Министерство образования и науки, молодежи и спорта Украины
Харьковский национальный университет

Ревенчук Илона Анатольевна

Программной инженерии

Раздел 16 - Оценка размера и возможности повторного использования ПО

6.050103 - Программная инженерия, 8.05010301 - Программное обеспечения систем, 8.05010302 - Инженерия программного обеспечения

Харьков
2005

Содержание

Раздел 16 - Оценка размера и возможности повторного использования ПО

В разделе рассматриваются:

- Единицы измерения, используемые при оценке размера ПО.
- Риски, связанные с оценкой размера разрабатываемого ПО.
- Метод оценки показателя LOC с помощью экспертных заключений и восходящего суммирования, его преимущества и недостатки.
- Метод функциональных точек, его преимущества и недостатки, заполнение рабочих листов.
- Метод точек свойств, его преимущества и недостатки, заполнение рабочих листов.
- Метод оценивания Wideband Delphi.
- Метод блиц-модели.

Теория

Введение

Труднее всего выполнить оценивание в начале проекта, когда еще не сформировались достаточно четкие представления о нем.

При оценке стоимости проекта и количества времени, требуемого для его выполнения, следует выполнить два основных этапа.

1. Оценивание размера проекта.

2. Представление об этих размерах наряду с информацией о других факторах среды позволяет оценивать общий объем трудозатрат и, соответственно, стоимость работ. Уточнение размеров создаваемого ПО предшествует этапу кодирования, выполняемого с целью реализации требований на практике.

Путем выполнения оценивания можно спрогнозировать общий объем ресурсов, который необходим для выполнения данного проекта. При этом учитываются затраты времени, количество штатных единиц и бюджетные ограничения.

(Первый этап - раздел 16, второй этап - раздел 17).

Единицы измерения при оценке размера ПО

При использовании единиц измерения возникает масса вопросов.

Каким образом можно определить количество строк кода (Lines of code, LOC) до того, как они фактически будут написаны либо просто спроектированы?

Как показатель количества строк кода может отражать величину трудозатрат, если не будет учитываться сложность производимого продукта, способности/стиль программиста либо возможности применяемого языка программирования?

Каким образом разница в количестве строк кода может быть трансформирована в объем эквивалентной работы?

Эти и другие подобные вопросы приводят к тому, что единицы измерения LOC получают "дурную репутацию", хотя они по-прежнему остаются наиболее широко используемыми.

Взаимосвязь между LOC и затрачиваемыми усилиями не является линейной. Несмотря на появление новых языков программирования, ***средняя производительность программиста*** за последние 20 лет осталась неизменной и составляет около ***3000 строк кода на одного***

программиста в год. Это говорит о том, что уменьшение времени, затрачиваемого на цикл разработки, не может быть достигнуто за счет значительного повышения производительности труда программистов. Причем это не зависит от усовершенствований языка программирования, усилий со стороны менеджеров либо от наличия/отсутствия сверхурочных работ.

На самом деле огромное значение имеет набор функциональных свойств и качество ПО, а не количество строк программного кода.

Примеры единиц измерения при оценке размера ПО

● количество строк кода (Lines Of Code, LOC);

● функциональные точки;

● точки свойств;

● количество "пузырьков" на диаграмме потока данных (Data flow diagram, DFD);

● количество сущностей на диаграмме сущностей (Entity relationship diagram, ERD);

● количество "квадратиков", соответствующих процессу/контролю (PSPEC/CSPEC) на структурном графике;

● количество различных элементов в составе управленческой спецификации;

объем документации; количество объектов, атрибутов и служб на объектной диаграмме.

Рассмотрим оценку размеров программного проекта. В качестве единиц измерения могут выступать самые различные величины. Выбор этих единиц зависит от конкретного проекта и потребностей вашей организации.

Иногда прибегают к сравнительному подсчету количества упоминаний "должно" и "желательно" при изучении документа, содержащего основные требования, учитывают число страниц пользовательской документации, объекты и/или классы объектной модели, а также прибегают к другим вариантам оценок.

Не столь важно, что некоторые термины звучат немного непривычно, поскольку при рассмотрении все же будут учитываться наиболее распространенные определения. Вне зависимости от того, оценивается ли конечный продукт, как в случае с применением показателя LOC, либо некоторая его абстракция или модель, как это происходит с объектами классов, в данном случае оценке подвергается то, чего еще нет в природе. Поэтому оценивание размеров представляет значительные трудности.

Оценивание размера и потенциала повторного использования ПО выполняется на ранних стадиях планирования проекта. Если воспользоваться терминологией из области обобщенной модели жизненного цикла, можно сказать, что планирование осуществляется на этапах

исследования концепций, системы и в фазе формирования требований.

Оценка размера и трудозатрат выполняется неоднократно во время осуществления жизненного цикла, причем после каждого оценивания повышается и уровень доверия к достигнутым результатам. Точность оценки значительно повышается после формулирования начальных требований заказчиков, проведения анализа, после завершения разработки проекта и т.д.

Оценивание размера программного продукта, результат которого затем применяется для оценивания трудозатрат и накладных расходов на этапе разработки проекта, хорошо вписывается в набор компетенций проекта. Оценки размера ПО, скорректированные с целью повторного применения, фиксируются в плане проекта. При оценке размера ПО используется метод измерения, который поддерживается на всем протяжении эволюции проекта. И наконец, данные о размере ПО являются исходными при составлении графика проекта.

Навыки менеджмента проектов

1. Создание структуры пооперационного перечня работ - задачи в рамках выполнения проекта и производства продукта разбиваются на достаточно малые составляющие элементы, размер которых будет затем оцениваться.

2. Документирование планов - разбитые на составляющие элементы задачи в рамках выполнения проекта и производства продукта оцениваются с целью вычисления величины, трудозатрат и накладных расходов. Результаты оценки используются при составлении графика. Все упомянутые выше параметры отображаются в плане управления программными проектами (SPMP); оценки рисков документируются в виде отдельного плана рисков, выступающего в качестве части плана SPMP.

3. Оценка стоимости и трудозатрат, требуемых для завершения проекта - благодаря оценке возможного размера ПО можно судить относительно размера планируемых трудозатрат.

Например, если размер разрабатываемого ПО оценивается величиной в 500 LOC (единица оценки размера), а скорость программирования в среднем равна 5 LOC/час (оценка производительности), то в этом случае трудозатраты составят 100 часов. Если ваша небольшая команда состоит из двух человек, которые могут распределять между собой рабочую нагрузку, трудозатраты каждого из них составят по 50 часов. Оценка трудозатрат влечет за собой оценку возможных накладных расходов. Если накладные расходы (стоимость) для первого разработчика составят 65 долларов в час, а накладные расходы для другого разработчика - 55 долларов в час, то накладные расходы при разработке всего ПО составят 6000 долларов ($ 65 * 50ч.+ $ 55 * 50ч.)=$ 3250 + $ 2750 = $ 6000.

4. Составление графика - на основе оценки трудозатрат можно составить график.

Если в рассматриваемом примере каждый разработчик затратил 25 часов на производство в

неделю и при этом он может работать в параллельном режиме (не учитывая зависимости), то согласно оценкам, 500 единиц LOC продукта будет поставлено в течение двух недель.

5. Выбор метрических показателей - на базе применяемых единиц измерения формируется метрический показатель. Его достаточно выбрать лишь один раз (в предыдущем примере LOC), а затем просто ссылаться.

Проблемы и риски, связанные с оцениванием размера ПО

Проблемы, возникающие в процессе оценивания

● проблема может быть недостаточно хорошо понята разработчиками и/или заказчиками из-за того, что некоторые факты были упущены или искажены из-за предвзятого отношения;

● недостаток либо полное отсутствие исторических данных не позволяет создать базу для оценок в будущем;

● специалисты-оценщики могут потерпеть неудачу при попытке описания того, насколько большой может быть система, еще до ее создания или даже еще до этапа разработки проекта;

● проектирующая организация не располагает стандартами, с помощью которых можно выполнять процесс оценивания (либо в случае наличия стандартов их никто не придерживается); в результате наблюдается недостаток совместимости при осуществлении процесса оценивания;

● согласно требованиям менеджеров и/или заказчиков, необходимо быстрое оценивание, и в этой ситуации важно понимать, что возможно непосредственное создание программного кода, минуя этапы анализа и разработки проектов;

● менеджеры проектов полагают, что было бы неплохо фиксировать требования в начале осуществления проекта, заказчики же считают, что не стоит тратить драгоценное время на разработку спецификации требований;

● при осуществлении менеджмента используются оценки с целью повышения производительности или достижения мотивированных целей;

● разработчики желают получать удовольствие от результатов менеджмента, а также находиться с заказчиками на равных;

● ошибки, как правило, будут скрытыми вместо того, чтобы оцениваться и отображаться, в результате чего создается ложное впечатление о фактической производительности;

● возможности оценивания существенно зависят от субъектов, вовлеченных в процесс внедрения;

● предварительное оценивание и составление предписаний рассматриваются в качестве целей исполнения; оценивание выполняется на подготовительном этапе, т.е. еще до того, как завершится исследование понятий;

● между предполагаемыми и реальными предписаниями возникают конфликты;

● повторное оценивание кажется излишним;

● для достижения желаемой четкости в функционировании других частей системы (интерфейсов наследственной системы, аппаратного обеспечения и т.д.) может потребоваться дополнительное ПО, что скажется на размерах программного продукта; имеет место недостаточно четкое представление об ограничениях как на уровне системы, так и иного рода;

● менеджеры, аналитики, разработчики, тестеры и те, кто внедряет продукт, могут бытовать самые разные представления о процессах оценивания и о возможностях совершенствования рассматриваемого продукта.

Итак, сложный и нелегкий процесс прогнозирования размеров программного продукта можно охарактеризовать с помощью проектных рисков.

Риски оценивания

Большинство типичных рисков, связанных с процессом создания программного продукта, возникают при оценивании размера, трудозатрат, стоимости работ и графиков по их выполнению. Причем в основе этих рисков лежат "проблемы, связанные с оцениванием".

Некоторые риски управления программными проектами, связанные с оценкой его размера

Если оценки неполные или некорректные, возникает очевидная угроза потерять заказчиков, а возможно, и утратить перспективы развития всего бизнеса. Столь же серьезные последствия имеют и ошибки при определении стоимости работ (в случае, если контракт заключается с указанием заранее оговоренной суммы). Слишком оптимистичные оценки могут привести к существенным финансовым потерям (возможно, даже в крупных размерах), а также к потере доверия со стороны клиентов.

Неверные оценки приводят к изменениям графика, к необходимости резко сокращать временные рамки выполнения работ, что зачастую приводит к существенным недоработкам в готовом проекте.

Возможности уменьшения степени риска, связанного с оценкой размера ПО.

Способы уменьшения влияния рисков, связанных с оценкой размера ПО

Многие из этих методов широко используются в практике:

● обратитесь к WBS, используйте самый малый уровень из возможных, применяйте подход "разделяй и властвуй";

● учитывайте то, что меньший размер компонентов способствует облегчению задачи оценщика;

● выполните обзор требований с учетом всех "отягчающих обстоятельств", включая выполнение действий, поддержку и сопровождение;

● если это возможно, различные предположения заменяйте информацией, основанной на опыте по организации подобных работ. Этот подход можно использовать и при отсутствии исторически конкретных данных. Самые невероятные случаи можно исключить из рассмотрения;

● необходимо поддерживать тесное сотрудничество с разработчиками других системных компонентов (возможно, при выполнении работы в параллельном режиме), причем желательно использовать единый понятийный аппарат;

● через определенные интервалы необходимо изменять оценки. Точность оценивания улучшается на протяжении всего жизненного цикла разработки ПО;

● с целью повышения степени доверия к получаемым результатам используйте несколько методов оценивания размера ПО;

● повышайте уровень квалификации разработчиков, широко используя методику оценивания размеров ПО.

