УДК 519
Шамилев Саидбек Руманович
директор, ООО «Издательский дом Интернаука»
Россия, Москва
Shamilev Saidbek Rumanovich
Limited Liability Company "Internauka Publishing House"
Russia, Moscow

ЭВОЛЮЦИОННЫЙ ПОДХОД К ОЦЕНИВАНИЮ СЛОЖНОСТИ ПРОГРАММ И ИХ КОМПЛЕКСОВ
EVOLUTIONARY APPROACH TO ESTIMATION OF COMPLEXITY OF PROGRAMS AND THEIR COMPLEXES

Аннотация. *Развитие программирования, больших и сложных комплексов программ, баз и структур данных привело к необходимости их оценивания – не только сравнительного, но и внутренней структуры и даже логики программ и БД. В условиях отсутствия достаточно эффективных методов верификации программ, необходимости полноценного тестирования, такие оценочные критерии и методика особенно будут актуальны «априори». Цель работы – исследовать существующие подходы, предложить простую и технологичную методику оценивания, хотя бы на уровне начала проектных работ. Использованы методы метрик надежности, эффективности, сложности ПО и методы системного анализа.*

Abstract. *In the conditions of absence enough effective methods of verification of programs, criteria and measures are especially important. The work purpose – research of the existing approaches to offer a simple and technological method of an assessment.*

Ключевые слова: *сложность; метрики; меры; программа; ПО; комплекс; тестирование; оценивание; информация; потенциал*

Keywords: *complexity; metrics; measures; the program; the software; a complex; testing; an assessment; information; potential; checks*

Функции, ресурсы и управление – «три кита» оценивания программ

Повышение требований к качеству программ и комплексов требует соответствующих аналитических инструментов, моделей и методик (технологий). С какими требованиями к используемым ресурсам, функциям и насколько устойчиво и адекватно (управляемо) функционирует программа? – Это основной триединый вопрос. Ответ требует наличия прогнозных моделей, гипотез моделирования и релевантных оценочных мер.

У комплексов программ всегда имеются свои функциональные показатели качества [1], самый общий из них – эффективность. Он актуален независимо от назначения и использования программ. Часто критерии эффективности отражают не саму даже эффективность, а его последствия, например, затраты или устойчивую результативную работу. Поэтому применяют часто классические, эконометрические подходы и экономические критерии (производительность, время, стоимость и др.). Распространен и технический подход – измерять надежность методами оценивания надежности технических комплексов.

Эволюционирующая программная система требует особых, например, синергетических подходов, базирующихся на системном анализе [2]. В теоретическом плане необходимы формально-алгебраические подходы, на практике необходим измерительный, оценочный инструментарий комплексов программ.

Жизненный цикл (ЖЦ) программного обеспечения (ПО) реализуется этапами:

1) специфицирование и проектирование;

2) разработка тестового инструментария (выбор тестового покрытия, метода тестирования – индуктивного, дедуктивного);

3) кодирование (собственно, программирование в узком смысле);

4) тестирование или верификация (формальными методами);

5) отладка и оптимизация;

6) развитие и сопровождение (называется этот этап верно и емко этапом продолжающейся разработки).

Сложность оценивания определяется сложность самой системы «ПО» с учетом абстрагирования-конкретизации, анализа-синтеза, индукции-дедукции, композиции-декомпозиции, формализации-конкретизации, макетирования, моделирования и эксперимента, распознавания и идентификации, классификации и кластеризации, экспертного и эвристического анализа и других методов.

Информационный, эволюционный подход к оцениванию ПО

Важен учет взаимодействий дезорганизационных (например, невнимательность к спецификациям) и организационных (например, модульный подход к проектированию) факторов, мероприятий. Противостояние их образует границу развиваемости ПО. Отклонения функционирования ПО, изменяющие, ухудшающие эволюционные возможности – начало системного сбоя ПО. Нарастающее неравновесие может привести к разрушению связей ПО. Функционирование программ (подсистем) ПО становится рассогласованным, приводит к потере устойчивости и релевантности системы, к необходимости качественно новых требований, к новым отношениям.

При разработке ПО, его совершенствовании и эволюционном развитии, необходимо идентифицировать оптимальные (локально-оптимальные, рациональные) характеристики ПО (как системы) в некотором диапазоне времени. За меру эволюции можно выбрать классическую информационную энтропию [3]. Но она со временем вносит неопределенность, снижая достоверность и надежность характеристик. Необходимо учитывать и идентифицировать приемлемое (оптимальное) время прогнозирования. Ниже рассматривается соответствующая математическая модель.

Существует методы прогнозирования эволюции системы: максимального правдоподобия, марковских цепей, мульти-агентные, многомерного шкалирования и др.

Информационно-энтропийный метод определяет изменения основных прогнозных параметров ПО через приращение энтропии, беспорядка (ошибок прогнозирования). По достижении требуемого ее значения, проводится оценка и переоценка характеристик ПО (например, отладочных) и идентифицируется уровень прогнозирования. Полученные данные – оптимальны, система имеет оптимальное (по упорядоченности, структурированности) соотношение энтропии. Если энтропия предыдущей версии велика, ПО будет недоработано («недоразвито»), это простая модернизация ПО (собственно, инжиниринг, а не реинжиниринг). Если энтропия мала – есть опасения, что новое ПО не будет адаптировано к существующей вычислительной среде, недостаточно преемственно с предыдущей версией ПО. Развития – нет.

Оценка прогнозирования – по принципу максимума энтропии [4]. В результате получаем прогнозные характеристики эволюции ПО на определенный, задаваемый промежуток времени (морального устаревания ПО). На каждом этапе эволюции ПО определяется оценочный эволюционный потенциал или энтропийный потенциал.

Формализуем задачу. Пусть существует схема (процедура) нахождения предпочтения на каждом интервале развития ПО. Оценивая (например, экспертным методом комитетов или методом Дельфи) эффективность решений ЛПР находим множество векторов

$x = (x_1, x_2, ..., x_n) \in R^n$ с нулевой гипотезой H_0, для заданной меры значимости (достаточной для обеспечения нахождения компонент вектора *x,* характеристик ПО).

Решение задачи дается процедурой:

1) находим вектор (размерность *m*) потенциала ПО;

2) каждому промежутку времени сопоставляем совокупность *m* характеристик ПО, определяющих его потенциал на данный промежуток;

3) оцениваем уровень значимости и значение меры прогноза развития ПО;

4) составляем инфологическое (морфологическое) описание [2] или таблицу со строками, соответствующими характеристикам ПО и со столбцами, соответствующими временной шкалой эволюции ПО;

5) снабжаем весами оценочного потенциала в зависимости от количественных мер объективной неопределенности (можно отождествить с распределением вероятностей);

6) нормируем значения показателей (шкалируем в [0;1]), например, относительно граничных значений показателей $r_{ji} = x_{ji} / x_j^{\max}$;

7) составляется матрица принятия решений $R = \|r_{ij}\|$, причем ее элементы можно идентифицировать с вероятностями *p(r)* элементарных событий *{x}* или модулей (функций, операторов языка программирования)

Распределение вероятностей обеспечит оценку гипотезы H_0 на заданном уровне значимости и для заданного критерия прогноза развития ПО.

Информация – мера снимаемой неопределенности, количество информации – вероятность, мера энтропии (согласно формуле Шеннона, например). Поэтому можно применять априорную (апостериорную) и условную вероятности, формулы Байеса, полной вероятности [5].

Получить нормированную меру *p(r)* элементарных событий {*r*}, можно отождествив оценочный потенциал комплекса ПО (их модулей) с функцией принадлежности, ставящей каждому *r* в соответствие число от 0 до 1.

Представим эту зависимость, как и в [6], функцией вида:

$$p_{ji}(r) = \sum_{i=1}^{n} r_{ji}\ .$$

Можно применить метод расчета вероятности наличия *j*-го признака (ошибки ПО) сравниваемых промежутков времени с помощью потенциального распределения вероятности [7]. Необходимо решить задачу условной оптимизации:

$$H(p) = -\sum_{j=1}^{m} p_j \log_2 p_j \to \max,$$

$$\sum_{j=1}^{m} p_j = 1,$$

$$\prod_{j=1}^{m} r^{p_i} = const\ .$$

Последнее условие гарантирует постоянство среднегеометрического показателя:

$$r_{ji} = \sum_{i=1}^{n} r_{ji}^{p_i} .$$

Условный экстремум определяется методом неопределенного множителя Лагранжа [8], на основе максимума выписанной выше целевой функции и принципа потенциального распределения вероятностей (предпочтения выбора с большей вероятностью величины вклада в итоговое значение оценочного потенциала).

Веса признаков в оценочном потенциале различны. Оценки априорного распределения p_j связаны с отношением порядка предпочтения. Для линейных отношений порядка, упорядоченности, эти оценки образуют арифметическую прогрессию, которая убывает:

$$p_j = \frac{2(m - j + 1)}{m(m + 1)}.$$

На основе оценок можно ввести априорную вероятность, учитывая веса характеристик, участвующих при формировании оценочного потенциала развития ПО. Принципом потенциального распределения и формулой Байеса можно получить апостериорные условные вероятности:

$$p_j = \frac{\sum_{i=1}^{n} r_{ji} p_j}{\sum_{j=1}^{m} \sum_{i=1}^{n} r_{ji} p_j}.$$

Введение априорной вероятности вносит также логическую ясность в модули и комплекс ПО.

После ввода априорной вероятности, нахождения апостериорной условной вероятности получаем по теореме Байеса вероятностные оценки проявления *j*-го признака *i*-го варианта на оценочный потенциал ПО.

Вероятность *p(a)* ситуации определяется как

$$p_{ji}(a) = \frac{p_{ji}(r) p_j}{\sum_{i=1}^{n} \sum_{j=1}^{m} p_{ji}(r) p_j}.$$

Проверяем ошибки первого рода – неприятия гипотезы H_0, вероятность такой ошибки определяет уровень значимости, меру риска разработчика. Он выше, если меньше мера упорядоченности (структурированности) ПО на рассматриваем этапе. Это вероятность присутствия старых «дыр» ПО в новой версии. На примере развития MS Windows, видно значение и роль, отводимые компанией Microsoft «закрытию дыр» в новых версиях ОС. Но она не исправляет ошибки в той же версии.

Ошибка второго рода H_1 - принятия гипотезы, когда она неверна, ошибка риска заказчика (потребителя) ПО. Она характеризует меру преемственности версий ПО.

Дихотомический анализ системного анализа используется при декомпозиции проектов, помогает определить шкалу анализа для исследования содержательности программ, других информационных конструкций.

Информационно измерима алгоритмическая (программная) конструкция, которая, кроме качественной оценки, имеет и количественные, позволяющие сравнивать ее с другими конструкциями. Каждое сообщение (каждый результат тестирования, например)

информирует разработчика, уменьшает его информационную неопределенность по мере отладки.

Применяются практические методы выявления тренда временного ряда: чем меньше будет разброс остатков (невязок) относительно общего разброса значений, тем будет и лучшим прогноз.

Сложность проектирования ПО: системно-информационный подход

Проектирование ПО на этапе моделирования потока воздействующих событий опирается на эмуляцию внешней среды. Без этого создать ПО сложно. Как и без этапа кодирования.

