекта. Эти способы могут использоваться как независимо, так и для уточнения результатов, полученных в результате использования мастера быстрой оценки.

С помощью функции Edit Historical Projects данные оценки могут быть экспортированы в SLIM DataManager. Для сравнительного анализа результатов оценки возможен импорт данных из программы SLIM Metrics, либо другой рабочей книги SLIM Estimate .

Для оценки размера проекта вместе с SLIM Estimate поставляется реализованная в Microsoft Excel таблица, значения из которой могут быть импортированы в рабочую книгу проекта. В ранних версиях SLIM Estimate основной единицей измерения было логическое выражение в исходном коде Logical Source Statement (LSS). Начиная с версии 5.0, в SLIM Estimate используется строки кода, функциональные и объектные точки (напрямую, без преобразования в LSS). Наиболее широко используемым способом калибровки [20] модели в SLIM Estimate является использование исторических параметров настройки (Historical Tuning Factors). Программный комплекс SLIM Estimate может экспортировать данные отчетов в наиболее популярные форматы файлов, такие, как Microsoft Word, Microsoft Excel, Enhanced Metafile, Microsoft Project, HTML.

В комплект поставки SLIM Estimate входит база реализованных проектров, которую можно использовать для калибровки используемой модели – установки значений параметров стоимости для описания характеристик проекта. Наиболее широко используемым способом калибровки модели в SLIM Estimate является использование исторических параметров настройки (Historical Tuning Factors). В случае его использования значения параметров стоимости для проекта вычисляются программным комплексом на основе выбранных проектов из базы реализованных проектов.

Модель Путнэма чрезвычайно чувствительна к значению технологических факторов, поэтому точное определение их значения является очень важным для правильной оценки на основе SLIM. Преимуществом модели Путнэма перед COCOMO 1.1 или COCOMO 2.0, является небольшое количество параметров, необходимых для оценки.

Средства оценки стоимости разработки ПО, основанные на модели SLIM не требуют обязательного использования исторической базы проектов. Поэтому они могут применяться непосредственно организацией, выполняющей проектирование ПО. При использовании исторических баз данных требуется участие специалиста для сопоставления реализованных и описанных проектов из базы с проектом, находящимся в разработке. Привлечение сторонней организации при выполнении оценки стоимости также может быть необходимо по причине наличия у нее достаточно большой исторической и детализированной базы реализованных проектов.

Costar (SoftStar Systems), Cost Xpert (Marotz), SoftwareCost Calculator (SoftwareCost.com). Средства, основанные на модели COCOMO [14]. Допускается использование всех реализаций модели COCOMO, моделей жизненного цикла ПО Waterfall и MBASE/RUP, поддерживается работа с проектом, составленным из компонентов, для каждого из которых можно выполнить раздельную оценку.

Costar позволяет проводить оценку в двух режимах: пошаговом, с помощью мастера оценки стоимости; интерактивном, обеспечивающим непосредственное указание значений параметров, влияющих на стоимость проекта. Для определения размера оцениваемого проекта используются функциональные точки либо строки кода. Для перевода значений, указанных в строках кода, в программе есть конвертатор, рассчитывающий значение размера кода в функциональных точках исходя из языка программирования, который используется для реализации проекта. Costar поддерживает ограничения проекта, основанные на предельных финансовых затратах и крайнем сроке реализации проекта.

Для оценки затрат, связанных с оплатой труда работников существует два альтернативных подхода: расчет затрат для каждого из этапов жизненного цикла ПО; расчет месячной оплаты труда для каждой категории сотрудников.

Для анализа результатов оценки Costar создает различные формы отчетов, графиков и диаграмм. Отчеты, представленные в форме таблиц, могут быть сохранены в формате Microsoft Excel, графики и диаграммы – в формате растрового изображения BMP.