Барри Боэм (Barry Boehm) и Каперс Джонс (Capers Jones) составили графические диаграммы, на основе которых можно сделать вывод, что уже на ранних этапах жизненного цикла можно успешно реализовывать оценку размера ПО. При этом становится возможным рассматривать перспективы эволюции первоначальных решений, принимаемых при работе над проектом. Если заказчики требуют приблизительных оценок, им следует учитывать, что в этом случае невозможно достижение большой степени точности. Обычно погрешность этого метода оценивания составляет 35% от конечной реальной величины (которая может представлять размер/объем затрачиваемых усилий / уровень выполнения графика работ / объем затрат). Этот метод чаще приводит к получению заниженных оценок размера ПО (см. рис.16.1).

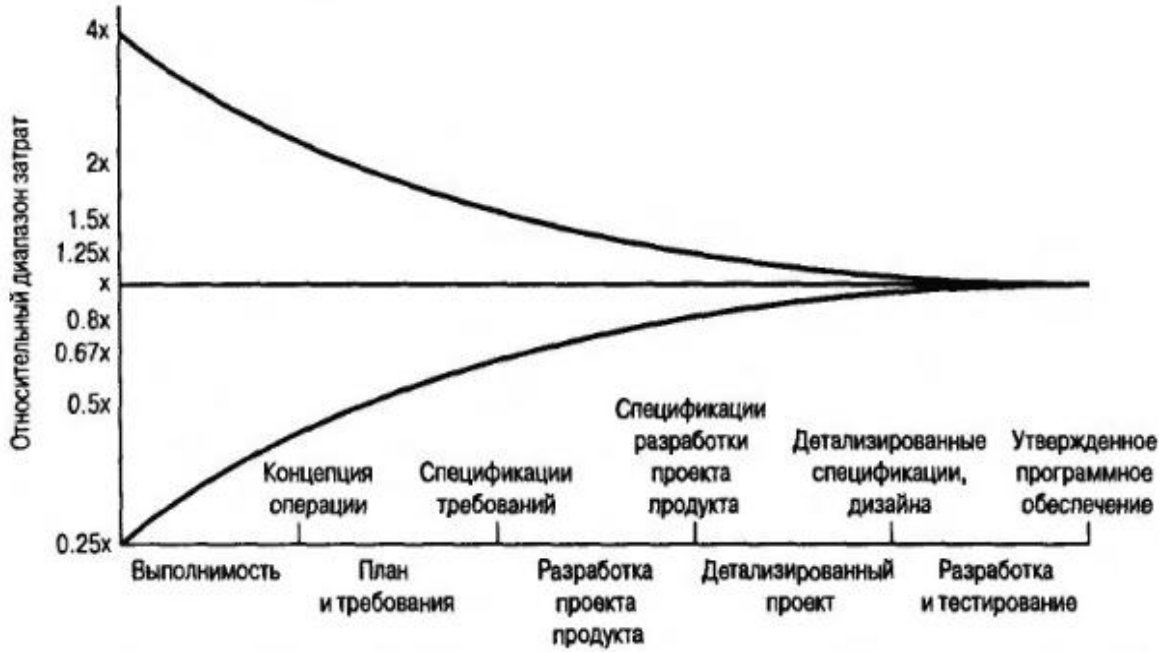

Рисунок 16.1 - Точность оценивания размера ПО

Определение размеров программного продукта

Начальный этап определения размеров программного продукта: оценивание

Определение размеров программного продукта представляет собой и прогнозирование структуры поставляемых продуктов (имеющих как внутреннюю, так и внешнюю природу) с целью удовлетворения требований проекта. В процессе оценивания также выполняется прогнозирование трудозатрат (ресурсов), необходимых для проектирования поставляемых продуктов.

Действия по определению размеров и оценке ПО попадают в центральную область последовательности задач по планированию программных проектов. Задаче оценке размеров ПО предшествует определение цели и области действия программного проекта, создание WBS, а также идентификация задач и действий. После задач по прогнозированию размеров ПО следуют задачи по оценке длительности и стоимости разработки, в процессе выполнения которых происходит распределение ресурсов, учет зависимостей, а также составление рабочего графика (см.рис.9.6).

Разбиение проекта на отдельные задачи (WBS)

Структура WBS представляет собой описание выполняемой работы, разбитой на ключевые

элементы (задачи). Каждая задача может быть управляемой, административной, интегральной либо поддающейся разработке.

Путем разбиения проекта на вышеупомянутые управляемые части производится оценивание размера каждого элемента (задачи), и трудозатрат на его разработку.

Трудозатраты выражаются в человеко-часах, неделях, месяцах и т.д. При использовании WBS задачи идентифицируются на уровне выполнимости с учетом наличия доступного персонала, обладающего необходимыми навыками. После определения необходимого количества персонала, а также установки требуемых навыков оценка трудозатрат будет соотноситься с календарным планированием для определения длительности выполнения проекта и его ключевых стадий. Результирующий график проекта обычно отображается в виде диаграмм Ганта.

И напоследок стоит отметить, что каждая задача определяется, оценивается и отслеживается на протяжении выполнения всего проекта.

Существует несколько представлений WBS.

● ***Представление продукта*** определяет иерархические взаимосвязи между элементами продукта (процедуры, модули, подсистемы и т.п.).

● ***Представление проекта*** указывает на иерархические взаимосвязи между рабочими действиями (элементы процесса). При выполнении оценок и измерительных операций для графиков-прогнозов потребуется учитывать оба набора элементов и действий. Задачи по разработке (продукта) могут включать анализ, высокоуровневое и низкоуровневое проектирование, кодирование, тестирование и т.д. Все вышеперечисленное происходит на фазах жизненного цикла.

Управленческие задачи включают:

● планирование проекта,

● отслеживание,

● контроль,

● анализ рисков и т.п.

Задачи поддержки могут включать документацию на поставляемые продукты, например, руководства пользователя, операционные пособия, а также справочники, описывающие выполнение сетевых коммуникаций.

Задачи управленческие, административные, а также задачи поддержки называются интегральными. При этом рассматривается одна задача или больше, а иногда и все задачи по разработке ПО. Другие интегральные задачи включают: процедуры управления конфигурацией, практику обеспечения качества ПО, управление и анализ рисков, технику и инструменты разработки ПО, методы проведения обзоров незавершенных поставляемых продуктов,

собираемые и анализируемые метрические показатели, определение и документирование стандартов (правила, применимые для незавершенных поставляемых продуктов), а также любые другие действия, необходимые для удовлетворения требований заказчиков (например, создание документации, учебные программы, инструменты разработки либо получения необходимых результатов).

Структура WBS обеспечивает выполнение планирования и отслеживания рабочих действий, определение стоимости или графика путем предоставления инженеру либо менеджеру глобальной точки зрения. Благодаря этому можно организовать и иллюстрировать процесс выполнения работы. В результате идентифицируется вся необходимая работа; распределение работы на небольшие по размеру, хорошо определенные задачи; облегчение планирования, оценки и составления графика для проекта. При этом поддерживается фундамент, необходимый для выполнения мониторинга данных и сбора исторических данных; а также для идентификации договорных задач и конструктивных узлов. Это позволит нам заменить высказывание "сделано на 90%" на "мы выполнили 97 из 234 задач". "Утяжеленные" задачи приобретают значимость и могут быть связаны с оценкой затрат.

При использовании структуры WBS в ваше распоряжение предоставляется содержание, иерархический перечень рабочих действий, требуемых для завершения проекта. В результате, WBS выступает в роли "неприметного" инструмента, применяемого для определения размера и оценки действий процесса.

Каждый процесс включает входные данные, преобразования и выходные данные.

Следующие дополнительные вводы применяются для измерения и оценки процесса (в то время как для выводов характерны оцененный размер, трудозатраты, длительность [график], а также затраты):

● план выполнения проекта или рабочий план (Statement of work, SOW);

● начальная проектная документация;

● перечень требований:

- ожидаемый уровень производительности;

- указанные для включения свойства;

- методы оценки результатов;

● ограничения;

● используемые процессы и стандарты:

- методы разработки ПО;

- правила, используемые в качестве руководства к действию, и применяемые критерии оценки качества;

● контракт на оказание услуг;

● предыдущий опыт решения подобных задач;

● хронологическая оценка и фактические данные, используемые в вашей организации;

● применяемые языки программирования;

● информация о повторно используемом ПО;

● сведения о разработке проекта системы;

● начальные концепции архитектуры ПО - основные компоненты;

назначение и область действия проекта, включая выполняемые задачи и поставляемые продукты.

Конечно, маловероятно, чтобы большая часть из этих входных данных была доступной в ходе процесса оценки, но в данном случае справедливым является принцип "чем больше, тем лучше". Благодаря использованию возможностей WBS оценка размера ПО превращается в серьезное мероприятие.

Оценка размера ПО в процессе разработки проекта

Оценка показателя LOC с помощью экспертных заключений и восходящего суммирования

Пусть структура WBS содержит несколько уровней декомпозиции в иерархии продукта/проекта. Требования к иерархии продуктов WBS могут быть разбиты на фактические компоненты программных систем.

Максимально детализированная структура WBS может принести пользу на стадии точных измерений, за которой следует стадия точных оценок.

Как только структура WBS будет разбита с учетом выделения самых нижних уровней, может быть создан "статистический" показатель размера.

При этом используются процессы измерения и суммирования. Величина размера каждого компонента может быть получена путем опроса экспертов, разрабатывавших подобные системы, либо путем использования данных опроса потенциальных разработчиков подобных систем. В результате становится возможной оценка размеров каждого блока на нижних уровнях структуры WBS.

После сложения результатов измерений полученный итог будет называться оценкой размера "снизу-вверх". Улучшенная оценка может быть получена в том случае, если каждый оценщик

произведет оптимистическую, пессимистическую и реалистическую оценку размеров. Затем формируется бета-распределение путем умножения реалистической оценки размера на 4, добавления оптимистической и пессимистической оценок с последующим делением результата на 6. Подобное взвешенное среднее значение является удобным в условиях естественной неопределенности процесса оценивания.

Например, если данный оконный объект отображается в структуре WBS для системы, поддерживающий код, который требуется для реализации процесса редактирования в данном окне, может занимать от 200 до 400 строк кода, причем, скорее всего эта цифра окажется более близкой к 250. Учитывая предложенные оценщиком пессимистические и оптимистические сценарии, можно получить следующую итоговую оценку:

ПО - Пессимистическая оценка размеров,

ОО - Оптимистическая оценка размеров,

РО - Реалистическая оценка размеров,

(ПО+ОО+(РО*4))/6= LOC

(200 + 400 +(250*4))/6 = 266,67=266 LOC

Количество тысяч строк исходного кода (KSLOC) является производным от общей метрики, вводимой при оценках производительности.

Обычно производительность выражается в KSLOC/SM либо KLOC/SM (где SM - staff-month (человеко-часы)).

Подсчет строк существующего кода вручную является достаточно утомительным и длительным занятием. В силу этого многие организации приобретают либо разрабатывают автоматизированные LOC-счетчики.

Подсчет вручную:

● Каждая учитываемая "строка исходного кода" должна содержать лишь один оператор (если в одной строке содержатся два выполняемых оператора, разделяемых точкой с запятой, то будут учитываться две строки; если же один выполняемый оператор разбит на две "физические" строки, он будет учитываться как один оператор). В языках программирования допускаются все опции кодирования, но обычно проще определять в строке один выполняемый оператор, обрабатываемый компилятором либо интерпретатором.