Современные языки программирования – модульные, языковые средства определяют (описывают) модульные связи, в том числе, на объектном уровне.

Для сложного ПО используются оценки качества вариантов, методы измерения показателей качества ПО, его параметров. В комплексах программ взаимодействуют множество модулей, функций, объектов, переменных. Используются иерархические структуры, объектно-ориентированные транзакции и отношения. Со своим локальными мерами эффективности (качества). Возникает проблема надёжности и устойчивости функционирования ПО. ЖЦ ПО – длительный промежуток проектирования и разработки, с относительно небольшим промежутком времени эксплуатации ПО.

Для проектирования и оптимизации ПО системный анализ предоставляет средства (методы, принципы) достижения системных целей, оптимизации их показателей, анализа и учета неопределённостей, многокритериальности показателей и критериев качества.

Показатели качества – это некоторые метрики (неотрицательные числовые функции, обладающие свойствами меры расстояния, например, симметричности и «неравенства треугольника»), например, интервальная шкала, порядковая шкала, номинальная (категорированная) шкала. Структура мера приведена на рис. 1.

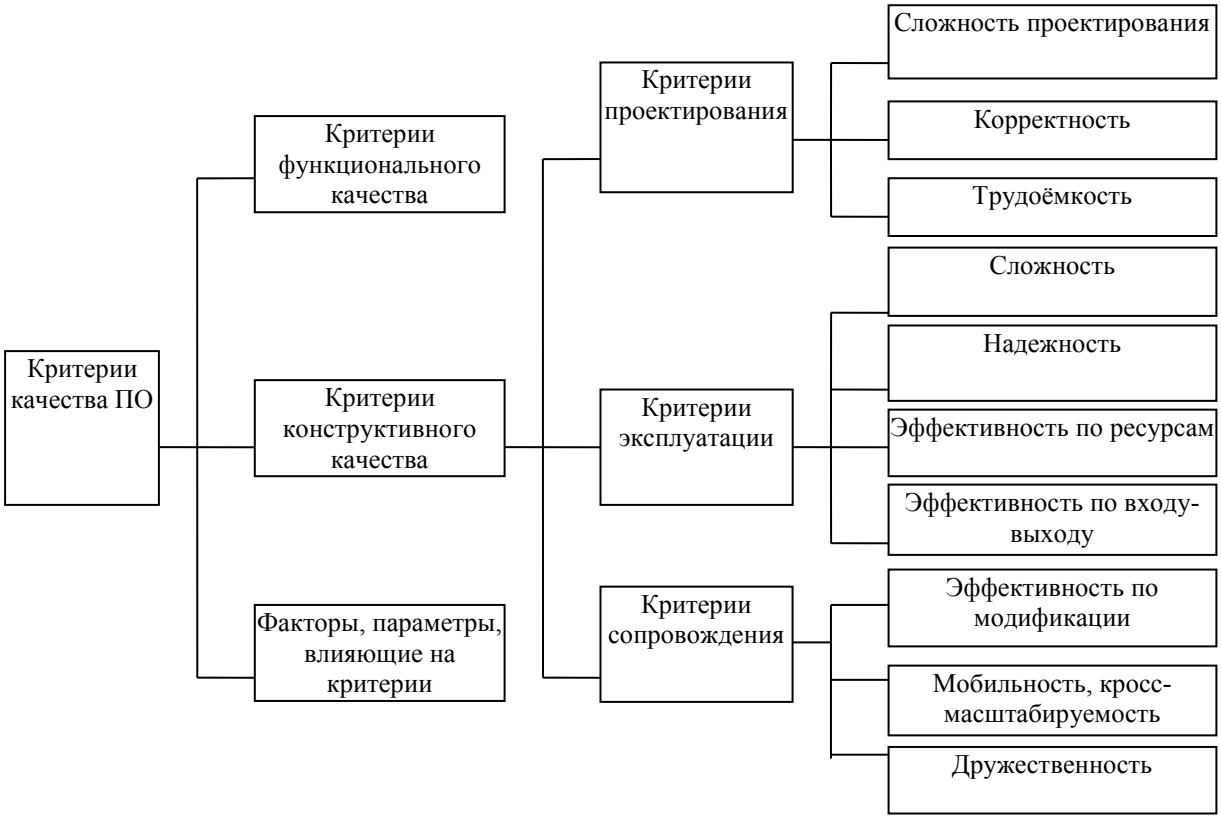

Рис. 1. Структура мер качества ПО

Сложность программ – показатель, характеризующий трудоёмкость разработки программы. Различают основные формы сложности – структурную (сложность, определяемая числом взаимодействующих компонентов программы и связей между ними или сложность программы в актуальном, рабочем состоянии) и статическую (сложность программы в неактуальном, нерабочем состоянии, когда она не получила управление).

Основные виды (типы) сложности ПО приведены на рис. 2.

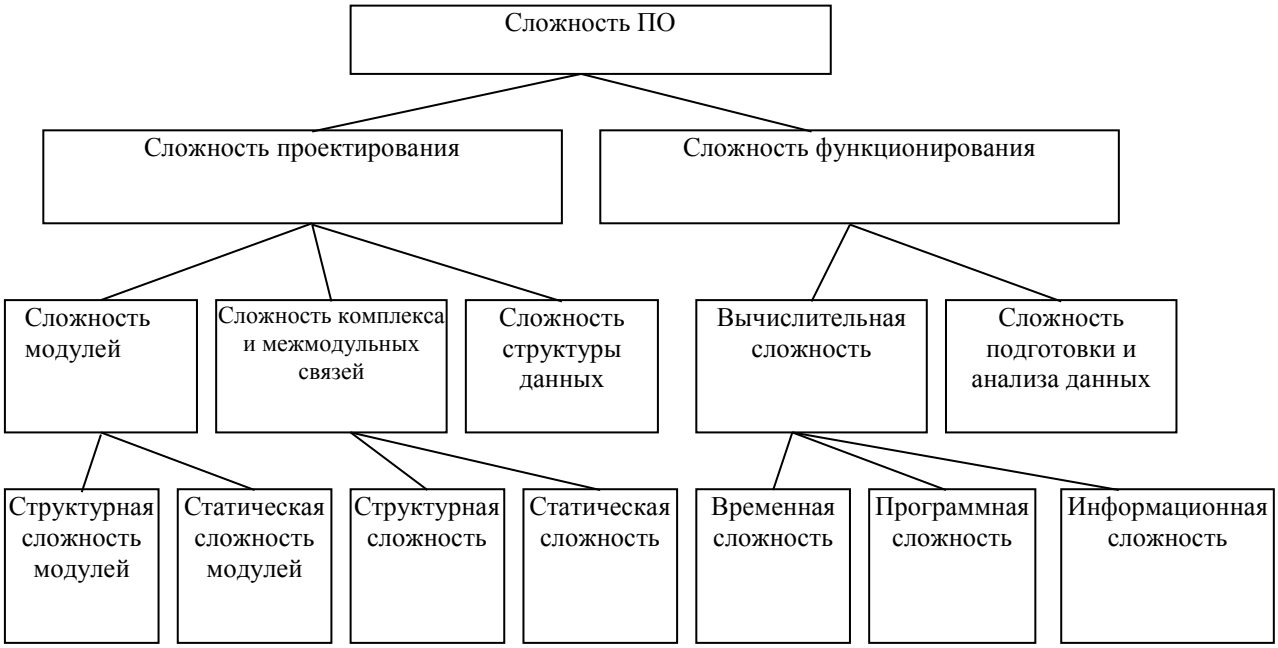

Рис. 2. Основные сложности проектирования (функционирования) ПО

На сложность проектирования ПО влияют факторы: количество команд и модулей, обрабатываемых переменных, объём памяти, трудоёмкость и длительность разработки, численность программистского коллектива и др.

Сложность проектирования ПО анализируют по сложности модулей, всего комплекса (его структуры), данных (их структур).

Динамическую сложность – по сложности вычислительной, подготовки и анализа результатов и др.

Исследования сложности вычислений развиваются по двум направлениям: интуитивно-прагматическому (или эвристическому) и формально-логическому (или аксиоматическому).

Развитие первого направления определяется практическими нуждами оценки вычислительных ресурсов для решения задач. Второе направление базируется на аксиоматическом определении сложности и исследует некоторые общие закономерности изменения сложности от объёма используемых данных, внимание акцентируется на сложности алгоритмической.

Структурная сложность определяется количеством взаимодействий, связей и сложностью модулей по числу μ ветвей, необходимых для тестирования (относительно числа условий ξ, которое необходимо задать в тестах для испытания всех ветвей ПО):

$$\xi = \sum_{i=1}^{\mu} \xi i \ ,$$

где ξ_i - количество условий (логических), определяющих *i*-ую ветвь дерева модуля (комплекса) ПО.

Сложность модуля $\aleph$ определяется количеством ветвей вычислительной схемы программы и операндов l_i ветви *i*, с весами, равными количеству различных значений v_i каждой величины:

$$\aleph = \sum_{i=1}^{\mu} l_i v_i \ .$$

Структурная сложность модуля рассчитывается количеством маршрутов μ в программе и сложностями каждого из них.

Частый критерий – минимальное множество ветвей программы (комплекса) для всех последовательностей передач управления (минимальное покрытие графа всей программы по управлению).

Можно ввести понятие остаточной сложности – максимальное число путей тестирования, необходимых для окончания тестирования по условию покрытия решений или покрытие логических переходов, т.е. по условию тестированности всех возможных ветвей дерева (маршрутов) программы. Покрытие решений требует получения в процессе тестирования каждого ожидаемого результата решений и любого оператора кода, по крайней мере, раз.

Определение статической сложности – на возможности их представления только из операторов и операндов. В качестве исходных для расчётов данных используется количество типов η_i операторов (*i=1*) и операндов (*i=2*). Сумма $\eta = \eta_1 + \eta_2$ - полный словарь анализируемой программы. Частота использования операторов *j*-го типа составляет f_{1j}, операндов – f_{2j}. Частоты их использования в программе пропорциональны, согласно формуле Шеннона, количеству их типов (логарифму от них):

$$N_1 = \sum_{j=1}^{\eta_1} f_{1j} = \eta_1 \log_2 \eta_1 ,$$

$$N_2 = \sum_{j=1}^{\eta_2} f_{2j} = \eta_2 \log_2 \eta_2 .$$

В результате длина модуля (комплекса) оцениваема по формуле:

$$N_c = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 .$$

Расчёты [9] показывают, что оценка N_c по составной программе в целом и по составляющим модулям отличаются незначительно (примерно на 10%), что объясняется уменьшением дисперсии исходных оценок.