Для проведения точной оценки стоимости разработки ПО модель COCOMO требует детального и разностороннего описания проекта. Это может затруднить применение основанных на ней средств на раннем этапе разработки ПО, и способствует увеличению точности оценки на поздних этапах разработки ПО, при анализе завершенного проекта.

При использовании средств на основе модели COCOMO или COCOMO II факторами, влияющим на точность оценки стоимости являются следующие: правильный выбор конкретной реализации модели COCOMO; точность калибровки – соответствие установок исходным данным. В связи с этим, для применения средств используют персонал, который не имеет прямого отношения к процессам проектирования и разработки ПО. Он формирует спецификации проекта и параметры, необходимые для оценки, которые предоставляются сотрудникам, которые выполняют оценку.

Эффективное применение алгоритмических моделей оценки стоимости ПО и основанных на них средств оценки предпочитают их совместное использование с неалгоритмическими методами оценки. Так, алгоритмические средства оценки могут быть применены членами экспертных комиссий для анализа проекта и формирования собственной оценки. Благодаря широким возможностям экспорта данных и визуализации, использование автоматизированных средств оценки стоимости ПО дает возможность формировать собственные базы характеристик реализованных проектов, а также создавать отчеты, иллюстрирующие процесс разработки проекта, что значительно снижает трудозатраты, связанные с подготовкой отчетности.

Параметры стоимости. Параметр стоимости (cost driver) – это субъективная величина, которая оценивает различные временные, качественные и ресурсные аспекты разработки ПО. Каждый из параметров может быть откалиброван. Калибровка параметров стоимости – это корректировка значений параметров, которая влияет на значение трудозатрат, и следовательно на время и стоимость, при оценке программного проекта. При калибровке указанных далее семнадцати параметров выбирается оценочный уровень (очень высокий, высокий, выше номинального, номинальный, ниже номинального, низкий, очень низкий) параметра. В формулах этот уровень отражается в виде коэффициента трудозатрат и, таким образом, на каждой стадии разработки проекта влияет на стоимость и длительность той или иной стадии. Выделяют следующие группы параметров [14] (см. табл.1): продукта (product factors), платформы (platform factors), персонала (personnel factors) и проекта (project factors). В табл. 2 дано краткое описание каждого параметра.

Таблица 1

Параметры	Описание
Продукта	Учитывают характеристики разрабатываемого ПО. (RELY, DATA, CPLX, RUSE, DOCU)
Платформы	Учитывают характеристики программно-аппаратного комплекса, требуемого для функционирования ПО. (TIME, STOR, PVOL)
Персонала	Учитывают уровень знаний и слаженности работы коллектива программистов. (ACAP, PCAP, PCON, APEX, PLEX, LTEX)
Проекта	Учитывают влияние современных подходов и технологий, территориальной удаленности членов коллектива разработчиков и сроки выполнения проекта. (TOOL, SITE, SCED)