● Необходимол учитывать все имеющиеся выполняемые операторы- конечный пользователь может не иметь возможности непосредственно использовать каждый оператор, но все операторы должны поддерживаться данным продуктом (в том числе и утилитами).

● Определение данных учитывается лишь один раз.

● Не учитываются строки, содержащие комментарии.

● Не учитывается отладочный код либо любой другой временный код (пробное ПО, средства тестирования, инструменты разработки и прототипирования, а также другие подобные средства).

● Необходимо учитывать каждую инициализацию, вызов либо включение (иногда называемое директивой компилятора) макроса в качестве части исходного кода, в котором осуществляется то либо иное действие (не учитывайте повторно используемые операторы исходного кода).

● Трансляция количество строк исходного кода в эквивалентные строки языка ассемблера, даст вам возможность одновременно выполнять сравнительный анализ для нескольких проектов.

Первый и второй столбцы таблицы рисунка. 16.2 представляют метод трансляции SLOC, применяемый в различных языках по отношению к среднему количеству строк SLOC в базовом языке ассемблера. (Обратите внимание, что SLOC и LOC являются взаимозаменяемыми.) Большинство менеджеров проектов предпочитают выполнять трансляцию с языков программирования на базовый язык ассемблера по той простой причине, что в этом случае операции сравнения во многих проектов могут производиться на идентичной основе. Эти данные могут быть полезными также в случае трансляции проекта с обычного языка программирования на язык преобразования.

Язык программирования	Basic Assembler SLOC (уровень)	Средний показатель SLOC из расчета на одну функциональную точку
Basic Assembler	1	320
Autocoder	1	320
Macro Assembler	1,5	213
C	2,5	**128**-150
Пакетные файлы DOS	2,5	128
Basic	3	107
Макросы LOTUS	3	107
ALGOL	3	105-106
COBOL	3	105-107
FORTRAN	3	105-106
JOVIAL	3	105-107
Смешанные языки программирования (по умолчанию)	3	105
Pascal	3,5	91
COBOL (ANSI 85)	3,5	91
RPG	4	80
MODULA-2	4,5	80
PL/1	4,5	80
Параллельный Pascal	4	80
FORTRAN 95	4,5	71
BASIC (ANSI)	5	64
FORTH	5	64
LISP	5	64
PROLOG	5	64
LOGO	5,5	58
Расширенный общий LISP	5,75	56
RPG III	5,75	56
C++	**6**	**53**
JAVA	6	53
YACC	6	53
Ada 95	6,5	49
CICS	7	46
SIMULA	7	46
Языки баз данных	8	40
CLIPPER DB и dBase III	8	40
INFORMIX	8	40
ORACLEи SYBASE	8	40
Access	8,5	38
Dbase IV	9	36
FileMaker Pro	9	36
Языки поддержки принятия решений	9	35
FOXPRO 2.5	9,5	34
APL	10	32
Статистические языки (SAS)	10	32
DELPHI	11	29
Стандартные объектно- ориентированные языки	11	29
OBJECTTVE-C	12	27
Oracle Developer/2000	14	23
SMALLTALK	15	21

Рисунок 16.2 - Показатели SLOC при преобразовании кода с языка программирования на язык Basic Assembler SLOC из расчета на одну функциональную точку

Например, предположим, что операционная система, написанная на языке С и насчитьшающая 50 000 строк LOC, будет преобразована в С++. Руководствуйтесь сведениями из табл. 10.1, где базовый показатель SLOC языка С относительно языка ассемблера равен 2.5. Поэтому операционная система, содержащая 50000 строк SLOC, написанных на С, будет эквивалентной 125000 строкам в случае использования языка ассемблера (50,000 x 2.5). Если же 125000 строк операционной системы на языке ассемблера были переписаны с учетом использования языка С++, получим 125000/6, или 20833 LOC или 20,833 SLOC.

Преимущества при использовании LOC в качестве единиц измерения

● эти единицы измерения широко распространены и могут легко адаптироваться;

● позволяют выполнять сопоставление методов измерения размеров и производительности в различных группах разработчиков;

● непосредственно связаны с конечным продуктом;

● единицы LOC легко оцениваются еще до завершения проекта;

● оценка размеров ПО производится на основе точки зрения разработчика - физическая оценка созданного продукта (количество написанных строк кода);

● действия по непрерывному улучшению базируются на оценочной технике - спрогнозированный размер может быть легко сопоставлен с реальным размером в ходе осуществления постпроектного анализа. (Насколько точной была оценка? Что означает определенный процент? Что может быть изучено в рамках оценки размера следующего проекта?).

Недостатки, связанные с применением метода LOC

● единицы измерения LOC затруднительны в применении при оценке размера ПО на ранних стадиях жизненного цикла разработки;

● исходные инструкции могут различаться в зависимости от типов языков программирования, методов проектирования, стиля и способностей программиста;

● применение методов оценки с помощью подсчета количества строк кода не регламентируется промышленными стандартами (например, ISO);

● разработка ПО может быть связана с большими затратами, которые прямо не зависят от размеров программного кода - "фиксированные затраты", такие как спецификации требований и пользовательские документы не включены в прямые затраты на кодирование;

● программисты могут быть незаслуженно премированы за достижение высоких показателей LOC в случае, если служба менеджмента по ошибке посчитает это признаком высокой

продуктивности, но при этом будет отсутствовать тщательно разработанный проект; исходный код не является самоцелью при создании готового продукта - главную роль играют функциональные свойства и показатели производительности;

● при подсчете количества единиц LOC следует различать автоматический и вручную созданный код - эта задача является более сложной, чем "простой подсчет", который может быть выполнен на основе листинга, сгенерированного компилятором, либо с помощью утилиты, выполняющей подсчет строк программного кода;

● показатели LOC не могут применяться при осуществлении нормализации в случае, если применяемые платформы или языки являются различными;

● единственный способ применения учета с помощью единиц измерения LOC по отношению к разрабатываемому ПО заключается в использовании метода аналогии на основе сравнения функциональных свойств у подобных программных продуктов, либо в использовании мнений экспертов (однако, эти методы не относятся к числу точных);

● генераторы кода зачастую продуцируют чрезмерный объем кода, в результате чего

● искажаются показатели LOC.

Метод функциональных точек

К сожалению, довольно часто производительность труда программистов оценивается количеством производимых LOC. И если средняя производительность программиста выросла с 200 LOC/месяц до 250 LOC/месяц, менеджер может прийти к заключению относительно роста производительности. Это впечатление зачастую является ложным, хотя в итоге будут поощряться те разработчики, которые производят больше строк LOC из расчета на разработку проекта. Более правильным будет награждать разработчиков не только за высокие показатели производительности, а за качественный безошибочный код. При оценке качества может применяться следующая формула:

количество дефектов / количество строк кода

Большая величина знаменателя будет свидетельствовать о высоком качестве кода.

Фаза кодирования в большинстве проектов обычно требует от 7% до 20% от общего объема трудозатрат. И конечно, более важным является качество программного кода, а не его объем. Результатом подобных размышлений является осознание необходимости другой единицы измерения. И в качестве такого метода может выступать метод функциональных точек.

Использование метода функциональных точек в качестве единиц измерения

Метод функциональных точек (Function point, FP) основывается на том, что размер ПО

лучше всего оценивать в терминах количества и сложности функций, реализованных в данном программном коде, а не посредством количества строк кода. При использовании метода функциональных точек измеряются категории пользовательских бизнес-функций. При этом способ их определения является более методологичным, чем в случае подсчета строк LOC. В этом случае уместно сравнить программу с домом: площадь дома, выраженная в квадратных метрах, аналогична применению единиц измерения LOC; количество спален и туалетных комнат в доме аналогично применению в качестве единиц измерения функциональных точек.

Ранее применявшийся подход учитывал лишь размер ПО; нынешний подход учитывает размер и функциональные свойства.

Метод функциональных точек позволяет выполнять следующие задачи:

● оценивать категории пользовательских бизнес-функций;

● разрешить проблему, связанную с попыткой применения единиц измерения LOC на ранних стадиях жизненного цикла разработки;

● определять количество и сложность входных и выходных данных, запросы к базе данных, файлы либо структуры данных, а также внешние интерфейсы, связанные с программной системой.

Краткий обзор процесса применения функциональных точек:

1. подсчитываются функции в каждой категории (перечень категории: вывод, ввод, опросы, структуры данных и интерфейсы);

2. определяется сложность каждой функции - простая, средняя, сложная;

3. устанавливаются требования для каждой из категорий;

4. каждая функция умножается на соответствующий ей параметр, а затем суммируется с целью получения общего количества функциональных точек;

5. производится преобразование функциональных точек в единицы измерения LOC с помощью следующей формулы:

LOC = Функциональные точки * ADJ * множитель преобразования

Здесь аббревиатура ADJ обозначает настройку общих характеристик приложения. Множитель преобразования основывается на статистических показателях для приложения и языка программирования, представляя среднее количество строк кода, применяемых для реализации простой функции. Какова же задача, выполняемая этим параметром? Потребность в нем обосновывается тем, что большинство автоматизированных инструментов, предназначенных для оценки трудозатрат, обычных затрат, а также составления графика, нуждаются в использовании LOC в качестве входных данных.

Процесс использования метода функциональных точек.

На рис. 16.3. представлен рабочий лист анализа по методу функциональных точек. Технологию заполнения и расчета расммотрим ниже

Шаг 1.	**Подсчет количества функций в каждой категории**			
Шаг 2.	**Применение весовых множителей сложности**			
	Простой	**Средний**	**Сложный**	**Функциональные точки**
Количество выводов	_*4	_*5	_*7	
Количество вводов	_*3	_*4	_*6	
Количество опросов выводов	_*4	_*5	_*7	
Количество опросов вводов	_*3	_*4	_*6	
Количество файлов	_*7	_*10	_*15	
Количество интерфейсов	_*5	_*7	_*10	
Общее количество функциональных точек				
Шаг 3.	**Применение факторов среды**			
Фактор среды				**Рейтинг (0,1,2,3,4,5)**
Каналы передачи данных				
Распределенные вычисления				
Требования к производительности				
Конфигурирование с ограничениями				
Частота транзакций				
Интерактивный запрос и/или запись				
Эффективность на уровне конечного пользователя				
Интерактивное обновление				
Сложная обработка				
Повторное использование				
Упрощение преобразования/установки				
Упрощение операции				
Используется на нескольких узлах				
Потенциал изменения функции				
Итого (N):				
Шаг 4.	**Вычисление корректировочного множителя сложности (CAF)**			
	CAF = 0,65 +(0,01*N) =			
Шаг 5.	**Вычисление скорректированных функциональных точек (AFF)**			
	AFP = FP (исходное число) * CAF =			
Шаг 6.	**Преобразование в строки LOC (дополнительно)**			
	LOC = AFP * LOC/AFP=			

Рисунок 16.3 - Рабочий лист анализа по методу функциональных точек

Шаг 1. Подсчет функций в каждой категории

1.1. Общие указания

● Учитываются только функции, удовлетворяющие программным требованиям.

● Учитываются логические представления. Если в процессе ввода либо вывода требуется другая обрабатывающая логика, каждое из подобных логических представлений рассматривается как уникальная функциональная точка.

Оценка размера разрабатываемой системы непременно влечет за собой выполнение исследования основных системных компонентов. При этом возникает множество вопросов. Каков объем выводимых данных? Насколько важны входные данные при генерировании вывода? Каков объем хранимых данных?