Объём кода программы зависит как от алгоритма, так и от языка, на котором алгоритм представлен, он описывается выражением:

$$V = N_c \log_2 \eta \ .$$

Для сравнения объёмов программы на разных языках (при различном качестве программирования) вводят потенциальный объём программы *V*:*

$$V^* = \eta^* \log_2 \eta^* .$$

При снижении качества программирования упрощается по содержанию сама программа, расширяется ее объём. Качество программирования измеряется мерой расширения текста программы (L≤1) относительно потенциально возможного объёма. Уровень качества программирования:

$$\boldsymbol{L} = \frac{\boldsymbol{V}^*}{\boldsymbol{V}} \ .$$

Уровень программы тем выше, чем компактнее представляются операнды, чем ближе их общее количество N_2 к минимальному необходимому объёму словаря η_2. Учитывая, что L пропорционально η_1^* / η_1 $(L \sim \eta_1^* / \eta_1)$, а также η_2 / Nc $(L \sim \eta_2 / Nc)$, то комбинируя их и полагая, что коэффициент пропорциональности равен 1, можно представить оценку качества программирования в виде:

$$\hat{L} = \frac{\eta_1^*}{\eta_1} * \frac{\eta_2}{N_2} = \frac{2}{\eta_1} * \frac{\eta_2}{N_2}$$

Имеет место закон сохранения:

$$LV^* = const$$

или гипотеза о наличии своеобразного закона сохранения, по которому при любом фиксированном алгоритме произведение уровня качества L на потенциальный объём текста программы V^* остаётся постоянным и характеризует уровень языка программирования. Константу предложено представлять как уровень языка программирования:

$$\lambda = LV^* = L^2 V \ .$$

Оценки λ для реальных языков программирования показывают бесплодность мелких частных улучшений языков и нецелесообразность создания их многочисленных версий, как было раньше.

Сложность и глубина внутренних связей определяют прочность модуля. Наиболее важной и трудно формализуемой является задача оценки функциональной прочности модуля.

Функциональная прочность, как правило, приводит к логической и процедурной прочности модулей. Она же в наибольшей степени влияет на формирование границ модулей. Для повышения наглядности анализа сложности структуры комплекса ПО с учётом прочности связей, введены понятия прямых предков и прямых наследников рассматриваемого модуля.

Правила взаимодействия модулей:

исполнение программы начинается с корневого модуля, и он единственный;

управление может передаваться только модулю-предку (прямому);

только прямой предок может обращаться к прямому наследнику;

когда модуль завершает исполнение, управление передаётся обратно вызывающему модулю;

каждый модуль имеет только одну точку входа (вызова) и одну точку выхода из модуля без возврата.

Иные методы оценивания сложности управления ПО

Кроме метрик программ, есть метрики их сложности и управляемости. С их помощью, обычно исследуют плотность потоков управляющих внутренних переходов, взаимосвязи этих переходов.

Пусть программа, как это и принято, представима орграфом $G=(V,E)$, где V – вершины (функции, операторы и даже, если необходимо такое укрупненное рассмотрение, модули, процедуры), E - дуги, соответствующие логическим переходам (вызовам).

Рассмотрим основные метрики.

Метрика Мак-Кейба. Базируется на $Z(G)$ – цикломатической сложности G, характеризующей сложность тестирования:

$Z(G)=e–v+2p$,

где e - число дуг, v - число вершин, p - число компонентов связности, количество дуг, добавляемых для преобразования орграфа G в граф сильно связный (две вершины всегда взаимодостижимы).

Для графов модулей (программ), у которых нет недостижимых (или «висячих») точек входа-выхода, сильно связный граф можно получить замыканием вершины конца на вершину начала (входа в нее). Иначе, $Z(G)$ показывает мощность проходов для покрытия всех контуров сильно связного графа (число тестов тестового покрытия, по всем ветвям).

Метрика Хансена (модификация метрики Мак-Кейба). Определяется мерой сложности в виде пары (цикломатическая сложность, мощность операторов), это чувствительная к структурированности мера.

Метрика Чена. Оценивается мера количеством пересечений границ областей графа программы. Если орграф имеет всего одну компоненту связности, метрика Чена тождественно метрике Мак-Кейба.

Метрика Майерса. Г.Майерсом предложена интервальная мера сложности вида: $[Z(G), Z(G)+h]$. Для простого (без операции) предиката полагают $h=0$, для n-местных предикатов $h=n–1$.

Метрика Вудварда-Хенел-Хидлей (мера подсчета точек пересечения). Подходит для неструктурного (Ассемблер, например) программирования. В G каждому оператору сопоставляется вершина, при передаче управления (от a к b) номера этих операторов, соответственно находят как $min(a,b)$ и $max(a,b)$. Точку пересечения дуг определяет предикат:

$(min(a,b)<min(p,q)<max(a,b))$ *and* $(max(p,q)>max(a,b))$ *or* $(min(a,b)<max(p,q)<max(a,b))$ *and* $(min(p,q)<min(a,b))$.

Точка пересечения существует, если управление вышло за дугу (a,b). Число таких точек – характеристика неструктурированности модуля, программы.

Метрика Джилба. Логическая сложность программы определяется ее насыщенностью конструкциями типа IF-THEN-ELSE.

Метрики Харрисона, Мейджела. Учитывается вложенность и длина программы. Вершинам назначается начальные сложности согласно оператору, соответствующему вершине. Например, согласно метрикам Холстеда. Выделяется для предикатной вершины подграф, порождаемый концами исходящих дуг и достижимыми из них вершинами,

вершинами из предикатной вершины в сферу ее влияния. Сложность вершины предикатной – сумма сложностей начальных (приведенных) вершин и первичной ее сложности.

Метрика функциональная, SCOPE – простая сумма приведенных сложностей по вершинам орграфа.

Метрика Пивоварского. Это модификация известной из теории графов цикломатической меры сложности. Отслеживают различия последовательных и вложенных конструкций, структурированных и неструктурированных модулей. Мера растает от перехода к вложенным или к неструктурированным программам.

Метрика Чепина. Оценивается информационная устойчивость («прочность») модуля в зависимости от переменных списков ввода-вывода.

Другие метрики сложности [16-17]: потока данных, граничных значений, актуализации глобальных переменных, спена (локализации обращений изнутри к данным каждой секции), информационной «прочности» модулей (метод Чепина), стилистики (понятности программы), Мартина, Чидамбера и Кемерера, Берлингера, Мак-Клура, Овиедо, Кафура, Шнейдевинда, тестирующая М-Мера и др.

Корректность и моделирование распространения ошибок в программе

Корректность программы, комплекса определяется множеством факторов: соответствием исходных данных используемым в программе структурам данных, корректностью логики и др. Синтаксические, семантические конструкции – соответствуют ли правилам? Насколько формализуемы правила?

Корректность важна и функциональная – обработки данных и генерации результатов, и структурная (конструктивная) – соответствие структуры программы правилам структурного, объектно-ориентированного программирования.

Корректность данных также вносит вклад, как в функциональную, так и в конструктивную корректность. Конструктивная их корректность определяется структурированием. Функциональная же корректность – конкретизацией в процессе исполнения, текущими значениями абстрактных данных. Корректность комплексов – преимущественно функциональная.

Конструктивная корректность – в рамках правил и межмодульных и комплексных связей. Функциональная корректность плохо формализуема: мешает большое количество значений, распределений ошибок – искажений объекта, процесса.

Важная особенность идентификации ошибок в программах –отсутствие полностью правильной версии программы (эталона). Установить (локализовать) ошибки часто невозможно. Поэтому и есть длительный часто этап отладки.

Если длительность отладки τ величины $P_k^{(i)}(\tau)$ - вероятности наличия в программе первичной ошибки (не являющейся следствием исправления другой ошибки), которая при выполнении вносит в результирующую *j*-ую дополнительную ошибку Δk_j, то значение вторичной ошибки в *j*-ой переменной представимо формулой:

$$\tau_j(\tau) = \sum_{k=1}^{m} p_k^{(1)}(\tau)\Delta k_j .$$

Здесь *m* - полное количество типов первичных ошибок, не выявленных в программе.

Рассмотрим модель (алгоритм моделирования) распространения ошибок в программах. Предполагаем, что для изменения среднего времени наработки на обнаруженную ошибку от T_1 до T_2 необходимы затраты:

$$\Delta C_k = Q_k \Delta r + \mu_k \Delta n,$$

где Δr - длительность выполнения программы до (для) обнаружения числа Δn ошибок; Q_k - коэффициент привлечения ресурсов во время проверки и отладки программ; μ_k – удельный коэффициент привлечения ресурсов проектирования (в пересчёте на число обнаруженных ошибок).

При отладке накапливаются группы ошибок, образующие очередь на корректировку (исправление). Можно построить простую модель распространения ошибок, если установить связь интенсивности обнаружения имеющихся еще ошибок при отладке (потоком $\frac{dn}{d\tau}$), интенсивности проявления (величина λ) и количества первичных ошибок n.

Пусть начальный уровень ($\tau=0$) выявил N_0 первичных ошибок. В результате отладки за τ времени все еще остается n_0 первичных ошибок, а устраненных n ($n_0+n=N$).

Для определения характеристик по проявлениям ошибок важна длительность эксплуатации (продолжающей отладки). При естественном условии коррелированности значений n_0 и $n'(\tau)$ имеем:

$$\frac{dn}{d\tau} = k(N_0 - n),$$

где k – коэффициент прироста обнаруживаемых ошибок (зависит от усилий, вычислительных мощностей и др.).

Итак, интенсивность обнаружения ошибок связывается с числом устраненных первичных ошибок уравнением:

$$\frac{dn}{d\tau} + kn = kN_0.$$

Предположим, что в начале отладочного процесса (τ=0) нет обнаруженных ошибок: $n(0)=0$. Решим неоднородное уравнение. Для этого решаем сперва однородное уравнение:

$$\frac{dn}{d\tau} + kn = 0.$$

Разделим, как обычно, переменные:

$$\frac{dn}{n} = -kd\tau.$$

Проинтегрируем части уравнения, получим:

$$n = Ce^{-k\tau}.$$

Варьируем постоянную $C=C(\tau)$:

$$n=C(\tau)e^{-k\tau},\ n'(\tau)=C'(\tau)e^{-k\tau}-kC(\tau)e^{-k\tau}.$$

Подставим в уравнение и найдем:

$C'(\tau)=kN_0e^{-k\tau}$.

Интегрируем выражение:

$C(\tau)=kN_0\int e^{k\tau}d\tau=kN_0(1/k)e^{k\tau}+C_1=N_0e^{k\tau}+C_1$.

Подставим $C(\tau)$ в решение:

$$n(\tau)=N_0e^{k\tau}e^{-k\tau}+C_1e^{-k\tau}=N_0+C_1e^{-k\tau}.$$

По начальному условию получаем: $n(0)=N_0+C_1=0$ или $C_1=-N_0$ и, окончательно,

$$n(\tau)=N_0(1-e^{-k\tau}).$$

Количество первичных ошибок, оставшихся в программе будет равно:

$n_0=N_0-n=e^{-k\tau}$,

т.е. будет пропорционально усилиям (интенсивности) обнаружения $n'(\tau)$.

Между появлениями ошибок будет время:

$$T=\frac{1}{n'(\tau)}=\frac{1}{kN_0}e^{k\tau}.$$

Таким образом, для начала тестирования с наработкой T_0 имеем:

$$T(\tau)=T_0e^{\frac{\tau}{N_0T_0}}.$$

Если моменты обнаружения всех ошибок t_i будут известны, то обнаруживаются (устраняются) достоверно первичные ошибки.