Таблица 2

Параметры	Описание
RELY (Required Software Reliability)	Учитывает меру выполнения программой задуманного действия в течение определенного времени
DATA (Database Size)	Учитывает влияние объёма тестовых данных на разработку продукта. Уровень этого параметра рассчитывается как соотношение байт в тестируемой базе данных к SLOC в программе
CPLX (Product Complexity)	Включает пять типов операций: управления, счетные, устройство-зависимые, управления данными, управления пользовательским интерфейсом. Уровень сложности это субъективное средне-взвешенное значение уровней типов операций
RUSE (Developed for Reusability)	Учитывает трудозатраты, требуемые дополнительно для написания компонентов, предназначенных для повторного использования в данном или последующих проектах. Использует следующие оценочные уровни: “в проекте”, “в программе”, “в линейке продуктов”, “в различных линейках продуктов”. Значение параметра накладывает ограничения на следующие параметры: RELY и DOCU
DOCU (Documentation Match To Life-Cycle Needs)	Учитывает степень соответствия документации проекта его жизненному циклу
TIME (Execution Time Constraint)	Учитывает временные ресурсы, используемые ПО, при выполнении поставленной задачи
STOR (Main Storage Constraint)	Учитывает процент использования хранилищ данных
PVOL (Platform Volatility)	Учитывает срок жизни платформы (комплекс аппаратного и программного обеспечения, который требуется для функционирования разрабатываемого ПО)
ACAP (Analyst Capability)	Учитывает анализ, способность проектировать, эффективность и коммуникативные способности группы специалистов, которые разрабатывают требования и спецификации проекта. Параметр не должен оценивать уровень квалификации отдельно взятого специалиста
PCAP (Programmer Capability)	Учитывает уровень программистов в коллективе. При выборе значения для этого параметра следует особо обратить внимание на коммуникативные и профессиональные способности программистов и на командную работу в целом
PCON (Personnel Continuity)	Учитывает текучесть кадров в коллективе
APEX (Applications Experience)	Учитывает опыт коллектива при работе над приложениями определенного типа
PLEX (Platform Experience)	Учитывает умение использовать особенности платформ, такие как графический интерфейс, базы данных, сетевой интерфейс, распределенные системы
LTEX (Language and Tool Experience)	Учитывает опыт программистов (языки, среды и инструменты)
TOOL (Use Of Software Tools)	Учитывает уровень использования инструментов разработки
SITE (Multisite Development)	Учитывает территориальную удаленность (от офиса до международных офисов) членов команды разработчиков и используемые ими средства коммуникации (от телефона до видео конференц-связи)
SCED (Required Development Schedule)	Учитывает влияние временных ограничений, накладываемых на проект и на значение трудозатрат

4. Практическое применение средств оценки стоимости ПО

Costar был использован в эксперименте для оценки стоимости разработки ПО системы дистанционного обучения [21, 22]. Оценка ПО производилась путем задания двух характеристик будущей системы: прогнозируемое число строк кода (SLOC) и уровень определенности архитектуры. Также была произведена калибровка следующих параметров стоимости: CPLX, APEX, PLEX, LTEX, PCON, PVOL, RELY, DOCU, TOOL, SITE. Значения параметров, характеризующих проект, выбирались в соответствии с опытом [16 – 20] и следующими особенностями [22]: значение параметра PLEX было выбрано High, поскольку коллектив программистов знаком с платформой, для которой разрабатывается проект и поэтому не требуется дополнительного времени на их обучение. Значение параметра TOOL – Very Low, поскольку при написании кода использовались простые средства разработки, без использования комплексных интегрированных сред. Аналогично калибровались все остальные параметры стоимости.

Для указанного проекта был получен следующий результат: общее время разработки – 8,4 месяцев, требуемые ресурсы для разработки проекта – 8,7 человеко-месяцев, общая стоимость проекта – $700. Такие данные были получены для номинального графика работы. Нужно отметить, что низкая стоимость проекта обусловлена прежде всего тем, что проект выполнялся студентами. Реальная же стоимость ПО значительно выше. Например, при уменьшении времени разработки до 6,3 месяцев ресурсы вырастают до 12,4 человеко-месяцев, а цена проекта повышается до $1000. Кроме того, в результате были получены следующие временные оценки для фаз жизненного цикла проекта: специфицирование требований – 0,8 мес., проектирование – 1,4 мес., детальное проектирование – 2,2 мес., программирование и тестирование – 2,9 мес. Результат сравнения расчетных показателей при номинальном графике и показателей, полученных после завершения проекта, показал, что расхождение времени разработки составило не более 6 %, а стоимости – не более 10 %. На выполнение указанных оценок и оформление результатов было затрачено 3 часа. Для ручного расчета потребовались бы недели, а риск ошибки был бы очевидно выше.

Таким образом, эксперимент показал возможность использования средств оценки стоимости ПО для отечественных проектов, с целью обеспечения значительного сокращения расходов на разработку больших программных систем и оптимизации работ (рисунок).