● Учитывается количество элементов: вывод, ввод, опросы и файлы.

Предварительная архитектура обеспечивает основу для выполнения подобных действий. Некоторые предпочитают начинать с архитектуры в форме текстовых требований, однако использование архитектуры в графической форме является весьма полезным.

Весовые множители, применяемые ко всем видимым внешним аспектам ПО, представляют собой набор эмпирических констант, которые являются производными от данных испытаний и ошибок.

1.2. Учет выводимых данных

● Внешние выводы представляют собой данные, продуцируемые ПО с целью их передачи за пределы системы.

● Выводы представляют собой единицы деловой информации, которые генерируются ПО и предназначены для конечного пользователя (ориентированы на приложение). В качестве примеров выводов могут служить экранные данные, отчетные данные, сообщения об ошибках и т.д.

● Учитывается каждая уникальная единица вывода, которая покидает область приложения. Единица вывода будет уникальной в случае, если она имеет другой формат и/или требует другой логики обработки.

● Применяются структурированные методы в случае, если вывод представляет собой поток данных, сгенерированный ПО и предназначенный для конечного пользователя. Количество выводимых данных, выходящих за пределы области приложения, может быть легко подсчитано с помощью диаграммы источников/приемников. Каждый выводимый результат добавляется к одному из трех итоговых значений в зависимости от сложности этого результата: итог для простых выводимых результатов, итог для выводимых результатов средней сложности и итог для сложных выводимых результатов. Такое разделение позволяет применить весовой множитель для каждого из типов итога- сложный вывод требует больше усилий при создании по сравнению со средним или простым выводом.

Инструкции по определению уровня сложности приведены на рис.16.4

Часть вывода	От 1 до 5 элементов	От 6 до 19 элементов	20 и более элементов данных
От 0 до 1	Простой 4	Простой 4	Средний 5
От 2 до 3	Простой 4	Средний 5	Сложный 7
4 и более	Средний 5	Сложный 7	Сложный 7
Часть ввода			
От 0 до 1	Простой 3	Простой 3	Средний 4
От 2 до 3	Простой 3	Средний 4	Сложный 6
4 и более	Средний 4	Сложный 6	Сложный 6

Рисунок 16.4 - Весовые множители для ***вывода/ввода, опросов***, применяемые при анализе методом функциональных точек

Учет структур данных (файлы)

При подсчете структур данных (файлов) следует учитывать следующее:

● внутренние файлы представляют собой логические файлы в составе программы;

● структуры данных (ранее известные как "файлы") представляют собой первичную логическую группу пользовательских данных, которые постоянно находятся полностью внутри границ программной системы;

● структуры данных доступны для пользователей с помощью ввода, вывода, опросов либо интерфейсов.

Структуры данных делятся на простые, средние и сложные.

Инструкции по определению степени сложности приводятся в табл. на рис. 16.5.

Учитывайте логические взаимосвязи, а не физические типы записей	От 1 до 9 элементов данных	От 20 до 50 элементов данных	51 и более элементов данных
1 логическая запись формат / взаимосвязь	Простой 7	Простой 7	Средний 10
От 2 до 5 логических записей типа формат / взаимосвязь	Простой 7	Средний 10	Сложный 15
6 и более логических записей типа формат / взаимосвязь	Средний 10	Сложный 15	Сложный 15

Рисунок 16.5 - Весовые множители ***файлов*** в анализе методом функциональных точек

Подсчет количества интерфейсов

При подсчете количества интерфейсов следует учитывать следующее:

● *внешние файлы* представляют собой файлы, сгенерированные компьютером, которые используются программой;

● *интерфейсы* представляют собой данные (и систему управления), которые хранятся за пределами программной системы при выполнении оценки;

● *структуры данных,* разделяемые несколькими системами, учитываются в виде интерфейсов и структур данных;

● учитывается каждый поток данных и управления в любом направлении в качестве уникального интерфейса. Интерфейсы делятся на простые, средние и сложные.

Инструкции по определению уровня сложности приведены в таблице на рис.16.6.

Учитывайте логические взаимосвязи, а не физические типы записей	От 1 до 9 элементов данных	От 20 до 50 элементов данных	51 и более элементов данных
1 логическая запись формат / взаимосвязь	Простой 5	Простой 5	Средний 7
От 2 до 5 логических записей типа формат / взаимосвязь	Простой 7	Средний 7	Сложный 10
6 и более логических записей типа формат / взаимосвязь	Средний 7	Сложный 15	Сложный 10

Рисунок 16.6 - Весовые множители интерфейсов в анализе методом функциональных точек

Шаг 2. Применение весовых множителей сложности

● Необходимо умножить каждую величину определенного типа (простой, средний, сложный) внутри каждой категории (вывод, ввод, опросы [вывод/ввод], структура данных [файлы], интерфейсы) на соответствующий весовой множитель. Весовые множители, приведенные в таблицах рис. 16.4-16.6.

● В каждую категорию добавляются тоговые результаты, выраженные в количестве "физических функциональных точек".

Шаг 3. Применение факторов среды

Корректировка общего итога по физическим функциональным точкам происходит так, чтобы учитывать факторы среды, влияющие на процесс разработки ПО в целом. Известно, что многие аспекты среды могут оказывать влияние на процесс разработки программ. Некоторые из этих аспектов оказывают на проект положительное влияние, другие же "качают весы" в отрицательном направлении; все они рассматриваются с учетом их уникальности в применении к конкретному проекту. В таблице на рис. 16.7 приводится детализованное определение каждого из 14 факторов среды, либо оказывающих влияние множителей корректировки. Здесь же можно найти инструкции по выбору веса фактора среды.

Ниже приводится краткое описание выполнение процесса взвешивания в среде.

Используя материал таблицы рис.16.7, оцените каждый фактор по шкале от 0 до 5 (в данном случае 0 означает невозможность применения фактора). С целью получения "эффекта присутствия" на одном из краев спектра оценивания, в табл. 16.8 приводятся примеры программных систем с высоким рейтингом - показатели рейтинга от 4 до 5 на шкале.

Просуммируйте рейтинги факторов (Fn) с целью вычисления итогового фактора влияния среды (N).

$$N = sum\ (Fn)$$

Значения заполняются в рабочий лист анализа по методу функциональных точек рис. 16.3. Заполненный рабочий лист представлен на рис. 16.9.

Фактор среды	Рейтинг (0,1, 2,3,4, 5)
Каналы передачи данных	Данные или контрольная информация, принимаемые или получаемые через каналы передачи данных. Интерактивные системы всегда испытывают некоторое влияние со стороны каналов передачи данных
Распределенные вычисления	Приложение, использующее данные, которые хранятся, обрабатываются и к которым обеспечивается доступ в хранилище или системе обработки, отличающихся от используемых в основной системе
Требования к производительности	Требования, одобренные пользователем, которые были применены для получения высокой скорости передачи данных либо малого времени отклика
Конфигурирование с ограничениями	Приложение, выполняемое с применением интенсивно используемой, ограниченной или наполненной конфигурации
Частота транзакций	Высокий сетевой трафик, перегруженность экранов информацией и графикой, высокая частота обновления экрана
Интерактивный запрос и/или запись	Высокая степень интерактивности
Эффективность на уровне конечного пользователя	Необходимо дополнительно учитывать человеческий фактор
Интерактивное обновление	Динамическое обновление базы данных, распределенные базы данных
Сложная обработка	Высокая степень безопасности, обработка со множеством транзакций, сложные алгоритмы, логика, управляемая прерываниями
Повторное использование	Код, разработанный с целью повторного использования, должен обладать высоким качеством
Упрощение преобразования/установки	Процессы преобразования и установки требуют наличия документов по планированию, которые были протестированы
Упрощение операции	Эффективные, но простые процедуры запуска, резервирования, восстановления при появлении ошибок, а также завершения. Минимальное вмешательство со стороны пользователя
Используется на нескольких узлах	Учет различий в бизнес-функциях
Потенциал изменения функции	Модульность, управляемость с помощью таблиц, поддержка со стороны пользователей, возможности по формированию запросов и т.д.

Рисунок 16.7 - Описание факторов среды при анализе методом функциональных точек

Шаг 4. Вычисление множителя корректировки сложности (CAF)

Уровень неопределенности оценок является функцией фазы жизненного цикла. Это поддерживает теорию эмпирических данных, согласно которой максимальная степень влияния факторов среды на итоги анализа функциональных точек может составлять +/- 35%. Причем рассматривается максимальная степень влияния, поскольку анализ функциональных точек выполняется в начале жизненного цикла. Как следствие, в этом случае существует

максимальная вероятность неточности измерений. С целью компенсации подобной неопределенности в случае недостаточного уровня знаний множитель корректировки сложности (CAF) будет применяться к итоговым значениям факторов среды.

№	Фактор среды	Примеры высокопроизводительных систем
1	Сложные коммуникации данных	Программа, предназначенная для международного банка, которая выполняет денежные переводы из финансовых учреждений, разбросанных по всему миру
2	Распределенная обработка	Механизм поиска в Internet, при реализации которого обработка выполняется несколькими десятками компьютеров, работающими в параллельном режиме
3	Цели, требующие достижения максимальной производительности	Система контроля воздушного трафика, поддерживающая точное и своевременное определение позиции воздушного судна на основе показаний радиолокатора
4	Интенсивно используемое конфигурирование	Университетская система, реализующая одновременную регистрацию нескольких сотен студентов, находящихся в лаборатории
5	Высокая частота транзакций	Банковская программа, выполняющая миллионы транзакций за ночь с целью сведения баланса по всем книгам на начало нового рабочего дня
6	Интерактивный ввод данных	Программа утверждения закладных, с помощью которой клерки могут вводить данные в компьютерную систему в интерактивном режиме, используя бумажные анкеты, заполненные будущими домовладельцами
7	Проект, дружественный по отношению к пользователю	Программное обеспечение для компьютеризованных касс на станциях метро, оборудованных сенсорными экранами, позволяющими покупать билеты с помощью кредитных карточек
8	Интерактивное обновление данных	Авиационная система, позволяющая агентам туристических фирм заказывать авиарейсы и получать информацию о наличии свободных мест. Это ПО должно обеспечивать возможность блокировки и модификации некоторых записей в базе данных с целью предотвращения повторной продажи билетов на одни и те же места
9	Комплексная обработка	Медицинское ПО, которое на основе различных симптомов, выявленных у пациента, а также путем обширного логического вывода позволяет устанавливать предварительный диагноз
10	Повторное использование	Рабочий процессор, который может быть спроектирован таким образом, что его панели инструментов и меню могут встраиваться в другие приложения, такие как электронные таблицы или генератор отчетов
11	Облегченная установка	Приложение, выполняющее контроль оборудования, с помощью которого даже неспециалист может установить и настроить оборудование морской буровой вышки
12	Облегченное оперирование	Программа, предназначенная для анализа большого количества хронологических финансовых записей, которая должна оптимизировать обработку информации таким образом, чтобы не вынуждать операторов постоянно менять носители данных
13	Несколько узлов	ПО по поддержке платежных ведомостей для транснациональной корпорации, которое должно учитывать различные характеристики разнообразных стран, включая различную валюту и правила определения налоговых ставок
14	Гибкость	Программа финансового прогноза, которая может формировать месячные, квартальные либо годовые прогнозы, необходимые конкретному бизнес-менеджеру, который может нуждаться в

Рисунок 16.8 - Факторы среды при анализе методом функциональных точек, примеры высокопроизводительных систем

CAF = 0.65 +(0.01 *N) , где N является суммой взвешенных факторов среды.

