УДК 004.05
DOI: 10.30987/conferencearticle_5c19e5ed9dde81.14689097

М.В. Лазарева, А.А. Горовик
(г. Фергана, Ферганский филиал Ташкентского университета информационных технологий им. Мухаммада ал-Хоразмий)

АНАЛИЗ МЕТОДОВ КАЧЕСТВЕННОЙ ОЦЕНКИ СЛОЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Проанализированы используемые методы качественной оценки сложности программного обеспечения.

The article analyzes the methods used for qualitative assessment of software complexity.

Ключевые слова: трудозатраты, сложность, точность, функциональные точки, метрики, объем программного кода, внешние входы, внешние выходы.

Keywords: labor, complexity, accuracy, function points, metrics, code volume, external inputs, external outputs.

Расчет трудозатрат программного обеспечения является актуальной проблемой. Для получения точной оценки сложности составления программы необходимо учесть большое количество факторов, связанных с жизненным циклом создаваемой программы. При неправильно произведенном расчете могут возникнуть проблемы, ведущие к ухудшению программы, дополнительным тратам, невыполнению договоров, или к излишним, напрасным затратам. Если учитывать рынки современных программных продуктов и их востребованность, то можно оценить масштаб потерь при неправильном расчете.

На практике и разработчик, и заказчик хотят знать точные расчеты трудозатрат. Поэтому важно использование алгоритмов и средств, точно определяющих все затраты: время, человеческие ресурсы, сложность и точность. Такими данными желательно обладать уже на этапе разработки различных проектов, при этом должны учитываться специфика разрабатываемого программного обеспечения, возможность изменения сроков с учетом такой специфики, вероятность обнаружения раннего отклонения от поставленных сроков.

У. Ройс, имеющий двадцатилетний успешный опыт управления проектами, определяет следующие критерии хорошей оценки создаваемых программ [2]: понятность, одобрение со стороны команды разработчиков, реальность осуществления. В основании такой оценки должна быть четкая модель и данные подобных проектов. В такой оценке должны учитываться области риска и определяться объективная оценка вероятности успеха.

Одним из значительных факторов**,** влияющих на реализацию и связанную с ней трудоемкость, является размер программного обеспечения. Исходный код имеет определенное количество строк и функциональных точек – самых популярных показателей.

Впервые использование таких точек (function points) было рассмотрено сотрудником фирмы IBM Аланом Альбрехтом в 1979 г. Преимущества использования функциональных точек, основанных на изучении требований, как раз и заключается в том, что оценка сложности программного обеспечения выполняется на первоначальных этапах работы над проектом.

LOC (LinesOfCode) — число непустых строк исходного текста, исключая комментарии [4]. Этот показатель в большой степени зависит от языка программирования программы, но в тоже время остается основным показателем размера программного обеспечения. Однако точное число LOC определяется только после окончания проекта. Получить это число можно используя экспертные оценки и метод PERT.

PERT – Program (Project) Evaluation and Review Technique – метод оценки и анализа проектов, который применяется для сложных проектов, учитывает неопределенности, возникающие при разработке, при этом знание деталей проекта не обязательно. Определяется минимальное время, необходимое для разработки проекта, на основе размера кода программного обеспечения проекта.

Метод использует предположения *n* экспертов, каждый i-й эксперт определяет верхнюю границу, нижнюю границу и наиболее вероятный размер. Тогда размер кода программы может быть вычислен как

$$S = \frac{1}{n}\sum_{i=1}^{n}\frac{Li+Hi+4Mi}{6},$$

где S – размер кода программы;

L_i-размер определенной нижней оценки;

H_i- размер определенной верхней оценки;

M_i –вероятный размер.

Если разбить проект на отдельные части и к каждой применить предложенный метод, то точность оценки значительно увеличивается, а общая оценка выводится суммированием полученных показателей.

Морис Ховард Хальстед в 1977 году в своем трактате по разработке программного обеспечения ввел программные метрики [6], которые выражают соотношения между свойствами программного обеспечения.

Метрики Хальстеда основываются на числе операторов и числе операндов.

Пусть

N_1 – общее число операторов в программном коде;

N_2 – число их операндов.

Длину кода можно определить как сумму

$$N= N_1+ N_2 .$$

Если

n_1 – число различных операторов в программном коде,

n_2 – число различных операндов,

тогда объем программного кода вычисляется как

$$V = N \log(n_1 + n_2).$$

Такой объем соответствует объему памяти, необходимой для хранения программного кода.

Метрики Хальстеда подверглись критики, ввиду того, что достаточно трудно посчитать общее число операторов и операндов до окончания проекта. Точно так же, как и в LOC, оно точно определяется только после окончания проекта. Метрики Хальстеда в последнее время потеряли свою актуальность.

Альтернативой расчета размера программного кода стало применение функциональных точек. Изучение всех требований может корректироваться в ходе жизненного цикла. Это даст понять заказчикам сложность, оценить затраты, связанные с изменением изначального задания. В 1986 году была сформирована IFPUG – International Function Point User Group – международная группа пользователей функционального измерения.

Суть данного метода, в зависимости от общего числа функциональных точек, определяется по 5-ти типам элементарных процессов:

1. EI – все входящие транзакции, они принимают данные от пользователя. Это так называемые «внешние входы», причем, считают только те входы, которые оказывают различное влияние.
2. EO – все исходящие транзакции, они представляют данные пользователю. Это так называемые «внешние выходы», причем, считают только выходы для различных алгоритмов. Это может быть выдача сообщения и вызов подпрограммы или функции (в данном случае 2 выхода).
3. EQ – интерактивные действия с пользователем, при этом пользователь должен выполнить какие-либо действия для работы программы.
4. ILF – файлы для внутренней логики, применяющиеся во внутренних действия системы. Это могут быть группы данных, непосредственно создаваемые при работе программы или поддерживаемые во время работы.
5. EIF – файлы для внешних взаимодействий, необходимые для согласованных действий с другими системами. Это могут быть данные, расположенные во внешних файлах.

Значения всех полученных точек умножаются соответственно на свой коэффициент сложности и суммируются.

В результате получается полный размер программного обеспечения. Данная методика постоянно улучшается и пользуется успехом в настоящее время.

Возникло несколько измененных методов функциональных точек. Один из них – метод точек свойств.

Если описанные процессы не имеют сложности, то используется метод точек свойств. Метод предложен Кейперсом Джонсоном и практически повторяет метод функциональных точек.

Чарльз Саймонс разработал метод Mark II.

Он применим к достаточно сложным системам и позволяет считать оценки отдельных частей сложных систем. Причем суммирование полученных оценок совпадает с полученным результатом оценки всей системы в целом.

Метод трехмерных функциональных точек.

Данный метод был предложен в 1991 году корпорацией Boeing. В данном методе сложность разрабатываемого программного обеспечения рассматривается и оценивается в трех различных аспектах: данные, функции, управление. При этом можно оценить не только проект, но и трудоемкость задач.

Метод Демарка.

Том Демарк предложил метод, получивший название метода Демарка. Его метод учитывает данные по выполняемым ранее разработкам. Метод дает точные оценки расходов времени и ресурсов.

Рассматриваемый метод функциональных точек и его модификации имеют значительный недостаток: затрачивается много времени для самого использования и подготовки (изучению). Поэтому предпочтительными в настоящее время являются методы более быстрые и с более точными результатами.

Список литературы

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.

Материал поступил в редколлегию 06.10.18