ется мощным инструментом для коммуникации между разработчиками и менеджерами, создавая единую систему понятий.

Собственно говоря, проектировщики являются связующим звеном между разработчиками и управленцами (менеджерами) так, как двое последних преследуют разные интересы и обладают разными ценностями. Для руководителей важным является жизнеспособность продукта и прибыль, а для программистов – технологии, нефункциональные требования, сложные архитектурные решения, нефункциональные требования, которые пользователь, к сожалению, не всегда может оценить в связи с когнитивным сопротивлением (вероятность которого возрастает без проектирования).

Преимущества проектирования, предшествующего написанию кода и являющегося неотъемлемой частью процесса разработки, очевидны. Его результатом является продукт, осуществление которого реально, и значительное увеличение вероятности, как на финансовый успех продукта, так и на количество счастливых пользователей.

Литература

1 Alan Cooper and Robert Reimann, About Face 2.0: The Essentials of Interaction Design: Published by John Wiley & Sons, 2003, 576 pp.

2 Alan Cooper, The inmates are Running the Asylum , Indianapolis: SAMS, 1999. 261 pages.

3 Jaime Levy, UX Strategy, 2015.

А.А. Горовик, М.В. Лазарева

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКОГО АЛГОРИТМА ПРИ ОЦЕНКЕ СЛОЖНОСТИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

(Ферганский филиал Ташкентского университета информационных технологий)

Аннотация: Основной проблемой современных методов оценки трудозатрат является сложность их адаптации к каждому конкретному проекту. В статье предложен принципиально новый, неклассический генетический подход к этой проблеме, главные принципы которого — обеспечение качественной и количественный оценки трудозатратности разработки программных проектов

Abstract The main problem of modern methods of evaluation of labor is the difficulty of adapting them to each specific project. This paper proposes is a fundamentally new, non-classical genetic approach to the problem, the main principles of which is providing qualitative and quantitative assessment of labor-intensive software development projects

Точный расчет ресурсов является одной из основных проблем в области управления проектами. При таком расчете возникают сложности учета огромного количества факторов, влияющих на жизненный цикл программного обес-

печения. Как следствие, на сегодняшний день многие компании сталкиваются с серьезными проблемами в случае неправильных расчетов необходимых сроков:

6. при недооценке — непредвиденная трата дополнительных средств, недовольство заказчика невыполнением обязательств в срок, «бессонные ночи» сотрудников, сложность управления «обвальным» проектом, низкое качество конечного продукта, слаборазвитые функции системы и т.д;

7. при переоценке — бесполезный расход ресурсов, привлеченных к проекту, отказ заказчика от контракта с данными условиями (в результате возможна потеря рабочих мест) и т.д.

На современном рынке крупных программных систем потери могут исчисляться миллионами долларов.

Точные оценки издержек производства программного обеспечения важны как разработчикам, так и заказчику (клиенту). Они могут использоваться при переговорах о контракте, планировании, контроле и т. п. Таким образом, возникает реальная потребность в разработке методов и средств, позволяющих менеджеру оценить требуемые временные и человеческие ресурсы на основе всех имеющихся характеристик проекта: истории предыдущих подобных проектов, опыта и производительности сотрудников, специфики компании и т. п. Кроме того, требуется возможность перерасчета и уточнения сроков и ресурсов уже на этапе разработки системы с учетом текущих тенденций, наблюдаемых при реализации проекта. Это поможет менеджеру своевременно обнаружить отклонения от установленного графика и принять соответствующие меры в управлении проектом.

Критерии хорошей оценки

По Ройсу [2], хорошая оценка издержек производства программного обеспечения должна быть:

1. понятной и поддержанной менеджером проекта и командой разработчиков;

2. одобреной всеми заинтересованными лицами как реально осуществимая;

3. базирующейся на четкой модели с заслуживающими доверия основаниями, а также на данных подобного проекта (с подобными бизнес-процессами, технологиями, внешней средой, людьми и требованиями);

4. определена настолько детально, что ключевые области риска понятны, и вероятность успеха объективно оценена.

Сравнение метрик размера программного обеспечения

Размер программного обеспечения — наиболее важный фактор, определяющий трудоемкость реализации. Число строк исходного кода и функциональные точки — самые популярные метрики.

Число строк кода

LOC (LinesOfCode) — число непустых строк исходного текста, исключая комментарии [4]. Несмотря на то, что эта метрика существенно зависит от выбранного языка программирования, она до сих пор остается самой используемой метрикой размера программного обеспечения. Точное число LOC может

быть получено только после того, как проект уже закончен. Поэтому оценка размера кода программы до его создания не намного проще оценки реальных трудозатрат, например, в человеко-месяцах.

Типичный метод осуществления такой оценки использует комбинацию экспертных оценок с техникой под названием PERT, заключающейся в следующем:

пусть имеется n экспертов, каждый i-й эксперт высказывает три предположения относительно конечного размера: L_i — нижняя оценка размера; H_i- — верхняя оценка размера; M_i — наиболее вероятный размер. Тогда размер S можно вычислить как

$$s = \frac{1}{n}\sum_{i=1}^{n}\frac{L_i + H_i + 4M_i}{6}. \quad (1)$$

Точность оценки может быть существенно улучшена, если применить PERT не к проекту в целом, а к его отдельным компонентам. В этом случае общую оценку размера можно получить как сумму «локальных» оценок.

Метрики Холстеда

М. Холстедом были предложены такие метрики, как длина кода и объем [6]. Длина кода определяется как

$$N = N_1 + N_2, \quad (2)$$

где N_1 — общее количество появлений операторов в программе; N_2 — общее количество их операндов.