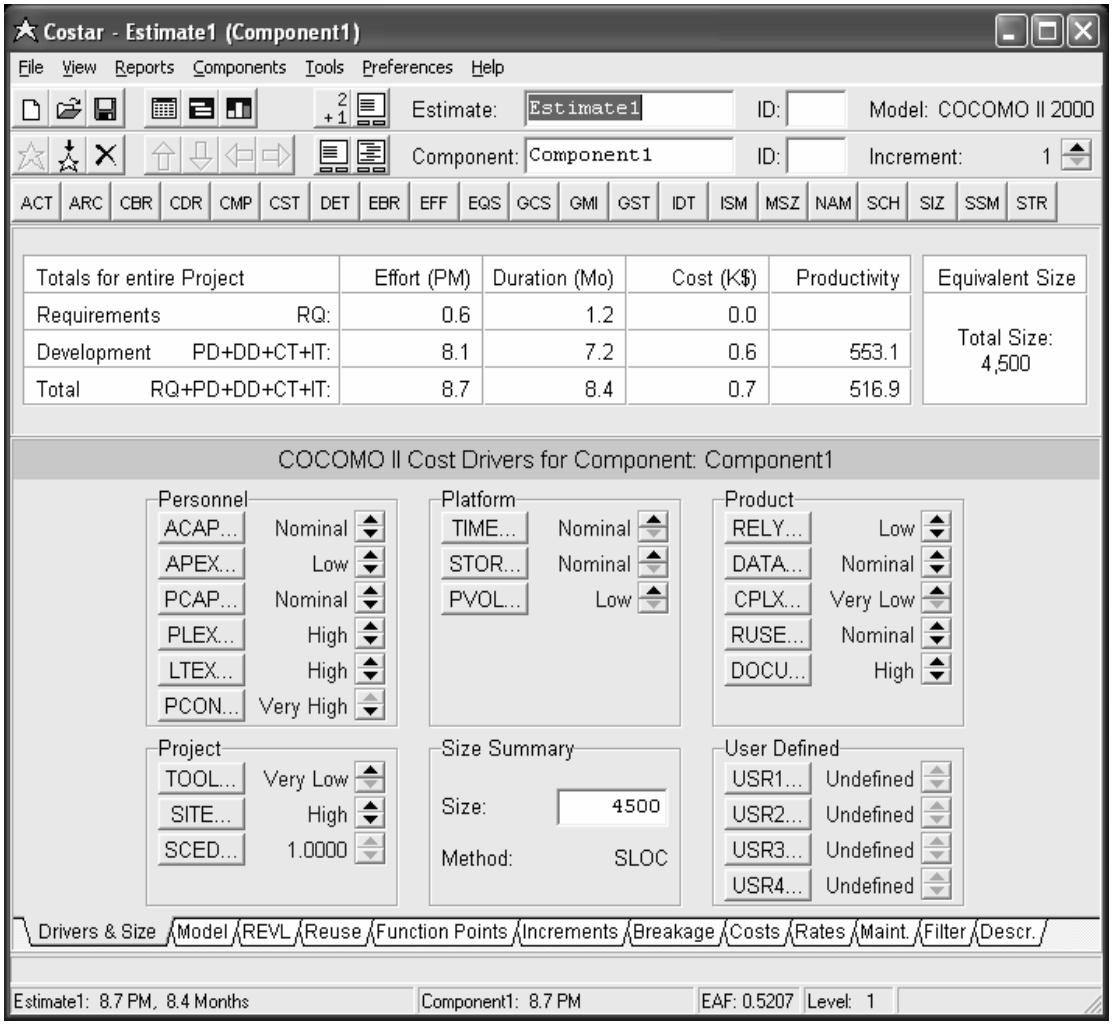

Рисунок. Основное окно программы Costar

Выводы

Оценка стоимости ПО сложный и ответственный процесс, требующий глубоких теоретических знаний, практического опыта, соответствующих инструментов. В Украине, на данный момент, отсутствуют специалисты, которые в состоянии произвести оценку ПО. Поэтому актуальным является, с одной стороны подготовка высококвалифицированных кадров, способных производить оценку, а с другой – создание и постоянное обновление базы данных со статистической информацией по разработанным программным проектам.