Поскольку приходится иметь дело с 14 предполагаемыми факторами среды, каждый из которых имеет вес, изменяющийся в диапазоне от 0 до 5, наименьшее значение для N может быть 0 (ни один из 14 не применим); а наибольшее значение для N может быть 70 (каждый из 14 факторов имеет максимальный вес, равный 5). Исходя из этих граничных условий, приходим к выводу, что минимум CAF = 0.65 + (0,01 * 0) = 0,65. Максимум CAF = 0.65 + (0,01 * 70) = 1,35. (1.35 - 0.65 = 0.70). Ранняя оценка размеров и трудозатрат может отклоняться при условии использования фактора на величину +/- 35%.

Шаг 4 проиллюстрирован в таблице рис.16.9.

Шаг 5. Вычисление скорректированных функциональных точек. Скорректированные функциональные точки (AFP) - физические функциональные точки x CAF. Шаг 5 рассматривается в в таблице рис.16.9.

Шаг 6. Преобразование в строки LOC (дополнительно).

Метод функциональных точек обеспечивает способ предварительной оценки размера потенциальных программ либо программных систем. При этом осуществляется анализ будущих функциональных свойств с пользовательской точки зрения. Языки программирования являются весьма различными с точки зрения их характеристик, однако существует некое среднее количество выполняемых операторов, требуемых для реализации одной функциональной точки.

Преобразование функциональных точек в строки LOC может потребоваться в силу следующих причин:

● с целью измерения и сравнения производительности либо размера программ, либо систем, которые были написаны на различных языках программирования;

● с целью использования стандартных единиц измерения для осуществления ввода данных в инструментальные средства оценки;

● для преобразования размера программы либо приложения, написанных на любом языке программирования, в эквивалентный размер в случае приложения, написанного на другом языке программирования.

Вплоть до завершения шагов 1-5 по отношению к данным может быть применено достаточно точное преобразование, позволяющее перейти от функциональных точек к LOC.

Частичное преобразование "функциональные точки/язык" показано в таблице рис.16.9.

Здесь иллюстрируется преобразование функциональных точек в LOC (первый и третий столбцы). В таблице перечислены далеко не все языковые преобразования, одобренные 11-TUG (этот перечень достаточно велик). Также следует отметить, что этот перечень постоянно пополняется по мере разработки новых языков программирования.

LOC = скорректированные функциональные точки * LOC на скорректированную функциональную

точку AFP x # LOC на AFP = LOC

Пример завершенного рабочего листа анализа функциональных точек приводится в таблице рис.16.9.

Шаг 1.	Подсчет количества функций в каждой категории			
Шаг 2.	Применение весовых множителей сложности			
	Простой	Средний	Сложный	Функциональные точки
	1	2	3	(1+2+3)
Количество выводов	12* 4=48	11* 5=55	5* 7=35	48+55+35=138
Количество вводов	8* 3=24	9* 4=36	6* 6=36	96
Количество опросов выводов	5* 4=20	7* 5=35	3* 7=21	76
Количество опросов вводов	5* 3=15	8* 4=32	4* 6=24	71
Количество файлов	12* 7=84	3* 10=30	2* 15=30	144
Количество интерфейсов	9* 5=45	6* 7=42	4* 10=40	127
Общее количество функциональных «физических»точек				652
Шаг 3.	Применение факторов среды			
Фактор среды				Рейтинг (0,1,2,3,4,5)
Каналы передачи данных				5
Распределенные вычисления				5
Требования к производительности				3
Конфигурирование с ограничениями				0
Частота транзакций				5
Интерактивный запрос и/или запись				4
Эффективность на уровне конечного пользователя				5
Интерактивное обновление				4
Сложная обработка				2
Повторное использование				2
Упрощение преобразования/установки				3
Упрощение операции				4
Используется на нескольких узлах				5
Потенциал изменения функции				4
			Итого (N):	51
Шаг 4.	Вычисление корректировочного множителя сложности (CAF)			
	CAF = 0,65 +(0,01*N) =0,65+(0,01*51)=1,16			
Шаг 5.	Вычисление скорректированных функциональных точек (AFP)			
	AFP = FP (исходное число) * CAF = 652 * 1,16 = 756,32			
Шаг 6.	Преобразование в строки LOC (дополнительно)			
	LOC = AFP * LOC/AFP= 756,32 * 128 = 96,808.96 LOC 128 из таблицы рис.16.2			

Рисунок 16.9 - Заполненный рабочий лист анализа по методу функциональных точек

Преимущества анализа методом функциональных точек

● Этот метод может применяться на ранних стадиях жизненного цикла разработки программного обеспечения - размер ПО может определяться на фазе выдвижения требований либо на фазе разработки проекта;

● Не зависит от используемого языка программирования, технологии, а также техники, исключая некоторые корректировки, выполняемые на заключительной стадии.

● Метод функциональных точек обеспечивает надежную взаимосвязь с затрачиваемыми усилиями (в случае, если вы сможете определить корректную функцию для выполнения нужных измерений).

● Создание дополнительных функциональных точек из расчета на час (неделю либо месяц), является целью при достижении желательного уровня производительности (в отличие от создания дополнительных строк LOC из расчета на час (неделю либо месяц), что является не столь оправданным, пожалуй, даже парадоксальным).

● Для пользователей эти единицы измерения могут быть более привычными. При этом облегчается понимание степени влияния изменений функциональных требований.

● Может быть измерен уровень производительности проектов, написанных на различных языках программирования.

● Функциональные точки обеспечивают механизм, позволяющий выполнять отслеживание и мониторинг "сползания" области видимости. Функциональные точки могут учитываться часто и на ранних этапах - результаты подсчета функциональных точек могут сравниваться в конце этапов формирования требований, проведения анализа, разработки проекта, а также внедрения. Если количество функциональных точек увеличивается при каждом подсчете, это означает, что проект был лучше определен либо размер проекта был увеличен (это довольно опасно, если только график и/или средства будут перераспределены).

● Функциональные точки могут использоваться в системах, в которых реализован графический пользовательский интерфейс (GUI), в клиент-серверных системах, а также при объектно-ориентированной разработке.

● Функциональные точки могут учитываться как пользователями высшего уровня (клиентами или заказчиками), так и простыми техниками.

● Учитываются факторы среды. Так же, как и в случае с моделями определения размеров и оценивания, может выполняться адаптация и калибровка. Могут учитываться способы применения весов, а также факторы среды, которые в целом являются изменяемыми. Например, в промышленности полупроводников в случае, когда физические модули тестируются с применением ПО, компоненты устройств могут учитываться вместо вводов и выводов.

Недостатки анализа методом функциональных точек

● Используются субъективные оценки, вследствие чего возможны судебные разбирательства.

● Получаемые результаты зависят от технологии, используемой для их реализации.

● Многие модели затрат и трудозатрат зависят от показателя LOG, поэтому приходится преобразовывать функциональные точки.

● Требуются дополнительные исследования данных на базе LOG, отличные от исследований в случае данных, основанных на функциональных точках.

● Этот метод лучше применять после создания спецификации дизайна. Данный метод не очень хорошо подходит в случае с приложениями, не относящимися к классу MIS-приложений (в этом случае используйте точки свойств).

Примечание

На сегодняшний день традиционным представлением корпоративной системы автоматизации промышленного предприятия является так называемая пирамида автоматизации, которая включает следующие известные три типовых уровня:

● нижний уровень автоматизации ТП - автоматизированные системы диспетчерского управления SCADA (Supervisory Control And Data Acquisition), контроллеры и распределенные системы управления DCS (Distributed Control Systems) и разработанные на их основе АСУТП, автоматизированные системы управления объектами электротехнического оборудования (АСУЭТО), автоматизированные системы коммерческого учета электроэнергии (АСКУЭ), тепла (АСКУТ), газа (АСКУГ) и др.;

● средний уровень оперативного управления - системы оперативного управления производством MES (Manufacturing Execution System), в состав которых иногда включают системы управления фондами предприятия EAM (Enterprise Asset Management System);

● верхний уровень бизнес_систем управления ресурсами предприятия ERP (Enterprise Resource Planning System).

В дополнение к классической концепции информационной системы промышленной автоматизации, на стыке MES с системами нижнего уровня находятся системы MIS.

Зарубежные авторы наиболее часто обозначают данной аббревиатурой следующие понятия: Manufacturing Information System (информационная система производства) и Management Information System (информационная система управления), реже - Manufacturing Intelligence Systems (интеллектуальная система управления производством). Несмотря на, казалось бы, явные отличия в наборе используемых терминов, в рамках производственных предприятий все они объединяются одним общим определением:

MIS - это система сбора, обработки, хранения и трансляции данных о внутренних операциях и внешних событиях ТП производства, которая обеспечивает своевременный доступ и предоставление информации в соответствующем формате, необходимом для организации контроля, планирования и оперативного управления.

Таблица рис.16.2 представляет другой пример, в котором существующая программа может проверяться с целью подсчета количества функциональных точек. Например, если в вашем распоряжении имеется приложение, состоящее из 500 строк SLOG на C++, то исходя из этой таблицы, можно сделать вывод что в данном случае имеется 6 * 500 = 3000 функциональных точек.

Данная техника, именуемая "обратным огнем", может использоваться для создания приближенной оценки размеров целого набора приложений. Этот набор может выступать в виде интенсивно используемой хронологической базы данных, которая может применяться при оценке будущих проектов, а также для моделей калибровки размеров и оценки.

Какое количество функциональных точек должно содержаться в системе, чтобы она считалась очень большой? Некоторые крупномасштабные военные приложения содержат примерно 400 тыс. функциональных точек, полная система SAP/R3 - 300 тыс. функциональных точек, Windows 98 включает примерно 100 тыс. функциональных точек, and MVS фирмы IBM тоже около 100 тыс. функциональных точек. Фирма Software Productivity Research, Inc., сгенерировала эти данные на основе своей базы данных, включающей 8500 проектов из более чем 600 организаций.

Метод точек свойств

Метод точек свойств представляет собой расширение метода функциональных точек, применяемого для измерений в приложениях различного типа, даже таких, как встроенные системы и/или системы реального времени. В 1986 году компания Software Productivity Research адаптировала метод анализа точек свойств в отношении системного ПО. Чистые подсчеты по методу функциональных точек, выполненные по отношению к ПО, которое не принадлежит классу MIS, могут давать неправильные результаты.

Это связано с тем, что приложения, как правило, неоднородны и могут иметь большую степень алгоритмической сложности, но при этом обладать простыми внешними вводами/выводами.

Точка свойства представляет собой новую категорию функции, которая может представлять сложные алгоритмы и управление (вызов / отклик).

Сложность алгоритма определяется в терминах количества "правил", необходимых для выражения алгоритма.

Точки свойств обычно применяются в следующих случаях:

- ПО, функционирующее в режиме реального времени;
- системное ПО (например, операционные системы и компиляторы);
- встроенное ПО, например, пакеты навигационных радаров или микропроцессоры автомобильных подушек безопасности;

● инженерные приложения, например, системы автоматизированного проектирования (САПР), автоматизированного производства (АЛ), а также математическое ПО;

● системы искусственного интеллекта (ИИ);

● ПО, предназначенное для поддержки телекоммуникаций (например, системы поддержки телефонных коммуникаций);

● ПО, выполняющее функции контроля над процессами (например, управление автоматизированными нефтеперегонными заводами). Точки свойств, по сути, являются функциональными точками, но в отличие от последних они лучше адаптированы к высокой сложности алгоритмов. В данном случае под алгоритмом подразумевается связанный набор правил (выполняемых операторов), требуемых для решения вычислительных проблем.