Объем кода соответствует объему памяти, требуемой для хранения программы, вычисляется по формуле:

$$V = N\log(n_1 + n_2), \quad (3)$$

где n_1 — число различных операторов; n_2 — число различных операндов, появляющихся в программе.

Очевидно, что оценить общее число операторов и их операндов до завершения проекта, как правило, еще более затруднительно, чем оценить LOC, поэтому в адрес предложенных Холстедом метрик было высказано множество замечаний. Поддержка такого подхода в последние годы постоянно уменьшается.

Функциональные точки (Function points)

Наиболее удачной заменой количеству строк кода для измерения размера стали функциональные точки (functionpoints), впервые предложенные сотрудником IBM A. Альбрехтом в 1979 г. [7]. Применение функциональных точек основано на оценке объема реализуемой функциональности за счет изучения требований, вследствие чего оценка необходимых трудозатрат может быть выполнена на самых ранних стадиях работы над проектом и далее будет уточняться по ходу жизненного цикла, а явная связь между требованиями к создаваемой системе и получаемой оценкой позволяет заказчику понять, за что именно он платит, и во что выльется изменение первоначального задания.

Автор кратко рассмотрит основные принципы данного метода. Общее число функциональных точек программы зависит от количества элементарных процессов пяти типов:

1) входящие транзакции (Externalinputs (El)) — получают данные от пользователя;

2) исходящие транзакции (Externaloutputs (EO)) — передают данные пользователю;

3) взаимодействия с пользователем (Externalinquiries (EQ)) — интерактивные диалоги с пользователем (требующие от него каких-либо действий);

4) файлы внутренней логики (Internallogicalfiles (ILF)) — файлы (логические группы информации), использующиеся во внутренних взаимодействиях системы;

5) файлы внешних взаимодействий (Externalinterfacefilese (EIF)) — участвуют во внешних взаимодействиях с другими системами.

В данной терминологии транзакция — элементарный неделимый замкнутый процесс, имеющий значение для пользователя и переводящий продукт из одного консистентного состояния в другое.

Базовая модель COCOMO. Размер проекта измеряется в LOC (KLOC), а трудозатраты — в человеко-месяцах. Создана на основе анализа статистических данных 63 проектов (главным образом, Министерства обороны США) различных типов.

Используются три набора параметров {a, to} в зависимости от сложности разрабатываемого программного обеспечения:

- для простых, легко понимаемых проектов, a=2,4, b=1,05;
- для сложных систем, a=3,0, b=1,15;
- для встроенных систем, a=3,6, b=1,20.

Модель была проста в применении, но не обеспечивала должной точности.

Полученные результаты

Таблица 1

Проект	Имитируемая дата осуществления предсказания	Фактическая дата релиза	COCOMO II	COCOMO II + Genetic	Увеличение точности, %
Firefox49.0.2	15.08.2017	20.10.2017	24.10.2017	26.10.2017	2,7
Fedora 25	10.08.2017	11.10.2017	3.10.2017	14.10.2017	8,1
KDE5.7.2	4.06.2017	19.07.2017	28.07.2017	29.07.2017	1,3

Описанные идеи позволили разработать прототип системы уточнения оценок трудозатрат, основанных на модели COCOMO II. По базам данных багтрекеров трех opensource- проектов (Firefox, Fedora, KDE) произведены оценки, которые в среднем оказались на 4% точнее, чем оценки, полученные с

помощью COCOMO II, без уточнения посредством рассмотренного метода (табл. 1). В дальнейшем планируется дополнить разрабатываемую систему упрощенными версиями существующих методов, что сделает возможным ее использование как полноценного самостоятельного программного пакета. Увеличение точности оценки для некоторых opensource-проектов

Литература

1. Boehm B. Software Cost Estimation with Cocomo II. NewJersey, Prentice-Hall, 1981.
2. Royce W. Software project management: a unified framework. Reading, Addison-Wesley Professional, 1998.
3. Boehm B. Software engineering economics. New Jersey, Prentice-Hall, 1981.
4. Fenton N. Software Metrics: A Rigorous and Practical Approach, London, Chapman and Hall, 1991.
5. Parkinson G. Parkinson's Law and Other Studies in Administration NY, Houghton-Miffin, 1962.
6. Halstead M. Elements of software science, NY, Elsevier, 1977.
7. 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. P 639-648.

Ю.Н. Горелов, С.Б. Данилов, Л.В. Курганская

ОБ ОПТИМАЛЬНОМ РАСПРЕДЕЛЕНИИ ФИЗИЧЕСКОГО РЕСУРСА В СИСТЕМЕ НЕЗАВИСИМЫХ ПРОЦЕССОВ УПРАВЛЕНИЯ

(Самарский университет)

Физические ограничения играют существеную роль в задачах управления, в особенности, когда возникает необходимость распределения каких-либо «ресурсов управления» между одновременно протекающими независимыми процессами управления [1-3]. В связи с этим целью настоящего доклада общая постановка задачи распределения некоторого физического ресурса управления (например, энергетического – в виде «энергии управления» или материального ресурса – в виде расходов «топлива») между независимыми системами.

Рассмотрим двухточечную граничную задачу управления для процесса

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{u}), \qquad (1)$$

где $\boldsymbol{x} \in \mathbb{R}^n$ – вектор переменных его состояния, $\boldsymbol{u} \in \mathbb{R}^m$ – вектор управляющих параметров, $\boldsymbol{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$, а в качестве целевых условий заданы граничные условия общего вида:

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0;\ \boldsymbol{x}(t_f) = \boldsymbol{x}_f, \qquad (2)$$