Методом максимального правдоподобия [11] несложно получить уравнение первичных ошибок N_0:

$$\sum_{i=1}^{n}\frac{1}{N_0-(i-1)}=n\sum_{i=1}^{n}\frac{t_i}{(N_0\sum_{i=1}^{n}t_i-\sum_{i=1}^{n}(i-1)t_i)}.$$

Коэффициента k можно идентифицировать из этого соотношения:

$$k=\frac{n}{N_0\sum_{i=0}^{n}t_i-\sum_{i=1}^{n}(i-1)t_i}.$$

Можно рассчитать количество первичных ошибок, оставшихся в программе и среднюю величину наработки T до момента идентификации очередной ошибки.

Параметр k вычисляем в виде:

$$k=\max_{1\le i\le T}\left\{\frac{n(t_i)}{N_0\sum_{i=0}^{n(t_0)}t_i-\sum_{i=1}^{n(t_i)}(i-1)t_i}\right\},\ i=1,2,...,T$$

или

$$k=\frac{1}{T}\sum_{i=1}^{T}\frac{n(t_i)}{N_0\sum_{i=0}^{n(t_i)}t_i-\sum_{i=1}^{n(t_i)}(i-1)t_i}.$$

В процессе отладки для удаления Δn ошибок необходимо осуществить тестирование по времени от T_1 до T_2. Из последнего выражения, при T_2 и T_1, соответственно, получим:

$$T_1 = \frac{1}{kN_0} e^{k\tau_1}$$

$$T_2 = \frac{1}{kN_0} e^{k\tau_2}$$

или

$$\Delta n = n_2 - n_1 = e^{-k\tau_1} - e^{-k\tau_2} = N_0 T_0 (\frac{1}{T_1} - \frac{1}{T_2}).$$

В течение отладки (за t времени) число ошибок ε_c(t) на одну команду машинного языка вырастет, удельная величина ошибок в программе на команду после t времени отладки, равно будет величине:

$\varepsilon_r(t) = \frac{E_t}{I_t} - \varepsilon_c(t)$,

где I_t - количество машинных команд (постоянно).

Предполагаем, что частота z(t) встречаемости ошибок пропорциональна их количеству после t времени отладки, z(t)=C$\varepsilon_r(r)$. Тогда надежность (вероятность безотказной работы до времени t), равна

$$R(t,r) = \exp\{-C[\frac{E_T}{I_T} - \varepsilon_c(r)]t\}$$

Если при отладке испытали программу на *k* тестах в интервалах (0,r_1), (0,r_2),…, (0,r_k), где $r_1 < r_2 < \cdots < r_k$, то при $k \geq 2$ для оценок максимального правдоподобия C и E_T находим остаточные дисперсии параметров C и E_T.

Пусть теперь n_{max} - максимальный уровень ошибок в программе, *n(t)* - их количество, оставшееся к моменту времени *t.* Тогда можно записать динамическую картину, описываемую соотношением (при обеспечении условия $\lambda(c - t_0) \neq -1$):

$$n(t) = n_0 \frac{1 + \lambda(c - t)}{1 + \lambda(c - t_0)}.$$

Имитационное моделирование и анализ результатов моделирования

Приведенная выше модель обнаружения ошибок позволяет реализовать имитационные расчеты. С этой целью разработаны две демонстрационные программы моделирования. Программа ForExamples1 позволяет проводить расчеты коэффициентов отладки. Текст программы – ниже.

```
program ForExamples1;
uses crt;
var
 i,p,T,m0:integer;
 n:array[0..100] of real;
 k,h,f,z:real;
 s:array[0..100]of real;
 c:array[0..100]of real;
begin
 clrscr;
```

```
writeln(' m0-количество первичных ошибок в момент времени t=0');
write(' m0='); readln(m0);
write(' Введите коэффициент отладки k='); readln(k);
writeln(' n[t]-массив количества ошибок в момент времени t ');
write(' Введите n[0]='); readln(n[0]);
writeln(' T-размер массива n[t] (T<=100)'); write(' T='); readln(T);
p:=T;
for t:=0 to p-1 do
begin
n[t+1]:=n[t]+k*(m0-n[t]);
if (n[t+1]<=0)
then n[t+1]:=0
end;
z:=n[t+1];
f:=0;
if (k<>0) and (m0<>0) and (z>=0)
then
begin
for t:=0 to trunc(z) do
s[t]:=exp(k*t)/(k*m0);
f:=1;
end;
clrscr;
writeln('t n[t]'); writeln;
for t:=0 to p do
writeln(t,' ',n[t]);
readln; clrscr;
if (f=0)
then
writeln(' Массив s[t] вычислить не удастся.')
else
begin
writeln(' s[t]'); writeln;
for i:=0 to trunc(z) do
writeln(' s[',i,']=',s[i]);
end;
readln;
clrscr;
if (z<=0)
then
writeln(' Новый коэффициент отладки вычислить не удаётся.')
else
begin
writeln(' Всего ошибок ',trunc(z));
writeln(' t[i]-момент обнаружения i-той ошибки.');
```

```
writeln(' Введите t[i] для вычисления нового коэффициента отладки');
h:=0;
f:=0;
for i:=1 to trunc(z) do
begin
write('t[',i,']='); readln(c[i]);
h:=h+c[i];
f:=f+(i-1)*c[i]
end;
clrscr;
if (m0*h-f=0)
then writeln(' Новый коэффициент отладки не вычисляем')
else begin
k:=n[p]/(m0*h-f);
writeln(' Новый вычисленный коэффициент отладки');
writeln('k=',k);
end;
end;
readln;
end.
```

Вторая программа позволяет идентифицировать наработку между появлениями ошибок, число ошибок через некоторое время и число устраненных ошибок к этому моменту. Текст программы – ниже.

```
Program ForExamples2;
uses crt;
var
 N0,i,n:integer;
 h,tx,k,k1,n01,
 n1,T0,Ttx,c,nt:real;
 t:array[0..500]of real;
 function Y(k1:real):real;
 var s,s1:real;
 begin
 for i:=0 to n do
 s:=s+t[i];
 s:=s*N0;
 for i:=1 to n do
 s1:=s1+(i-1)*t[i];
 s:=s-s1;
 Y:=N0*(1-exp(-k1*tx))/s;
 end;
 function L(r:real):real;
 var
 c1:real;
 begin
```

```
Randomize;
c1:=t[1]+Random(trunc(Ttx-t[0]));
L:=abs((n1*t[0]-n01*r-n1+n01)/(c1*(n1-n01)));
end;
begin
ClrScr;
WriteLn('Обработка ошибок при тестировании'); WriteLn;
Write('Введите число первичных ошибок в начале'); ReadLn(N0);
Write('Введите время тестирования (мин)'); ReadLn(tx);
Write('Введите шаг по времени'); ReadLn(h);
t[0]:=0;
i:=0;
k:=0.1;
while(t[i]<=tx)do
begin
t[i+1]:=h+t[i];
i:=i+1;
end;
n:=trunc(N0*(1-exp(-tx*k)));
n01:=exp(-k*tx);
T0:=exp(k*tx)/(k*tx);
Ttx:=T0*exp(tx/(N0*T0));
k1:=Y(k);
while(abs(k1-k)>=0.01)do
begin
k:=k1;
k1:=Y(k);
end;
n1:=N0*(1-exp(-k*t[1]));
c:=(n1*t[0]-n01*t[1])/(L(t[1])*n1-L(t[1])*n01)-(1/L(t[1]));
nt:=n01*(1+L(c-tx))/(1+L(c-t[0]));
WriteLn; WriteLn(' Результаты: '); WriteLn;
WriteLn(' Наработка между появлениями ошибок');
WriteLn('Число ошибок через ',tx:2:1,' мин. = ',nt:1:0);
WriteLn('Число устраненных ошибок',N0-nt:1:0); ReadLn;
end.
```

Результаты тестовых расчетов – ниже.

Тест 1. Количество первичных ошибок m_0=36, коэффициент отладки k=0.004, количество ошибок в начале n[0]=100, время прогноза T=10 дней. Результаты прогноза от 0 до 10 дней: 100, 100, 99, 99, 98, 98, 98, 98, 97, 97, 97, 97.

Тест 2. Количество первичных ошибок m_0=19, коэффициент отладки k=0.01, количество ошибок в начале n[0]=4, время прогноза T=5. Количество ошибок меняется со временем: 4, 4, 4, 4, 3, 3. На шестые сутки останется всего три ошибки. Наработка между ошибками, соответственно: 5.26, 5.31, 5.37, 5.42, 5.48 суток. Идентифицированное новое значение коэффициента отладки k=0.0197.

Итак, убеждаемся:

- «простота теста – простота результата»;
- «простота модели – погрешность результата и понимание процесса»;
- «тестирование на гипотетических тестах – малопродуктивное».

Тем не менее, проведенные расчеты позволяют выделить основные требования к показателям качества программ:

1. критерий должен оценивать меру соответствия целевой функции программы (системы) на каждом этапе разработки;
2. критерий должен оценивать меру влияния на качество различных внешних факторов и внутренних параметров;
3. критерии (модели) должны быть простыми, слабо зависящими от неконтролируемых факторов.

Большое количество критериев и факторов, а также отсутствие упорядоченных и ранжирования затрудняют конструктивное применение таких модулей. Анализ качества проводится с выделением и упорядочиванием факторов и показателей на всем ЖЦ ПО.

Показатели качества отслеживаются разработчиками (согласно представлениями самих разработчиков). Программы, имеющие длительную эксплуатацию, необходимы при регулярной обработке информации, управлении функционированием сложных вычислительных систем.

Учитывая, что хотя примерные размеры таких программ от 1000 до 1 миллиона команд, все потенциально обладают свойством модифицируемости в процессе сопровождения и использования.

При написании программы часто требуется определить их сложность. Уровень сложной программы определяется по таким параметрам, как удобочитаемость, количество комментариев, количество используемых логических и управляющих операторов. Также целесообразно определить количество и уровень вложенности условных и циклических операторов.

В результате проведенного анализа выявлено: для тестируемого файла, например, написанного на Паскале, целесообразно произвести подсчет по метрикам Холстеда и выдать:

1. количество типов операторов:
2. количество типов операндов:
3. частота использования операторов каждого типа:
4. частота использования операндов каждого типа:
5. максимальный уровень сложности операторов цикла:
6. число комментариев:

В работе такой программы необходимо использовать следующие основные части:

1. считывание файла;
2. удаление комментариев и их подсчет;
3. подсчет количества используемых типов операторов и количества операторов каждого типа.

Эффективность и качество систем в большинстве случаев желательно сопоставить с затратами в тех же единицах измерении, широко применяя экономические критерии. Показатели качества идентифицируются вводом метрик, шкал, функционалов качества.

Например, в зависимости от применяемых метрик и шкал границы показателей могут быть следующими:

1. средняя длина модуля – 4-40 строк;
2. средняя длина имени (идентификатора) – 1-12 символов (не сильно обращает внимание на часто разрешаемую, например, как в Паскале длину до 255 знаков);
3. процент комментариев – 5-15%;
4. процент отступов – 10-80%;
5. процент пустых строк – 5-50%;
6. среднее число символов в строке – 1-50;
7. среднее число пробелов в строке – 1-20;
8. число подключаемых файлов – 1-8 и др.