Инструкции по подсчету с помощью точек свойств

Шаг 1. Подсчет с помощью точек свойств

Этот шаг аналогичен шагу, выполняемому при подсчете функциональных точек, - учитываются вводы, выводы, файлы (структуры данных), опросы и интерфейсы.

Шаг 2. Продолжение подсчета с помощью точек свойств путем учета количества алгоритмов.

Любой алгоритм представляет собой связанную вычислительную проблему, которая включена в состав конкретной программы.

Значимые и исчисляемые алгоритмы могут разрешать определенные и в принципе решаемые связанные проблемы, для которых существует одна точка входа и одна точка выхода.

Разработчики, которые на этапе разработки проекта часто имеют дело со схемами информационных потоков либо со структурными диаграммами, сравнивают алгоритмы со спецификациями базового процесса либо модуля.

Шаг 3. Сложность взвешивания.

Используйте "средние" веса вместо простых, средних либо сложных (обратите внимание, что понятие "средний" для точек свойств отличается от "среднего" в случае с функциональными точками) для вводов, выводов, файлов (структур данных), опросов и интерфейсов. При взвешивании алгоритмов применяются простые, средние и сложные множители.

Средний фактор сложности для "файлов" уменьшается с 10 до 7, что отражает уменьшение значимости логических файлов по сравнению с ПО, предназначенным для выполнения интенсивных вычислений.

По умолчанию весовой множитель для алгоритмов равен 3. Это значение может

варьироваться в диапазоне от 1 до 10. Алгоритмам, использующим базовые арифметические операции и некоторые правила принятия решений, присваивается значение 1. Алгоритмам, в состав которых включены сложные уравнения, матричные операции и сложная логическая обработка, назначается значение 10. Значимые алгоритмы, которые должны быть исчисляемыми, имеют следующие характеристики:

● представлять решаемую, связанную и определенную проблему;

● обладать свойством конечности;

● быть точным и однозначным;

● иметь ввод или начальное значение;

● иметь вывод или продуцировать результат;

● являться реализуемым - каждый шаг может выполняться на компьютере;

● обеспечивать возможность представления с помощью одной из стандартных программных конструкций: последовательность, конструкция if-then-else, do- case, do-while и do-until.

Шаг 4. Оценка факторов среды

Вместо 14 факторов среды (как в случае с анализом функциональных точек), в данном случае используются лишь два фактора: сложность логики и сложность данных.

Диапазон значений варьируется от 1 до 5.

Логические значения

1. простые алгоритмы и вычисления;

2. большинство простых алгоритмов;

3. алгоритмы средней сложности;

4. некоторые сложные алгоритмы;

5. многие сложные алгоритмы.

Значения данных

1. простые данные;

2. числовые переменные, но простые взаимосвязи;

3. несколько полей, файлов и интерактивных взаимодействий;

4. сложные файловые структуры;

5. очень сложные файлы и взаимосвязи между данными.

При суммировании факторов сложности логики и данных получаем значения, находящиеся в диапазоне от 2 до 10.

Шаг 5. Вычисление фактора корректировки сложности.

Воспользуйтесь таблицей рис. 16.10 для вычисления фактора корректировки сложности.

Сумма степеней сложности логики и данных	Фактор корректировки сложности
2	0,6
3	0,7
4	0,8
5	0,9
6	1,0
7	1,1
8	1,2
9	1,3
10	1,4

Рисунок 16.10 - Фактор корректировки сложности метода точек свойств

Шаг 6. Результат умножения физического количества точек свойств на фактор корректировки сложности.

Шаг 7. Преобразование в количество строк кода с помощью таблицы преобразования функциональных точек (дополнительно).

Бланк рабочего листа (таблица на рис.16.11), применяемый при методе точек свойств, служит примером, иллюстрирующим каждый из 7 упомянутых шагов.

Заполненный рабочий лист представлен в таблице на рисунке 16.12.

Факторы	Среднее значение	Точки свойств
Шаг 1. Подсчет количества точек свойств		
Количество вводов	x4	
Количество выводов	x5	
Количество файлов (структуры данных)	x7	
Количество опросов	x4	
Количество интерфейсов	x7	
Шаг 2. Подсчет количества алгоритмов		
Количество алгоритмов	x3	
Шаг 3. Сложность взвешивания		
Шаг 4. Оценка факторов среды корректировки сложности (CAF)		
Шаг 5. Вычисление фактора		
Логические значения (выбор одного из них)1		
Простые алгоритмы и вычисления	1	
Большинство простых алгоритмов	2	
Алгоритмы средней сложности	3	
Некоторые сложные алгоритмы	4	
Многие сложные алгоритмы — 5	5	
Значения данных (выбор одного из них)		
Простые данные	1	
Числовые переменные, но простые взаимосвязи	2	
Несколько полей, файлов и интерактивных взаимодействий	3	
Сложные файловые структуры	4	
Очень сложные файлы и взаимосвязи между данными	5	
Итого по CAF:		
Шаг 6. Умножение физического количества точек свойств		
Физические FP * CAF =		
Шаг 7. Преобразование в строки кода (дополнительно)		
LOC =AFP * LOC/AFP =		

Рисунок 16.11 - Бланк рабочего листа анализа методом точек свойств

Факторы	Среднее значение	Точки свойств
Шаг 1. Подсчет количества точек свойств		
Количество вводов	12x4	48
Количество выводов	15x5	75
Количество файлов (структуры данных)	22x7	154
Количество опросов	17x4	68
Количество интерфейсов	8x7	56
Шаг 2. Подсчет количества алгоритмов		
Количество алгоритмов	43x3	129
Итого по точкам свойств: физических точек свойств		**530**
Шаг 3. Сложность взвешивания		
Шаг 4. Оценка факторов среды корректировки сложности (CAF)		
Шаг 5. Вычисление фактора		
Логические значения (выбор одного из них)1		
Простые алгоритмы и вычисления	1	
Большинство простых алгоритмов	2	
Алгоритмы средней сложности	3	
Некоторые сложные алгоритмы	**4**	
Многие сложные алгоритмы — 5	5	
Значения данных (выбор одного из них)		
Простые данные	1	
Числовые переменные, но простые взаимосвязи	2	
Несколько полей, файлов и интерактивных взаимодействий	**3**	
Сложные файловые структуры	4	
Очень сложные файлы и взаимосвязи между данными	5	
Итого по CAF: 4+3=7 – фактор корректировки сложности		
Шаг 6. Умножение физического количества точек свойств		
Физические FP * CAF = 530*1,1 (из таблицы рис.16.10) =**583** скорректированных точек свойств		
Шаг 7. Преобразование в строки кода (дополнительно)		
LOC (для языка Java) =AFP * LOC/AFP = 583*53 (из таблицы рис.16.2)=**30899LOC**		

Рисунок 16.12 - Пример рабочего листа анализа методом точек свойств

Преимущества анализа методом точек свойств.

Анализ точек свойств обеспечивает те же самые преимущества, что и анализ функциональных свойств. Причем при этом обеспечивается улучшенный подход к использованию оценки размера алгоритмически насыщенных систем.

Недостатки анализа методом точек свойств

Основным недостатком анализа точек свойств является субъективный подход к классификации алгоритмической сложности.

Метод объектных точек

Подход с использование подсчета "объектных точек" для определения размера ПО позаимствован из объектно-ориентированной технологии. При использовании этого подхода оценка выполняется на более обобщенном уровне, чем в случае с функциональными точками. В данном случае каждому уникальному классу или объекту (экран, выводимый отчет и т.д.) назначается одна объектная точка. В остальном этот метод подобен методу функциональных точек и точек свойств, лишь следует учесть, что применяются различные факторы преобразования.

Блиц-модель.

Оценка, выполняемая на каждой фазе проекта, будет более качественной в случае, если доступен больший объем исходных данных. Влияние исходных знаний особенно хорошо заметно на стадии анализа и проектирования, на которых обычно формируются модели (глава 22). Здесь особенно хорошо заметно, что от количества начальных данных зависит точность оценки размера ПО. Еще до того, как будет достигнута текущая стадия проекта, могут оказаться весьма полезными некоторые модели высокого уровня, созданные на фазе планирования. Эти модели могут применяться в качестве быстрого и простого метода оценки размера ПО.

Концепция блиц-моделирования основана на банг-метрике. В этом случае грубые оценки размера программного кода определяются путем подсчета компонентных частей системы (элементов дизайна) и дальнейшим умножением результата на множитель производительности (как правило, этот множитель определяется на основе хронологических данных и определяет количество строк процедурного кода, необходимых для реализации алгоритма).

Например, если данные высокого уровня в схеме последовательности операций либо в объектных моделях генерируются в качестве части исследования концепции или планирования, их компоненты могут быть просмотрены на предмет их размера.

Представьте себе, что 20 классов объектов и их характеристики получаются путем наблюдения существующих систем и реализованы как средние величины в качестве пяти процедурных программ из расчета на один класс. Также путем наблюдения за существующими системами стало известно, что средний размер процедурной программы (язык С) равен 75 LOC. Затем размер может быть быстро вычислен следующим образом:

Количество процессов (классы объекта) * количество программ из расчета на класс

**** размер средней программы = расчетный размер***

20 классов объекта * 5 программ на класс * 75 строк LOC на программу = 7500 вычисленных LOC

Приведенные выше выкладки известны как "блиц", примененный по отношению к документам, созданным на ранних стадиях. Компоненты любой модели ("пузырьки" процесса, потоки данных, репозитории данных, сущности, взаимосвязи, объекты, атрибуты, службы и т.д.) могут умножаться на множитель, который вычисляется как результат выполнения предыдущих проектов.

А теперь рассмотрим другие примеры. Если известно, что каждый "пузырек" процесса на уровне 0 **DFD,** приблизительно соответствует четырем фактическим программам на языке SQL, а также известно, что средний размер программ в библиотеке SQL равен 350 строкам LOC, поэтому простое умножение будет достаточным для подсчета начального размера. Известно, что в данном случае существует семь основных "пузырьков" процесса:

***Примечание**. Метод совместной разработки приложений (JAD) - это зарегистрированный товарный знак, относящийся к компании IBM. Сеанс JAD имеет определенную структуру, на нем придерживаются определенной дисциплины, и он проходит под руководством арбитра. В его основе лежит обмен информацией с использованием документации, фиксированных требований и правил работы. С момента появления методики JAD на сеансах используются CASE- инструменты и другие программные средства, предназначенные для построения **диаграмм потока данных (Data flow diagram, DFD)**, диаграмм взаимосвязей между сущностями (Entity relationship diagrams, ERD), диаграмм смены состояний и других объектно-ориентированных диаграмм.*

Количество процессов ("пузырьки" DFD) * количество программ на "пузырек" * размер средней программы = вычисленный размер

7 "пузырьков" * 4 программы на "пузырек" * 350 строк LOC на программу 9800 вычисленных строк LOC

Если глобальная модель высокого уровня продуцируется на этапе определения времени выполнения, а на основе исторического опыта известно, что каждая служба соответствует двум программам на языке C++, а стандарты компании ограничивают каждый размер служебного пакета величиной в 100 строк LOC и менее, приведенная ниже формула обеспечивает хорошую начальную базу для оценивания размера систем, выраженного в количестве строк LOC: ***number of services * 2 * 100***

В данном случае ключевой является фраза "на базе исторического опыта". База данных,

содержащая данные за достаточно долгий период времени, является весьма важной в деле повышения точности оценки. При этом база данных должна содержать запись, соответствующую фактическому размеру каждого программного компонента.