1. *Boehm B.W.* Software engineering economics. – Prentice-Hall. – 1981. – 320 p.
2. *Albrecht A.J., Gaffney J.E.* Software function, source lines of codes, and development effort prediction: a software science validation. – IEEE Trans Software Eng. – 1983. – 4 p.
3. *Software* engineering: IFPUG 4.1 Unadjusted functional size measurement method: Counting practices manual.– ISO/IEC.– 2003. – 430 p.
4. *Longstreet D.* Function Point Analysis Training Course. – Longstreet Consulting Inc. – 2004. – 280 p.
5. *Fenton N.E., Pfleeger S.L.* Software Metrics: A Rigorous and Practical Approach. – PWS Publishing Company. – 1997. – 455 p.
6. *Parkinson S.N.* Parkinson's Law and Other Studies in Administration. – Houghton-Miffin. – 1957.– 148 p.
7. *Coates J.* Technological Forecasting and Social Change. – Elsevier Science Inc. – 1999. – 235 p.
8. *Shepperd M., C. Schofield* Estimating software project effort using analogy. – IEEE Trans Software Eng. – 1997. – P. 736 – 743.
9. *David L.* Norden-Raleigh Analysis: A Useful Tool for EVM in Development projects. – The Measurable News. – 2002. – 24 p.
10. *Johnson Kim.* Software cost estimation – Metrics and models. – University of Calgary. – 2001. – 115 p.
11. *McGibbon* Th. Modern Empirical and Schedule Estimation Tools. – DACS Report. – 1997. – 72 p.
12. *Heires J., Doing T.* More with Less: SLIM-Estimate 5.0 Product Review. – QSM Software –2002. – 46 p.
13. *Aron J.D.* Estimating Resource for Large Programming Systems. – NATO Science Committee. – 1969. – 40 p.
14. *Boehm B.W.* The COCOMO 2.0 Software Cost Estimation Model. – American Programmer. – 2000. – 586 p.
15. *Capers J.* Applied Software Measurement: Assuring Productivity and Quality. – McGraw-Hill. – 1996. – 590 p.
16. *Сидоров Н.А.* Утилизация программного обеспечения, экономический аспект Кибернетика и системный анализ. – 1994. – № 3. – С 151-167.
17. *Василенко Ю.Н.* Алгоритмические методы оценки программного обеспечения. – Мат. конф. "Інженерія програмного забезпечення 2005". – НАУ. – 2005. – С. 42 – 51.
18. *Баценко Д.В.* Классификация параметров стоимости модели постархитектуры. – Мат. конф. "Інженерія програмного забезпечення 2005". – НАУ. – 2005. – С. 51 – 56.
19. *Щебетин Ю.В.* Методика анализа функциональных точек программного обеспечения. – Мат. конф. "Інженерія програмного забезпечення 2005". – НАУ. – 2005. – С. 56 – 62
20. *Сидоров Н.А., Баценко Д.В., Василенко Ю.Н., Щебетин Ю.В., Иванова Л.Н.* Методы и средства оценки стоимости программного обеспечения. "Проблеми системного підходу в економіці". – НАУ. – 2004. – № 7 – С. 113 – 118.
21. *Сидоров Н.А., Лидер Д.А., Хальдун Д.*. Дескриптивная модель дистанционного завершения высшего образования в области информационных технологий // Проблеми системного підходу в економіці. – НАУ. – 2003. – № 5 – С 6 – 10.
22. *Д.А. Лидер, Дакак Хальдун.* Портал для завершения высшего образования в области информационных технологий. – Мат. конф. "УкрПрог-2004". – 2004. – С. 605 – 612.