Важной смежной проблемой для рассматриваемой нами задачи является оценка трудозатрат на разработку ПО. Качественная оценка трудозатрат на разработку ПО – центральная проблема в управлении проектами на рынке ПО. Оффшорные компании и их клиенты испытывают все возрастающую потребность в надежных методах и средствах оценки трудоемкости ПО, главная цель обеих сторон – получение качественного законченного продукта в рамках установленных графика и бюджета. Существуют различные методики (COCOMO, Putnam's SLIM, методы экспертно-эвристические, аналогий, нисходящий, восходящий и др.).

Модель оценки трудоемкости ПО считается хорошей, если обеспечивает результаты с точностью 20-70% случаев на определенном классе ПО.

Выявлены основные шаги процесса оценивания:

1. формулировка целей;
2. планирование требуемых ресурсов;
3. уточнение требований к ПО;
4. более полная проработка деталей;
5. использование нескольких независимых методов оценки;
6. сравнение и взаимное уточнение оценок;
7. плановый учет и контроль.

Автоматизация расчета метрик и оценки трудоемкости проектирования

При проектировании ПО (вообще, любой ИС) оценивание сложности процесса реализации зависит от сложности структур данных и БД, на которые опирается проект. Это необходимо для автоматизации систем (АС).

CASE-средства позволяют ускорить этот процесс, а метрики Холстеда – количественно оценивать сложность системы. Хотя мы рассматривали выше элементы ПО (модули, программы), метрики Холстеда позволяют измерять и реальные БД, представленные, например, SQL-скриптами объектов БД [13]. Учитываются параметры: количество уникальных и всех операторов и операндов скрипта (согласно определениям данных SQL – операции, процедур, функций пользователей, имена полей, представлений, таблиц и др.).

Анализ SQL-скрипта дает оценки словаря скрипта, его длины, объема, уровня, интеллектуального содержания (интеллектуальных затрат на создание БД).

Основная трудность – декомпозиция операторов SQL для выделения операторов и операндов. Можно использовать рекурсивные определения и методы (например, рекурсивный спуск). Для автоматизации оценивания и проектирования можно использовать,

например, инструментарий MySQL (Community Edition 5.6.15) и PHP-5. Сама АС может включать различные модули (анализа SQL-скрипта, расчёта метрик, архивирования, визуализации, генерации отчётов и др.). Основной же модуль – модуль реализации функционала проекта. Основные алгоритмы – оценивания сложности SQL-скрипта и всего проекта, производительности скрипта и всего проекта, а также программиста, сравнения с граничными и оптимальными значениями.

Например, в [13] выводится график (рис. 3) вовлеченности в процесс разработки программистов, который формируется на основе значений метрик по скриптам, загружаемым разработчиком по текущим заданиям.

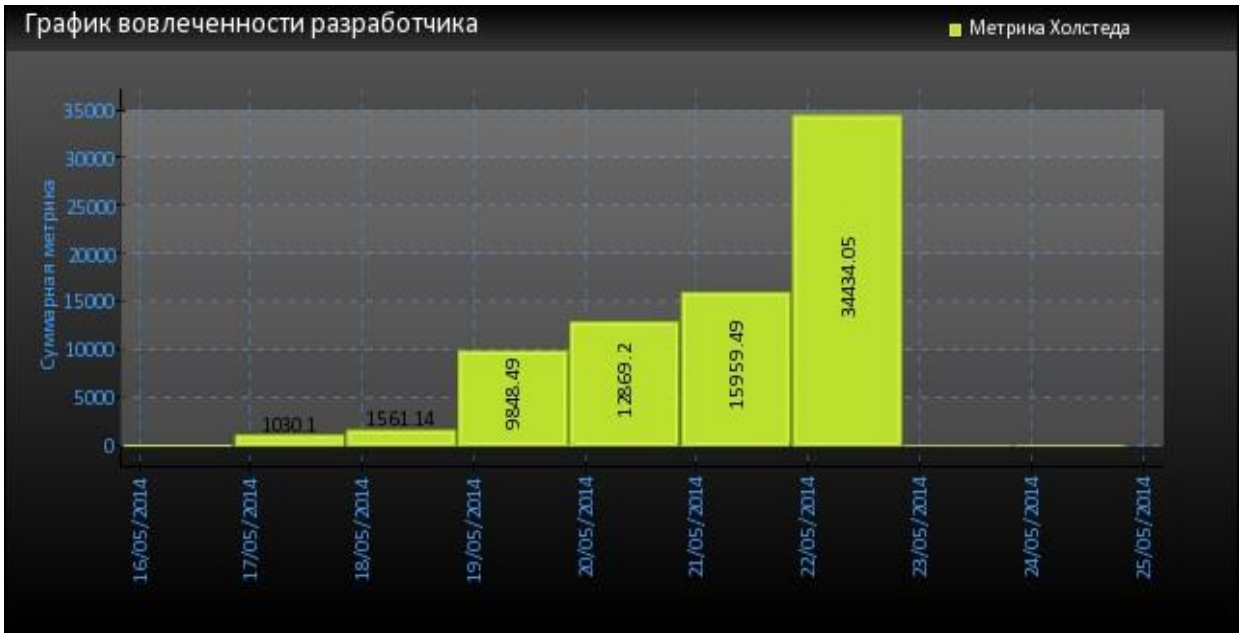

Рис. 3. График вовлеченности разработчика (взято из [13])

Рассмотрим второй пример, построенный нами для демонстрации учебных возможностей всего описанного выше, особенно, автоматизации.

Delphi – сложная среда, обеспечивающая высокоэффективную визуализируемую работу программиста. Реализуется многооконностью и визуальным дружественным интерфейсом. Окна перемещаемы по экрану, перекрываемы. Можно быстро отыскать окно, изменить свойства создаваемой программы, каждое окно предназначено для решения определенных задач. Запустив Delphi можно увидеть окна: главное (Project), формы (Form1), Инспектора Объектов (Object Inspector) и кода программы (Unit1 или, точнее, Unit1.PAS). Функциональность не зависит от расположения, размеров окон.

В окне кода создается и редактируется текст программы. В Delphi используем язык Object Pascal – расширенная, модифицированная версия Turbo (или Borland) Pascal. Окна взаимно связываются. Структура программ (файл с расширением .DPR), модулей (файлы .PAS), описывающих программные единицы Паскаль, автоматически создают Delphi и имеет структуру:

```
Program Project1;
Uses
 Forms,
 Unit1 in `Unit1.pas`;
{$R.RES}
```

```
begin
 Application.Initialize;
 Application.CreateForm(TfmExample, fmExample);
 Application.Run;
end.
```

Фрагмент

```
 Uses
 Forms
 Unit1 in `Unit1.pas`;
```

сообщает, что с проектом используются модули Forms (стандартный) и Unit1.

Следующий фрагмент – вариант модуля:

```
Unit Unit1;
Interface // Интерфейсные объявления
Implementation // Реализация
Begin // Инициация
End. // Завершение
```

Разделение на секции программы обеспечивает дружественную процедуру обмена алгоритмами (частями программы). Теперь, собственно, о примере: реализуется программа (программная система) обучения методам сортировки [14-15], визуализирующая процесс метода сортировки, влияние на его ход выбранной структуры данных.

Это хороший пример, когда цель достигаема различными алгоритмами, акцентирования и визуализации их достоинств и недостатков в конкретной ситуации (для конкретного теста). Кроме того, визуализация методов сортировки отлично подходит к визуализации понятия устойчивости и эффективности алгоритма.

Анализируются, оцениваются показателей эффективности метода – количество сравнений и перестановок, они определяют эффективность метода в конкретном случае. Реализация в среде Delphi – релевантна проблеме (привлекает сравнительный анализ, синтез, отлично демонстрирует зависимость метода, алгоритма от выбранной структуры данных и «внутренний» процесс в памяти компьютера). Описание объектов – в рабочем окне, достаточно пользователю выбрать метод сортировки, программа пошагово отсортирует ряд, прокомментирует и визуализирует процесс. На рис. 4 приведен интуитивно понятный сценарий работы с программой.

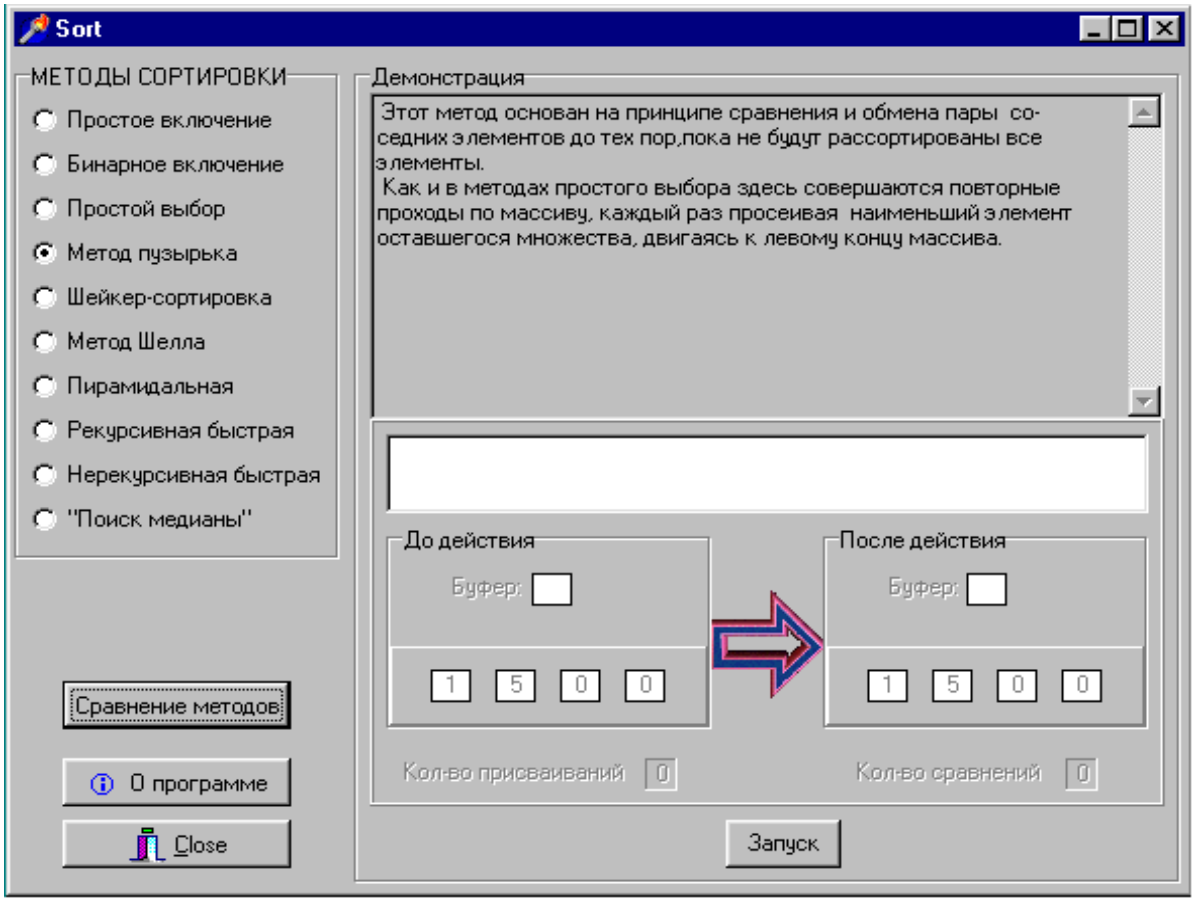

Рис.4. Сценарий визуализации «пузырьковой» сортировки по программе

Метрика автоматизации должны быть измеримы, объективны, основаны на легкодоступных данных, осмыслены, просты (аналитически, вычислительны), способны совершенствовать процесс автоматизации.