Объем трудозатрат, понесенных на создание компонента данного размера, также должен отслеживаться. По мере роста количества точек данных, улучшается степень соответствия среднего количества строк LOC из расчета на программу со средним объемом трудозатрат, понесенных при создании компонента. Как только станут известными размеры фактических компонентов и величины объемов соответствующих усилий, затраченных на этапе разработки, становится известной также средняя "производительность" (размер/трудозатраты).

При использовании банг-метрики системы, использующие множество функций (в том числе и системы реального времени), должны оцениваться отдельно от систем, обрабатывающих большие объемы данных. При оценке функционально богатых систем в качестве базисных значений используется количество неделимых функциональных примитивов, как определено с помощью схемы информационных потоков. Системы, обрабатывающие большое количество данных, в качестве базисных значений используют количество объектов в модели данных, глобальных на системном уровне. При этом также применяются весовые множители (Weighting factor, WF).

Ниже приводится пример функционально богатой системы: множитель WF (среднее количество модулей, требуемых для завершения функции) равен трем, количество процессов плюс спецификации контроля (функции) равно восьми, а средний размер одной функции равен 78 LOC. Затем воспользуемся следующей формулой:

WF * (количество процессов и спецификаций контроля) * среднее количества LOC для данного модуля = LOC

3 модуля, требуемые для функции * 8 функций * 78 LOC = 1,872 LOC

Чем же все это отличается от анализа методом функциональных точек, представленного на фазе определения области осуществимости? Менеджер проекта может выбрать выполнение анализа точек свойств во время фазы определения области выполнения, когда существуют только модели высокого уровня, такие как DFD уровня содержимого. Затем полученная оценка улучшается на фазе планирования, когда доступен больший объем знаний о проекте и дополнительная документация, например DFD уровня 0 наравне с DFD уровня 1 для некоторых из основных подсистем. Любые из этих моделей могут применяться на протяжении любой фазы. Если они применяются последовательно, повышается ожидаемая точность измерений и получения оценок.

Преимущества блиц-модели

● Облегчается использование структурных систем (схемы информационных потоков, диаграммы взаимосвязи сущностей) совместно с объектно-ориентированными классами, службами и т.д.

● Повышается степень точности при использовании хронологических данных.

● Действия по непрерывному улучшению используются при реализации техники оценки.

● Оцененный размер может быть легко сопоставлен с фактическим размером при выполнении постпроектного анализа. (Насколько точной была оценка? Каков процент возможного отклонения? Что потребуется освоить при оценке размера следующего проекта?)

Недостатки блиц-модели

● Требуется использование методологии дизайна.

● Оценка не может начинаться до завершения разработки дизайна.

● Требуются хронологические данные.

● Не могут оцениваться факторы среды.

Метод оценивания Wideband Delphi

Еще одним популярным и простым методом, применяемым при оценке размера ПО и величины трудозатрат, является сбалансированный подход, разработанный группой Wideband Delphi. Техника Delphi изначально была разработана в Rand Corporation; само название позаимствовано из греческой мифологии (знаменитый Оракул в Дельфийском храме). Этот метод был успешно использован в корпорации Rand с целью прогнозирования развития большинства технологий в мире.

Этот метод основывается на использовании опыта многих экспертов с целью выполнения оценки, которая отражает всю сумму их знаний.

В среде разработчиков ПО исходный подход Delphi был модифицирован. "Чистый" подход используется при сборе изолированных мнений экспертов, обратной связи, в ходе реализации которой отсылаются анонимные итоговые результаты, а также при выполнении итераций до тех пор, пока не будет достигнут консенсус (без проведения групповых дискуссий).

Инструкции по достижению консенсуса согласно модели группы Wideband Delphi.

Поскольку реализация подхода Delphi требует очень больших затрат времени, разработанная концепция Wideband Delphi предназначалась для ускорения этого процесса. Улучшенный подход использует групповые дискуссии.

Шаги по достижению консенсуса согласно модели Wideband Delphi

1. Представление проблемы и ответной формы на рассмотрение экспертов.

2. Проведение групповой дискуссии.

3. Анонимный сбор мнений экспертов.

4. Обратная связь по сводке результатов с каждым экспертом.

5. Поддержка других групповых дискуссий.

6. Выполнение итераций (при необходимости) до тех пор, пока не будет достигнут консенсус.

Основное отличие между "чистым" методом Delphi и Wideband Delphi заключается в том, что в последнем случае проводятся групповые дискуссии. Сводка по результатам, достигнутым на шаге 4, представлена в таблице рис.16.13.

Название проекта: _ _					Дата:		
Оценка диапазона размеров из предыдущего периода оценки							
Ваша оценка 5,000 LOC			Средняя оценка 8,000 LOC				
5,000	6,000	7,000	8,000	9,000	10,000	11,000	12,000
Пожалуйста, укажите вашу оценку для следующего периода либо состояние, которое является причиной остающейся предыдущей оценки							

Рисунок 16.13 - Сводная форма результатов оценки размеров ПО по методу Delphi

Ниже представлен другой взгляд на процесс Wideband Delphi.

Опросите нескольких экспертов (обычно их число варьируется от трех до пяти).

Включите их опыт во все области "риска"- область приложения, язык программирования, алгоритмы, целевое аппаратное обеспечение, операционные системы и т.д.

Организуйте встречи с экспертами для обсуждения вопросов и объяснения деталей функционирования ПО. Подготовьте спецификации, другие исходные документы, структуру WBS и т.д. Позвольте им дополнить все это собственной информацией и вопросами. Пусть каждый из них сделает свои замечания.

Попросите каждого эксперта выполнить свою оценку, включающую минимальный, ожидаемый и максимальный рейтинг. Разрешите им оставаться независимыми и анонимными.

Запишите анонимные оценки в виде графа.

Встретьтесь с каждым экспертом и позвольте ему высказать в дискуссии мнение

относительно его оценки, допущениях и причинах выполнения подобной оценки.

Найдите консенсус по поводу допущений. Это отразится на элементах действия, выполняемых с целью получения фактических данных.

Если возможно, выполните оценку достижения консенсуса.

● Если консенсус не был достигнут, прервитесь до тех пор, пока не сможете получить доступ к дополнительным данным; затем повторите.

● Прекратите повторение в случае, если достигнете консенсуса, либо на протяжении двух последовательных циклов не будет внесено больших изменений, а также не будет появляться значительный объем свободных дополнительных данных (с которыми можно будет согласиться либо нет). В итоге будет достигнут консенсус при оценке ожидаемого значения. Также при выполнении оценки должен учитываться максимальный и минимальный объем возможной конфиденциальности.

Преимущества модели Wideband Delphi

● простая и недорогая реализация на практике;

● экспертиза проводится несколькими специалистами в своей области;

● все участники обсуждения повышают свою квалификацию в области рассматриваемого ПО;

● не требуются хронологические данные, хотя они могут применяться в случае их доступности;

● используется на высшем уровне и при выполнении детализированной оценки;

● результаты будут более точными и менее "опасными", чем в случае использования оценки по методу LOC;

● обеспечивается поддержка глобальной точки зрения на проект со стороны членов команды.

Недостатки модели Wideband Delphi

● затруднительная повторяемость различными группами экспертов;

● существует вероятность достижения консенсуса для некорректной оценки. Поскольку вы "все выкупили", вы не сможете быть достаточно критичными в случае, если фактические данные будут выглядеть некорректными;

● вас может посетить ложное чувство самоуверенности;

● иногда вы не сможете достичь консенсуса;

● все эксперты могут работать в одном и том же направлении.

Влияние эффектов повторного использования на размер ПО

Многие программы происходят от предыдущих версий этих же программ. В результате может быть достигнута экономия средств и/или времени, а также возрасти уровень качества. Однако при повторном использовании могут расходоваться дополнительные средства и время, а уровень качества будет низким.

Повторному использованию подлежат: код, тестовый код, тестовые условия, тестовые процедуры, документация, дизайн, спецификации дизайна, спецификации требований и т.д.

Терминология повторного использования:

● ***Новый код*** - это код, разработанный для нового приложения, который не включает большие порции ранее написанного кода.

● ***Модифицируемый код***- это код, разработанный для предыдущих приложений, который становился пригодным для использования в новым приложениях после внесения умеренного объема изменений.

● ***Повторно используемый код*** - это код, разработанный для предыдущих приложений, который будет пригодным для новых приложений без внесения каких-либо изменений.

● ***Наследственный код*** - это код, разработанный для предыдущих приложений, использование которого ожидается новым приложением.

Почему же код считается "модифицируемым" в случае, если он будет подвергаться лишь незначительным изменениям? Код, повторно используемый, в полном объеме, имеет идентичную документацию, идентичные тестовые процедуры и код, но лишь одну копию с целью поддержки библиотеки менеджмента конфигурирования. Даже если изменяется единственная строка комментария, две копии кода, тестовый код, документация и прочее должны поддерживаться в библиотеке управления конфигурированием. Если изменяется лишь одна строка исполняемого кода, подвергается изменению тестовый код и документация.

При использовании наследственного кода имейте в виду, что он может быть снабжен документацией плохого качества, могут отсутствовать тестовый код либо процедуры. Для этого кода может отсутствовать тщательно проработанный проект, и он может не отвечать стандартам качества.

Первый этап при оценке систем, которые могут повторно использовать код, заключается в отделении нового кода от модифицируемого и повторно используемого кода. Это необходимо по той причине, что модифицируемый и повторно используемый код практически никогда не может быть использован непосредственно, - для выполнения интеграции этого кода требуются некоторые изменения, в результате чего возрастает размер кода и объем трудозатрат, требуемых для реализации подобных изменений. Соответствующий пример приводится в

таблице на рис.16.14.

Элемент	LOC для нового кода	LOC для модифицируемого кода	LOC для повторноиспользуемого кода	Итого по LOC
Компонент 1	1233	0	0	1233
Компонент 2	0	988	0	988
Компонент 3	0	0	781	781
Компонент 4	560	245	0	805
Компонент 5	345	549	420	1314
Итого	2138	1782	1201	5121

Рисунок 16.14 - Разделение новых, модифицируемых и повторно используемых строк кода

А теперь возникает вопрос о том, как можно узнать, в какой степени компонент может быть модифицируемым или повторно используемым? Обратимся к компонентам 4 и 5 в таблице рис.16.15. Правило "большого пальца" говорит о необходимости проверки наименьшего уровня, известного для единицы либо модуля (обычно около 100 LOC). Если единица в целом не была изменена, она может "повторно использоваться". Если единица была изменена (пусть даже речь идет об одном комментарии или выполняемом операторе), она является "модифицируемой". Если более чем 50% кода единицы было изменено, рассматривайте ее как "новую". Кроме того, как только модифицируемый код был идентифицирован, его можно разбить на категории, представляющие тип модификации. Широко используемый подход заключается в том, чтобы отделять модификации, предназначенные для корректировки дефектов, от модификаций, направленных на добавление расширений. Как видно, таблицы рис.16.14 и 16.15 весьма похожи за исключением одного выполненного разделения.

Элемент	LOC для нового кода	Модифицировано для устранения ошибок	Модифицировано с целью добавления расширений	LOC для повторноиспользуемого кода	Итого по LOC
Компонент 1	1233	0	0	0	1233
Компонент 2	0	988	0	0	988
Компонент 3	0	0	0	781	781
Компонент 4	560	245	245	0	805
Компонент 5	345	549	247	420	1314
Общее количество произведенных LOC	2138	1782	492	1201	5121

Рисунок 16.15 - Различные типы модифицируемого кода

После завершения оценки общего объема разработанного программного кода повторно используемый код будет преобразован в "эквивалентный" новый код. В ходе процесса преобразования ставится цель минимизировать объем трудозатрат при внедрении повторно используемого или модифицируемого кода по сравнению с внедрением нового программного кода. Обычно определяется множитель преобразования с целью отражения количества трудозатрат, сэкономленных при повторном использовании кода. Предполагая наличие хронологического выравнивания множителей преобразования, можно заключить, что в этом случае выполняется простая калькуляция.

Возвращаясь к примеру рис.16.16, можно применить множители повторного использования, свидетельствующие о том, что при внедрении повторно используемого ПО экономится примерно 70% трудозатрат по сравнению с внедрением нового ПО.

Элемент	LOC для нового кода	LOC для модифицируемого кода	LOC для повторноиспользуемого кода	Итого по LOC
Итого	1238	1782	1201	5121
Множитель	100%	60%	30%	-
Результат	2138	1069	360	3567

Рисунок 16.16 - Применение множителей повторного использования

При использовании модифицируемых программ экономия составляет примерно 40%. Это говорит о том, что объем трудозатрат, требуемых при написании 5121 строк повторно используемого, модифицируемого и нового кода эквивалентен объему трудозатрат в случае написания 3567 строк нового кода.

Множители повторного использования вычисляются на основании опыта.

Множители, равные 30% (трудозатраты на создание повторно используемого кода) и 60% (трудозатраты на создание модифицируемого кода) наблюдаются в сотнях проектов. Однако в данном случае речь идет о средних значениях и возможный диапазон будет шире. Наилучшим индикатором достигаемого размера и понесенных трудозатрат в вашей организации являются фактические данные, полученные в ней, - отслеживайте оценки и фактические данные в хронологической базе данных. Типичные множители повторного использования приведены в таблице рис. 16.17.

Степень простоты использования	Повторно используемый, %	Модифицируемый, %
Простой	10	25
Средний	30	60
Сложный	40	75

Рисунок 16.17 - Типичные множители повторного использования

Будьте особенно внимательны при повторном использовании кода. Можно достичь большей точности при повторном использовании кода, если более внимательно присмотреться к процессу и повторно использовать характеристики.

Возвращаясь к нашему примеру, заметим, что первый шаг заключается в проверке процесса и в определении процента от общего количества трудозатрат, понесенных на каждом шаге разработки нового кода.

Предположим, нам известно о том, что данная организация тратит 18% своего времени на разработку требований, 25% времени - на дизайн, 25% времени - на кодирование и тестирование, а 32% времени - на выполнение интеграции (в данном примере рассматриваются лишь четыре фазы жизненного цикла).

Как показано в таблице рис. 16.18, на разработку нового кода уходит максимальное количество трудозатрат. В случае применения повторно используемого и модифицируемого кода трудозатраты будут меньшими. Отметим, что, вместо обычных 100%, модифицируемый код требует лишь 20% трудозатрат на этапе формулировки требований, а повторно используемый

код - лишь 10% трудозатрат.

Вместо 100% трудозатрат на этапе разработки проекта, модифицируемое ПО требует 40% трудозатрат, а повторно используемое ПО вообще не нуждается в разработке проекта (0%). Показатель 40% появляется по той причине, что потребуется разработать план тестирования модифицируемого ПО, а также вполне возможно, что придется специальным образом переработать часть ПО. Таким образом отчетливо просматривается важность повторного использования.

Вместо 100% трудозатрат на этапе кодирования и тестирования единицы, необходимых при разработке новых программ, модифицируемое ПО требует лишь 70% трудозатрат, а повторно используемое ПО является полностью "свободным" от трудозатрат. При использовании "чистого", повторно используемого ПО данный показатель равен нулю. Если будет изменена лишь одна строка кода, все ПО будет считаться модифицируемым.

Трудозатраты по интеграции кода не будут сильно уменьшены даже в случае применения модифицируемого или повторно используемого кода. Даже в случае чистого повторно используемого кода трудозатраты по интеграции часто составляют 50%-100%. На каждой фазе процесса эффект от повторного использования кода может определяться после того, как установлено процентное значение для каждой повторно используемой категории.

В таблице рис. 16.18 демонстрируется, что включение повторно используемого либо модифицируемого кода в любом случае может привести к экономии размера, трудозатрат, времени по графику или затрат денежных средств.

	Шаг процесса	Требования	Дизайн	Код	Интеграция	
	Процент	18%	25%	25%	32%	
Произведено	Новый	100%	100%	100%	100%	Эквивалент
2138	Модифицируемый	20%	40%	70%	100%	2138
1782	Повторно используемый	10%	0%	0%	100%	1124
1201						406
5121						3668

Рисунок 16.18 - Применение метода точной оценки в повторно используемом или модифицируемом коде

Оценка трудозатрат

После того, как размер программного продукта был оценен, можно перейти к оценке трудозатрат, понесенных на производстве этого продукта.

Резюме

Измерение размера, оценка и составление графика сложным образом переплетаются в процессе планирования проекта. На самом деле довольно сложно создать реальный график (учитывающий обязанности исполнителей, зависимости, перекрытие действий, а также дату поставки продукта) без информации об объеме трудозатрат, требуемых для выполнения каждой задачи (например, определения нагрузки сотрудников на месяц). Достаточно трудно оценить объем трудозатрат, необходимых для выполнения задачи, без информации относительно ее "размера". Таким образом, измерение предшествует оценке, а оценка, в свою очередь, предшествует составлению графика. Каждое из этих важных действий проекта может быть выполнено с помощью множества различных методик. Способность к выполнению оценки является весьма важной для зрелой организации, выполняющей разработку ПО.

Плохие оценки влекут проблемы и увеличивают степень риска. Негативные последствия могут быть в значительной степени смягчены при использовании стандартных методов оценки. В начале работы с любыми методами следует получить хорошее представление о пооперационной разбивке проекта, как в случае со структурой WBS.

При выполнении операции определения размеров используются пять весьма полезных и широко известных технологий, каждая из которых создана на базе структуры WBS:

● подсчет строк кода (LOC) в качестве единиц оценки размера;

● подсчет функциональных точек в качестве единиц оценки размера;

● подсчет точек свойств в качестве единиц оценки размера;

● применение блиц-модели (также известной под именем банг-метрики и оценивания "снизу-вверх");

● применение модели Wideband Delphi.

Повторное использование существующих компонентов не всегда возможно в полном объеме. Существует так называемая плата за интеграцию: особенности существующих компонентов может не допускать обеспечение качества и возможность повторного использования. Выход состоит в использовании набора весовых множителей, произведенных на основе эмпирических правил, с целью их применения при оценке процесса повторного использования.

Индустрия ПО часто возвращается к использованию метрики LOC, единицы измерения, хорошо знакомой практикам в области разработки ПО. Они находят ее комфортной и простой в применении. Какая бы технология не использовалась, оценка размера продукта является весьма важной, поскольку она является составной частью одной наиболее важной задачи проекта: установка реальных ожиданий.

Нереальные ожидания, основанные на неточных оценках, представляют собой одну из наиболее частых причин сбоев ПО. Зачастую причина кроется не в недостаточной производительности команды проекта, а в неточном предвидении уровня этой производительности. Точное оценивание обеспечивает улучшенный контроль над проектом и является жизненно важным в деле проектного менеджмента; неточное оценивание приводит к возникновению паники "в последние минуты", что ведет к появлению дефектов.

Для обеспечения точного оценивания в первую очередь следует иметь представление относительно размера продуцируемого ПО. Эта величина должна определяться на ранних стадиях цикла разработки и выражаться в единицах, которые являются простыми в процессе наблюдения и при осуществлении коммуникаций.

Контрольный список оценивания может включать следующие аспекты:

● Ставилась ли при основании фирмы цель выполнения оценки размера ПО?

● Были ли обучены заказчики, менеджеры и разработчики основам измерений и получения оценки?

● Были ли корректно применены хронологические данные?

● Применялись ли корректно модели?

● Был ли адаптирован метод оценки к процессу разработки?

● Были ли выполнены приемлемые допущения относительно факторов продукта и процесса, оказывающих влияние на производительность?

● Применялись ли реальные стратегии, способствующие достижению агрессивных целей?

● Использовался ли альтернативный метод оценивания в целях сравнения?

● Были ли применены несколько методов? (В большинстве случаев достаточно одного метода. Если оба метода не подходят, могут быть утеряны важные факты.)

● Были ли задействованы все используемые функции? (Многие из функций легко пропустить или недооценить, например, контроль, включение электропитания, сбой электропитания, диагностика, операционные системы и т.д.)

● Удалось ли вам избежать выполнения той работы, в которой вы плохо разбираетесь? При оценке размера ПО важно помнить о следующем:

- хронология является вашим "союзником";

- используйте несколько методов с целью их изучения и снижения степени риска;

- учитывайте возможность повторного использования.

Практика

Вопросы для самопроверки

1. Определить количество строк кода, если пессимистическая оценка размеров = 500 LOC, оптимистическая оценка размеров = 3500 LOC, реалистическая оценка размеров = 2000 LOC.

2. Определить количество строк программы на языке Java, если известно что эта же программа содержит 500 LOC на языке C.

3. Определите размер ПО, если 18 классов объектов и их характеристики получаются путем наблюдения существующих систем и реализованы как средние величины в качестве 3 процедурных программ из расчета на один класс. Также путем наблюдения за существующими системами стало известно, что средний размер процедурной программы (язык C) равен 40 LOC.

4. Определите размер ПО, если известно, "пузырьков"=5 и каждый "пузырек" процесса на уровне 0 DFD, приблизительно соответствует 2 фактическим программам на языке SQL, а также известно, что средний размер программ в библиотеке SQL равен 100 строкам LOC.

5. Определите размер ПО, в функциональной системе среднее количество модулей, требуемых для завершения 8 функций равно 3, размер одной функции равен 100 LOC.

6.Определите количество строк кода программы на языке DELPHI (LOC), если дано общее количество функциональных точек = 100, сумма взвешенных факторов среды =20.

7. Определите количество строк кода программы на языке C (LOC), если в программе будут использоваться простые алгоритмы и простые данные, общее количество физических точек = 100.

Вывод к разделу 16 - Оценка размера и возможности повторного использования ПО

В разделе рассматривались:

- Единицы измерения, используемые при оценке размера ПО.
- Риски, связанные с оценкой размера разрабатываемого ПО.
- Метод оценки показателя LOC с помощью экспертных заключений и восходящего суммирования, его преимущества и недостатки.
- Метод функциональных точек, его преимущества и недостатки, заполнение рабочих листов.
- Метод точек свойств, его преимущества и недостатки, заполнение рабочих листов.
- Метод оценивания Wideband Delphi.
- Метод блиц-модели.

Перечень ссылок

Источники, использованные в материалах

Американский национальный стандарт ANSI/PMI 99-001-2004. Руководство к Своду знаний по управлению проектами. Введ. 2004.- Третье издание. (Руководство PMBOOK®). 401с.

Мари Кантор. Управление программными проектами. Практическое руководство по разработке успешного программного обеспечения . СПБ. Вильямс. 2002. -642с.

Управление программными проектами. Достижение оптимального качества при минимуме затрат. Роберт, Т. Фатрелл, Дональд Ф. Шафер, Линда И. Шафер / М-СПб-К. Вильямс. 2003. -1118с.

Уокер Ройс. Управление проектами по созданию программного обеспечения . М. Лори. 2002. -450с.

Элейн Маркел. Microsoft Project 2002. Библия пользователя. М. Диалектика. 2003. -880с.

Microsoft Project 2003 course certification materials. Trainer kit. [Электронный ресурс] Режим доступа: http://www.cheltenhamcourseware.com/