Важно учитывать:

1. процент автоматизируемых кейсов, ведь не все возможно (и даже нужно) автоматизировать;
2. продвигаемость автоматизации (относительно тестов, тестового покрытия) и количества автоматизированных кейсов (относительно количества, которые могут автоматизироваться в текущий момент);
3. плотность дефектов (багов), можно оценить количеством открытых багов (относительно объема, точнее длины приложения);
4. покрытие, количество автоматизированных кейсов (относительно количества требований);
5. эффективность устранения дефектов (относительно багов при тестировании и найденных пользователями при эксплуатации);
6. возможность численной оценки функции программы;
7. возможность оценки затрат для достижения критериального (нормативного) уровня качества, степени влияния на показатели качества внешних факторов;
8. малость остаточной дисперсии.

Заключение

Исследователь, эксперт, тестировщик ПО должен снабжать тестируемое ПО входными данными. Назовем их функциональным разрезом, определим, главным образом, через распределения вероятностей значений входных переменных.

На практике используется сотни метрик программ, но наиболее существенные (для автоматизации, оценивания, сравнения, обучения) из них группируются следующим образом:

1. меры сложности программ (структурной, информационной);
2. меры надежности программ, прогноза риск-ситуаций;
3. меры производительности, его повышения снижением ошибок проектирования и разработки;
4. меры уровня языковых сред;
5. меры сложности восприятия (понимания) текстов, психологического характера, но важные для модификации и сопровождения программ;
6. меры производительности труда разработчиков, программистов, тестировщиков ПО.

Метрики Холстеда позволяют оценивать качественные и количественные характеристики программ (статическое состояние).

Выбор использованных в работе моделей обусловлен был рядом причин и гипотез (например, простоты), учетом существенных (управляющих) факторов, достаточно оригинальных. Модели могут быть полезны для прогнозирования надежности ПО.

Важно прогнозировать отладочное состояние программ, комплексов, в многообразии и сложности их поведения. Без специальных (часто искусственных) гипотез, необходимых при оценивании качества. Расчеты на основе построенной несложной имитационной модели показывают возможность нахождения вариантов развития программных комплексов, изменяя значения параметров, как количество ошибок, вычислительная мощность, тестовое покрытие, распределение ошибок, логическая корректность и др.

Обнаружение ошибок и число устраненных первичных ошибок можно исследовать простым дифференциальным уравнением. Эта модель позволяет оценивать количественные характеристики программ (динамическое состояние). Решая это уравнение при условии Коши, можно получить динамику распределения синтаксических, семантических и логических ошибок в программе (программном комплексе).

На основе этой динамики возможен расчёт (прогноз) времени отладки, начиная от времени начала тестирования с наработкой T_0.

Если известны моменты обнаружения ошибок, то достоверно устраняются все первичные ошибки. При наличии ошибок в программе она реализует уже не заданную функцию, а иную (работа программы прекращается, выполнение не заканчивается). Все аналогичные случаи ведут к рабочим отказам.

При разработке нового комплекса ПО, совершенствуя эволюционно систему, необходимо идентифицировать параметры (оптимальные) характеристик будущего комплекса на желаемый момент времени. В качестве меры эволюции комплекса выбираем информационную энтропию. Очевидно, чем дальше прогноз развития, тем меньше надежность получаемых характеристик. Имеется приемлемое оптимальное время прогнозирования.

Все перечисленное является полезным инструментарием не только в программистских фирмах, но и в ВУЗе, когда необходимо оценивать (самооценивать) программы студентов по качественным и количественным характеристикам.

Литература

[1]. Липаев В.В. // КАЧЕСТВО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ // М.: Финансы и статистика, 1983

[2]. Казиев В.М. // ВВЕДЕНИЕ В АНАЛИЗ, СИНТЕЗ И МОДЕЛИРОВАНИЕ СИСТЕМ. // М., Бином. Лаборатория знаний – Интуит, 2007, 244 с.

[3]. Седов Е.А. // ЭВОЛЮЦИЯ И ИНФОРМАЦИЯ. // М.:Наука, 1976.

[4]. Gutlin Libal // INFORMATION THEORYAND THE LIVING SYSTEM. // N. Y.: Columbia University Press, 1972.

[5]. Прохоров Ю.В., Понамаренко Л.С. // ЛЕКЦИИ ПО ТЕОРИИ ВЕРОЯТНОСТЕЙ. // М.: Изд-во МГУ, 2012, - 256 с.

[6]. Костин В.Н., Шевченко С. Н. // МЕТОДИКА ФОРМИРОВАНИЯ ТРЕБОВАНИЙ К СИСТЕМЕ ФИЗИЧЕСКОЙ ЗАЩИТЫ НА ОСНОВЕ КОНЦЕПТУАЛЬНОЙ ИМИТАЦИОННОЙ МОДЕЛИ // Инфокоммуникационные технологии. 2013. № 2. С.91-98.

[7]. Мушков А. Ю., Тихомиров В. А., Тихомиров А. В. // МОДЕЛИ И МЕТОДЫ СТРАТЕГИЧЕСКОГО УПРАВЛЕНИЯ СЛОЖНЫМИ ЭКОНОМИЧЕСКИМИ И ТЕХНОЛОГИЧЕСКИМИ СИСТЕМАМИ. // Тверь: ВУ ПВО, 2003, 244 с.

[8]. УСЛОВНЫЙ ЭКСТРЕМУМ. МЕТОД МНОЖИТЕЛЕЙ ЛАГРАНЖА. http://math1.ru/education/funct_sev_var/lagranj.html (доступ 01.06.2016).

[9]. Холстед, Морис Х. // НАЧАЛА НАУКИ О ПРОГРАММАХ // М.: Финансы и статистика, 1981. – 128 с.

[10]. Цветков В.Я. // ИНФОРМАЦИОННАЯ НЕОПРЕДЕЛЕННОСТЬ И ИНФОРМАЦИОННАЯ ОПРЕДЕЛЕННОСТЬ В НАУКАХ ОБ ИНФОРМАЦИИ. // «Информационные технологии», №1, 2015, стр.3-6.

[11]. Джонсон Н., Лион Ф. // СТАТИСТИКА И ПЛАНИРОВАНИЕ ЭКСПЕРИМЕНТА В ТЕХНИКЕ И НАУКЕ. // Пер. с англ. - М.: Мир, 1981.

[12]. Подловченко Р.И. // ОБ ОДНОЙ МЕТОДИКЕ РАСПОЗНАВАНИЯ ЭКВИВАЛЕНТНОСТИ В АЛГЕБРАИЧЕСКИХ МОДЕЛЯХ ПРОГРАММ // Программирование. 2011. №6. С. 33-43.

[13]. Азаров А.В., Рыбанов А.А. // АВТОМАТИЗИРОВАННАЯ СИСТЕМА РАСЧЕТА МЕТРИЧЕСКИХ ХАРАКТЕРИСТИК ФИЗИЧЕСКОЙ СХЕМЫ БАЗЫ ДАННЫХ С ЦЕЛЬЮ ОЦЕНКИ ТРУДОЕМКОСТИ ПРОЦЕССА ПРОЕКТИРОВАНИЯ // Современная техника и технологии. 2014. №5, http://technology.snauka.ru/2014/05/3812 (дата обращения: 02.06.2016).

[14]. Кнут, Дональд. // ИСКУССТВО ПРОГРАММИРОВАНИЯ // том 3. Сортировка и поиск, 2-е изд. –М.: Вильямс, 2007. – с. 824.

[15]. Кормен Томас Х., Лейзер сон Чарльз И., Ривест Рональд Л., Штайн Клиффорд. // Алгоритмы: построение и анализ. // 2-ое изд. –М.: Вильямс, 2006. – с. 1296.

[16]. Антошина И.В., Домрачев В.Г., Ретинская И.В. // ОСНОВНЫЕ ТЕНДЕНЦИИ ОЦЕНИВАНИЯ КАЧЕСТВА ПРОГРАММНЫХ СРЕДСТВ // Качество, инновации, образование. 2004. № 1. С. 70–75.

[17]. Калинина Л.Ю. // ОЦЕНКА КАЧЕСТВА ПРОГРАММНЫХ ПРОДУКТОВ // Качество, инновации, образование. 2006. № 4. С. 52–55.

[18]. Фишборн П.С. // ТЕОРИЯ ПОЛЕЗНОСТИ ДЛЯ ПРИНЯТИЯ РЕШЕНИЙ // М.: Наука, 1978.

[19]. Саджид Манзур. // ИСПОЛНИТЕЛЬНЫЕ ПРИЗНАКИ ПРИЛОЖЕНИЙ СЕТИ И ИДЕНТИФИКАЦИЯ УЗКИХ МЕСТ, http://ru.agileload.com/agileload//blog/2012/11/27/web-applications-performance-symptoms-and-bottlenecks-identification (доступ 01.06.2016)

[20]. Воронцов К.В. // ЛЕКЦИИ ПО АЛГОРИТМАМ КЛАСТЕРИЗАЦИИ И МНОГОМЕРНОГО ШКАЛИРОВАНИЯ. // М. 2007. – 182 с.

[21]. Липаев В.В. // О ПРОБЛЕМАХ ОЦЕНИВАНИЯ КАЧЕСТВА ПРОГРАММНЫХ СРЕДСТВ // Качество, инновации, образование. 2002. № 1. С. 93–97.

[22]. Баркалов С.А., Бурков В.Н., Пинигин А.Ю., Хорохордина Н.В. // ПОСТРОЕНИЕ ГИБКИХ СИСТЕМ КОМПЛЕКСНОГО ОЦЕНИВАНИЯ В ЗАДАЧАХ ОПТИМИЗАЦИИ РЕГИОНАЛЬНЫХ ПРОГРАММ // Вестник Воронежского государственного технического университета. 2009. Т. 5. № 3. С. 70-73.

[23]. Бастраков С.И., Донченко Р.В., Мееров И.Б., Половинкин А.Н. // ОСОБЕННОСТИ ОПТИМИЗАЦИИ ВЫЧИСЛЕНИЙ В ПРИКЛАДНЫХ ПРОГРАММАХ НА ЯЗЫКЕ С НА ПРИМЕРЕ ОЦЕНИВАНИЯ ОПЦИОНОВ ЕВРОПЕЙСКОГО ТИПА // Научно-технический вестник информационных технологий, механики и оптики. 2010. № 5 (69). С. 95-100.

[24]. Бахмутский А.Е., Савинов В.М. // ИНДИКАТОРЫ ЭФФЕКТОВ ПРОГРАММ РАЗВИТИЯ ОБРАЗОВАНИЯ И МЕТОДЫ ИХ ОЦЕНИВАНИЯ // Письма в Эмиссия.Оффлайн: электронный научный журнал. 2005. № 2. С. 994.

[25]. Бистерфельд О.А. // МЕТОДИКА АВТОМАТИЗИРОВАННОГО ОЦЕНИВАНИЯ ЭФФЕКТИВНОСТИ ПРОГРАММ И ПРОГРАММНЫХ КОМПЛЕКСОВ // Информационные системы и технологии. 2011. №

2 (64). С. 12-18.
[26]. Быстров О.Ф. // ЭКСПРЕСС - ОЦЕНКА СТЕПЕНИ ТУРИСТСКОГО ИНВЕСТИЦИОННОГО ПРОЕКТА (ПРОГРАММЫ) С ИСПОЛЬЗОВАНИЕМ ПРОЦЕДУРЫ РЕЙТИНГОВОГО ОЦЕНИВАНИЯ // Научный вестник МГИИТ. 2010. Т. 6. № 4. С. 40-43.
[27]. Диасамидзе С.В. // МЕТОД И МЕТОДИКА ВЫЯВЛЕНИЯ НЕДЕКЛАРИРОВАННЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ СТРУКТУРИРОВАННЫХ МЕТРИК СЛОЖНОСТИ // Известия Петербургского университета путей сообщения. 2012. № 3 (32). С. 29-37.
[28]. Зацман И.М., Шубников С.К. // МЕТОДЫ ВЕРИФИЦИРУЕМОГО ОЦЕНИВАНИЯ ЦЕЛЕВЫХ ПРОГРАММ НАУЧНЫХ ИССЛЕДОВАНИЙ // Системы и средства информатики. 2010. Т. 20. № 2. С. 23-48.
[29]. Карпов Ю.Г., Трифонов П.В. // СЛОЖНОСТЬ АЛГОРИТМОВ И ПРОГРАММ // Компьютерные инструменты в образовании. 2007. № 6. С. 3-10.
[30]. Карпухин И.А., Короткова Н.Н. // КРИТЕРИЙ ОЦЕНКИ СЛОЖНОСТИ ПРОГРАММ // Международный студенческий научный вестник. 2016. № 3-1. С. 114-115.
[31]. Кибзун А.И., Иноземцев А.О. // ОЦЕНИВАНИЕ УРОВНЕЙ СЛОЖНОСТИ ТЕСТОВ НА ОСНОВЕ МЕТОДА МАКСИМАЛЬНОГО ПРАВДОПОДОБИЯ // Автоматика и телемеханика. 2014. № 4. С. 20-37
[32]. Клемент Л., Мостерд М.Ч. // О СЛОЖНОСТИ ОЦЕНИВАНИЯ НАУЧНОЙ ДЕЯТЕЛЬНОСТИ // Проблемы управления в социальных системах. 2014. Т. 7. № 10. С. 22-39.
[33]. Кожуховская Е.И., Морозова Л.Э. // О ПЕРСПЕКТИВАХ ОЦЕНИВАНИЯ ГОСУДАРСТВЕННЫХ ПРОГРАММ И ПРОЕКТОВ // Научные труды SWorld. 2009. Т. 29. № 1. С. 44-49.
[34]. Колесников А.К. //
[35]. Колесников А.К. // ВАРИАНТ МОДЕЛИРОВАНИЯ ВЗАИМОСВЯЗЕЙ ПОКАЗАТЕЛЕЙ СЛОЖНОСТИ И ТРУДНОСТИ ОБРАЗОВАТЕЛЬНЫХ ПРОГРАММ НА ОСНОВЕ РЕГРЕССИОННЫХ УРАВНЕНИЙ // Наука и современность. 2011. № 12-2. С. 44-49.
[36]. Колесников А.К. // КОНЦЕПЦИЯ ИЗМЕРЕНИЯ СЛОЖНОСТИ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ (НА ПРИМЕРЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ) // Сибирский педагогический журнал. 2011. № 9. С. 29-41.
[37]. Комаринский С.М. // ОСНОВНЫЕ РЕЗУЛЬТАТЫ ЭКСПЕРТНОГО ОЦЕНИВАНИЯ КАЧЕСТВА ТЕСТОВЫХ ПОДСИСТЕМ ПРОГРАММ ДИСТАНЦИОННОГО ОБУЧЕНИЯ // Известия Южного федерального университета. Педагогические науки. 2014. № 4. С. 117-124.
[38]. Кузнецов Б.П. // СТРУКТУРА И СЛОЖНОСТЬ МОДУЛЕЙ ЦИКЛИЧЕСКИХ ПРОГРАММ // Автоматика и телемеханика. 1999. № 2. С. 151-165.
[39]. Курочкин А.В. // ОЦЕНИВАНИЕ ПРОГРАММ И ПОЛИТИК В КОНТЕКСТЕ АДМИНИСТРАТИВНОЙ РЕФОРМЫ В РФ // Политическая экспертиза: ПОЛИТЭКС. 2011. Т. 7. № 1. С. 117-125.
[40]. Лукьянов Г.В. // ПРОБЛЕМА МЕТОДОЛОГИИ ОЦЕНИВАНИЯ РЕЗУЛЬТАТИВНОСТИ НАУЧНЫХ ПРОГРАММ И ПРОЕКТОВ // Системы и средства информатики. 2010. Т. 20. № 2. С. 75-87.
[41]. МЕТОДИКА ОЦЕНКА СЛОЖНОСТИ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ (НА ПРИМЕРЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ) //
[42]. Морозова Л.Э. // МЕТОДОЛОГИИ ОЦЕНИВАНИЯ ГОСУДАРСТВЕННЫХ ПРОГРАММ // Вестник Российского нового университета. Серия: Человек и общество. 2008. № 2. С. 168-173.
[43]. Наводнов В.Г., Мотова Г.Н., Аносова Н.А. // ТЕХНОЛОГИЯ ОЦЕНИВАНИЯ КАЧЕСТВА КОРОТКИХ ОБРАЗОВАТЕЛЬНЫХ ПРОГРАММ НА ОСНОВЕ ИСПОЛЬЗОВАНИЯ ИНСТИТУЦИОНАЛЬНОГО МЕХАНИЗМА // Труды Поволжского государственного технологического университета. Серия: Социально-экономическая. 2013. № 1. С. 47-54.
[44]. Овчинников М.Н. // ОБ ОЦЕНИВАНИИ ДЕЯТЕЛЬНОСТИ УНИВЕРСИТЕТОВ И ПОКАЗАТЕЛЯХ ЭФФЕКТИВНОСТИ ПРОГРАММ РАЗВИТИЯ // Университетское управление: практика и анализ. 2012. № 1. С. 25-30.
[45]. Попов С.В. // СЛОЖНОСТЬ ВЫЧИСЛЕНИЯ ПРОГРАММ // Международный научно-исследовательский журнал. 2014. № 1-2 (20). С. 36-38.
[46]. Профильная школа. 2011. № 6. С. 42-49.
[47]. Романова Е.Л. // ПРИМЕНЕНИЕ ИДЕЙ БЛЕЙКА И МОУТОН ДЛЯ РАЗРАБОТКИ ПРОГРАММЫ ОЦЕНИВАНИЯ СТИЛЯ РАБОТЫ ТЬЮТОРА // Вестник Международного института менеджмента ЛИНК. 2007. № 18. С. 67-71.
[48]. Соловьев В.Н. // КОМПЛЕКС ПРОГРАММ ОПТИМАЛЬНОГО ГАРАНТИРУЮЩЕГО ОЦЕНИВАНИЯ // Системный анализ в науке и образовании. 2010. № 4. С. 37-43.
[49]. Таранцева К.Р., Моисеев В.Б., Пятирублевый Л.Г. // РАСПРЕДЕЛЕНИЕ ЗАДАНИЙ ПО УРОВНЮ СЛОЖНОСТИ И УЧЕБНЫМ ЦЕЛЯМ ПРИ РАЗРАБОТКЕ КОМПЕТЕНТНОСТНОГО ПОДХОДА К ОЦЕНИВАНИЮ ЗНАНИЙ // XXI век: итоги прошлого и проблемы настоящего плюс. 2015. Т. 3. № 6 (28). С. 161-165.
[50]. Утёмов В.В., Горев П.М. // ОЦЕНИВАНИЕ МЕТАПРЕДМЕТНЫХ РЕЗУЛЬТАТОВ ОСВОЕНИЯ ПРОГРАММ ОБЩЕГО ОБРАЗОВАНИЯ НА ОСНОВЕ КОЭФФИЦИЕНТА ИНТЕЛЛЕКТУАЛЬНОСТИ // Научно-методический электронный журнал Концепт. 2014. № 4. С. 1-5

[51]. Чуновкина А.Г., Спаев В.А., Степанов А.В., Звягин Н.Д. // ОЦЕНИВАНИЕ НЕОПРЕДЕЛЕННОСТИ ИЗМЕРЕНИЙ ПРИ ИСПОЛЬЗОВАНИИ ПРОГРАММ ОБРАБОТКИ ДАННЫХ // Измерительная техника. 2008. № 7. С. 3-8.
[52]. Щербань А.Б. // ПОДХОД К ОЦЕНИВАНИЮ СТРУКТУРНОЙ СЛОЖНОСТИ СИСТЕМЫ // Современные информационные технологии. 2016. № 23 (23). С. 62-64.

References

[1]. Lipaev V.V. // QUALITY OF THE SOFTWARE / / M .: Finances and Statistics, 1983
[2]. Kaziev V.M. // INTRODUCTION TO ANALYSIS, SYNTHESIS AND MODELING OF SYSTEMS. / M., Bean. Laboratory of Knowledge - Intuit, 2007, 244 p.
[3]. Sedov E.A. // EVOLUTION AND INFORMATION. / / M.: Science, 1976.
[4]. Gutlin Libal // INFORMATION THEORYAND THE LIVING SYSTEM. // N. Y .: Columbia University Press, 1972.
[5]. Prokhorov Yu.V., Ponamarenko LS // LECTURES ON THE THEORY OF PROBABILITY. / Moscow: Publishing House of Moscow State University, 2012, - 256 p.
[6]. Kostin VN, Shevchenko SN // METHODOLOGY FOR FORMING REQUIREMENTS TO THE PHYSICAL PROTECTION SYSTEM BASED ON THE CONCEPTUAL IMITATION MODEL // Infocommunication technologies. 2013. № 2. P.91-98.
[7]. Mushkov A. Yu., Tikhomirov VA, Tikhomirov AV // MODELS AND METHODS OF STRATEGIC MANAGEMENT OF COMPLEX ECONOMIC AND TECHNOLOGICAL SYSTEMS. // Tver: VU PVO, 2003, 244 p.
[8]. CONDITIONAL EXTREMUM. METHOD OF LAGRANGE MULTIPLAYERS. http://math1.ru/education/funct_sev_var/lagranj.html (access 01/06/2016).
[9]. Halstead, Maurice H. // BEGINNING SCIENCE ABOUT PROGRAMS // Moscow: Finances and Statistics, 1981. - 128 p.
[10]. Tsvetkov V.Ya. // INFORMATION UNCERTAINTY AND INFORMATION DEFINITION IN SCIENCES ABOUT INFORMATION. // "Information technologies", №1, 2015, pp.3-6.
[11]. Johnson N., Lyon F. // STATISTICS AND PLANNING OF THE EXPERIMENT IN TECHNIQUE AND SCIENCE. // Transl. with English. - Moscow: Mir, 1981.
[12]. Podlovchenko R.I. // ABOUT ONE METHODOLOGY OF RECOGNITION OF EQUIVALENCE IN ALGEBRAIC MODELS OF PROGRAMS // Programming. 2011. № 6. Pp. 33-43.
[13]. Azarov AV, Rybanov AA // AUTOMATED SYSTEM OF CALCULATION OF METRIC CHARACTERISTICS OF THE PHYSICAL DIAGRAM OF THE DATABASE WITH THE PURPOSE OF ASSESSING THE LABOR PROCESS OF THE DESIGN PROCESS // Modern engineering and technology. 2014. №5, http://technology.snauka.ru/2014/05/3812 (date of circulation: 02.06.2016).
[14]. Knut, Donald. // THE ART OF PROGRAMMING // Volume 3. Sorting and searching, 2nd ed. -M .: Williams, 2007. - p. 824.
[15]. Kormen Thomas H., Leiser dream Charles I., Rivest Ronald L., Stein Clifford. // Algorithms: construction and analysis. // 2nd ed. -M .: Williams, 2006. - p. 1296.
[16]. Antoshina IV, Domrachev VG, Retinskaya IV // MAIN TRENDS OF ESTIMATION OF QUALITY OF SOFTWARE / / Quality, innovation, education. 2004. № 1. C. 70-75.
[17]. Kalinina L.Yu. // EVALUATION OF QUALITY OF SOFTWARE PRODUCTS // Quality, innovation, education. 2006. № 4. P. 52-55.
[18]. Fishborn PS // THEORY OF USEFULITY FOR DECISION MAKING // Moscow: Nauka, 1978.
[19]. Sajid Manzur. // EXECUTIVE SIGNS OF NETWORK APPLICATIONS AND NODE IDENTIFICATION, http://www.agileload.com/agileload//blog/2012/11/27/web-applications-performance-symptoms-and-bottlenecks-identification (access 01/06/2016)
[20]. Vorontsov K.V. // LECTURES ON CLUSTERIZATION ALGORITHM AND MULTIDIMENSIONAL SCALING. // M. 2007. - 182 c.
[21]. Lipaev V.V. // ABOUT PROBLEMS OF ESTIMATION OF QUALITY OF SOFTWARE / / Quality, innovation, education. 2002. № 1. C. 93-97.
[22]. Barkalov SA, Burkov VN, Pinigin A.Yu., Horohordina N.V. // CONSTRUCTION OF FLEXIBLE SYSTEMS OF COMPLEX ESTIMATION IN PROBLEMS OF OPTIMIZATION OF REGIONAL PROGRAMS // Bulletin of Voronezh State Technical University. 2009. T. 5. № 3. P. 70-73.
[23]. Bastrakov SI, Donchenko RV, Meerov IB, Polovinkin A.N. // FEATURES OF OPTIMIZATION OF CALCULATIONS IN APPLIED PROGRAMS ON LANGUAGE WITH THE EXAMPLE OF ESTIMATION OF EUROPEAN TYPE OPTIONS // Scientific and Technical Herald of Information Technologies, Mechanics and Optics. 2010. № 5 (69). Pp. 95-100.
[24]. Bakhmutsky A.E., Savinov V.M. // INDICATORS OF EFFECTS OF DEVELOPMENT PROGRAMS OF EDUCATION AND THE METHODS OF THEIR ESTIMATION // Letters to the Issue. Offline: electronic scientific journal. 2005. № 2. P. 994.

[25]. Bisterfeld OA // METHODOLOGY OF AUTOMATED EFFECTIVENESS ESTIMATION OF PROGRAMS AND SOFTWARE COMPLEXES // Information Systems and Technologies. 2011. № 2 (64). Pp. 12-18.
[26]. Bystrov OF // EXPRESS - EVALUATION OF THE DEGREE OF TOURISTIC INVESTMENT PROJECT (PROGRAM) WITH THE USE OF THE RATING ESTIMATION PROCEDURE // Scientific Bulletin of the Moscow State Institute of International Relations. 2010. T. 6. № 4. P. 40-43.
[27]. Diasamidze S.V. // METHOD AND METHODOLOGY OF IDENTIFICATION OF UNDECLARATED POSSIBILITIES OF PROGRAMS WITH THE USE OF STRUCTURED METRICS OF COMPLEXITY // Proceedings of St. Petersburg University of Communications. 2012. No. 3 (32). Pp. 29-37.
[28]. Zatsman IM, Shubnikov SK // METHODS OF VERIFIED EVALUATION OF TARGETED SCIENTIFIC RESEARCH PROGRAMS // Systems and Means of Informatics. 2010. T. 20. № 2. P. 23-48.
[29]. Karpov Yu.G., Trifonov P.V. // COMPLEXITY OF ALGORITHMS AND PROGRAMS // Computer tools in education. 2007. № 6. P. 3-10.
[30]. Karpukhin IA, Korotkova NN // CRITERIA FOR EVALUATING THE COMPLEXITY OF PROGRAMS // International Student Scientific Bulletin. 2016. № 3-1. Pp. 114-115.
[31]. Kibzun AI, Inozemtsev A.O. // EVALUATION OF LEVELS OF DIFFICULTY OF TESTS BASED ON THE METHOD OF MAXIMAL JUSTIFICATION // Automation and telemechanics. 2014. № 4. P. 20-37
[32]. Clement L., Mostard M.C. // ABOUT THE COMPLEXITY OF ESTIMATION OF SCIENTIFIC ACTIVITY // Problems of management in social systems. 2014. T. 7. № 10. P. 22-39.
[33]. Kozhuhovskaya E.I., Morozova L.E. // ABOUT PERSPECTIVES OF ASSESSMENT OF STATE PROGRAMS AND PROJECTS // Scientific works of SWORLD. 2009. T. 29. № 1. P. 44-49.
[34]. Kolesnikov A.K. //
[35]. Kolesnikov A.K. // A MODE OF MODELING THE INTERDEPENDENCE OF INDICES OF COMPLEXITY AND DIFFICULTY OF EDUCATIONAL PROGRAMS ON THE BASIS OF REGGRESSIONAL EQUATIONS // Science and Modernity. 2011. № 12-2. Pp. 44-49.
[36]. Kolesnikov A.K. // THE CONCEPT OF MEASURING THE COMPLEXITY OF THE EDUCATIONAL PROGRAM (ON THE EXAMPLE OF HIGHER PROFESSIONAL EDUCATION) // Siberian Pedagogical Journal. 2011. № 9. With. 29-41.
[37]. Komarinsky SM // MAIN RESULTS OF EXPERT EVALUATION OF QUALITY OF TEST SUBSYSTRIES OF REMOTE TRAINING PROGRAMS // News of Southern Federal University. Pedagogical sciences. 2014. № 4. P. 117-124.
[38]. Kuznetsov B.P. // STRUCTURE AND DIFFICULTY OF MODULES OF CYCLIC PROGRAMS // Automation and telemechanics. 1999. № 2. P. 151-165.
[39]. Kurochkin A.V. // ESTIMATION OF PROGRAMS AND POLITICS IN THE CONTEXT OF ADMINISTRATIVE REFORM IN THE RUSSIAN FEDERATION // Political Expertise: POLITEX. 2011. T. 7. № 1. P. 117-125.
[40]. Lukyanov G.V. // THE PROBLEM OF THE METHODOLOGY OF ESTIMATION OF THE RESULTIVITY OF SCIENTIFIC PROGRAMS AND PROJECTS // Systems and Means of Informatics. 2010. T. 20. № 2. P. 75-87.
[41]. METHODOLOGY ASSESSMENT OF THE COMPLEXITY OF THE EDUCATIONAL PROGRAM (EXAMPLE OF HIGHER PROFESSIONAL EDUCATION) //
[42]. Morozova L.E. // METHODOLOGIES OF ESTIMATION OF STATE PROGRAMS // Bulletin of the Russian New University. Series: Man and Society. 2008. № 2. P. 168-173.
[43]. Navodnov VG, Motova GN, Anosova NA // TECHNOLOGY OF ESTIMATE QUALITY OF SHORT EDUCATIONAL PROGRAMS ON THE BASIS OF USE OF INSTITUTIONAL MECHANISM // Proceedings of the Volga State Technological University. Series: Socio-economic. 2013. No. 1. P. 47-54.
[44]. Ovchinnikov M.N. // ABOUT EVALUATION OF THE ACTIVITY OF UNIVERSITIES AND INDICATORS OF EFFECTIVENESS OF DEVELOPMENT PROGRAMS // University management: practice and analysis. 2012. № 1. P. 25-30.
[45]. Popov S.V. // COMPLEXITY OF CALCULATING PROGRAMS // International Scientific and Research Journal. 2014. No. 1-2 (20). Pp. 36-38.
[46]. Profile school. 2011. № 6. P. 42-49.
[47]. Romanova E.L. // APPLYING THE IDEAS OF BLAKE AND MOUTON FOR DEVELOPMENT OF THE PROGRAM OF ESTIMATION OF THE TYOTOR WORKING STYLE // Bulletin of the International Institute of Management LINK. 2007. № 18. P. 67-71.
[48]. Soloviev V.N. // COMPLEX OF OPTIMAL GUARANTEING ESTIMATION PROGRAMS // System analysis in science and education. 2010. № 4. P. 37-43.
[49]. Tarentseva KR, Moiseev VB, Pyatirublevy LG // DISTRIBUTION OF TASKS BY THE LEVEL OF COMPLEXITY AND EDUCATIONAL OBJECTIVES AT THE DEVELOPMENT OF THE COMPETENCY APPROACH TO THE ASSESSMENT OF KNOWLEDGE // XXI century: the results of the past and the problems of the present plus. 2015. Vol. 3. No. 6 (28). Pp. 161-165.
[50]. Utyomov VV, Gorev PM // ESTIMATION OF METAPREDMET RESULTS OF THE DEVELOPMENT OF PROGRAMS OF GENERAL EDUCATION BASED ON THE INTELLECTUALITY COEFFICIENT // Scientific and Methodical Electronic Journal Concept. 2014. No. 4. C. 1-5

[51]. Chunovkina AG, Spaev VA, Stepanov AV, Zvyagin ND // ESTIMATION OF UNCERTAINTY OF MEASUREMENTS WHEN USING DATA PROCESSING PROGRAMS // Measuring technique. 2008. № 7. P. 3-8.
[52]. Shcherban A.B. // APPROACH TO ESTIMATE THE STRUCTURAL COMPLEXITY OF THE SYSTEM // Modern Information Technologies. 2016. No. 23 (23). Pp. 62-64.