Интеллектуальная система оценивания и прогнозирования сложности и качества программного обеспечения

Введение. Ключевым фактором обеспечения эффективного применения программных продуктов является тщательное оценивание и достижение высоких значений показателей качества. Постоянный рост сложности функций, реализованных программами в информационных системах, непосредственно приводит к увеличению их объема и трудоемкости их создания. Согласно изменениям сложности программ возрастает количество найденных и оставленных в них дефектов и ошибок, что отражается на их качестве. Программа объемом в миллионы строк кода в принципе не может быть безошибочной. Проблема нахождения и устранения ошибок обостряется по мере увеличения сложности задач и программ, которые их решают, и угрожает катастрофами в областях, которые используют программное обеспечение (ПО) [1-3].

Проблема заключается в том, чтобы обеспечить нужное качество функционирования ПО с учетом того, что некоторое неизвестное количество ошибок и дефектов всегда остается в сложных комплексах программ, и должно быть блокировано или сокращено их негативное воздействие до допустимого уровня. В связи с этим стратегическая задача в жизненном цикле современного ПО - обеспечение качества программных продуктов [4].

Качество ПО определяется качеством методов и инструментальных средств, которые применялись для обеспечения всего их жизненного цикла. На практике важно оценивать качество программ не только в завершенном виде, но и в процессе их проектирования и разработки.

Сегодня ПО является определяющей составляющей многих систем, среди которых системы критического применения, встроенные и специализированные системы различного назначения. Для указанных систем наличие ошибок и низкое качество ПО грозят катастрофами, которые приводят к человеческим жертвам, экологическим катаклизмам, экономическим потерям. Развитие современных технологий разработки ПО требует динамичного развития средств оценки качества ПО, причем уже на этапе проектирования (с точки зрения экономической и временной целесообразности).

Современная индустрия ПО характеризуется высокой конкуренцией. Для успешной работы на этом рынке софтверная компания должна разрабатывать, внедрять и сопровождать ПО быстро, укладываясь в сроки, и с удовлетворительным качеством. Поэтому многие софтверные компании вкладывают средства в модернизацию технологий разработки ПО, помня о том, что такое вложение средств обязательно окупается.

На этапе проектирования важно заложить целый ряд требований по качеству: требования к структуре программы, требования к навигации по

ПО, требования к дизайну интерфейсов пользователя, требования к мультимедиа-компонентам ПО, требования к удобству применения (usability), технические требования.

На этапе проектирования формируется ответ на вопрос "Каким образом программное обеспечение будет реализовывать выдвинутые к нему требования?".

Информационные потоки этапа проектирования ПО [5]: требования к ПО, представленные информационной, функциональной и поведенческой моделями анализа. Информационная модель описывает информацию, котрую должно обрабатывать ПО по мнению заказчика. Функциональная модель определяет перечень функций обработки информации и перечень модулей ПО. Поведенческая модель фиксирует желаемую динамику системы (режимы ее работы). На выходе этапа проектирования - разработка данных, разработка архитектуры и процедурная разработка ПО.

Для получения оценки значений показателей качества по стандарту [6] используются следующие методы:

1)измерительный - базируется на использовании инструментальных, измерительных и специальных программных средств для получения информации о свойствах и характеристиках ПО (объем ПО, количество строк кода, количество операторов, число ветвей в программе, количество точек входа / выхода и др.);

2) регистрационный - основан на получении информации во время испытаний или функционирования ПО, когда регистрируются или подсчитываются определенные события (время и количество сбоев и отказов, время передачи управления от одного модуля к другому, время начала и завершения работы ПО);

3) органолептический - основан на использовании информации, полученной в результате анализа восприятий органов чувств, и применяется для определения таких показателей как удобство применения, эффективность;

4) расчетный - основан на использовании теоретических и эмпирических зависимостей (на ранних этапах разработки), статистических данных, собранных при проведении испытаний, эксплуатации и сопровождении ПО. Расчетными методами оцениваются показатели надежности, точности, устойчивости, время реакции, необходимые ресурсы и проч.;

5) экспертный - осуществляется группой экспертов. Их оценка базируется на опыте и интуиции, а не на непосредственных результатах расчетов или экспериментов. Этот метод реализуется путем просмотра программ, кодов, сопроводительных документов, описаний требований к ПО группой экспертов. Для этого устанавливаются контролируемые признаки, коррелированные с одним или несколькими показателями качества и включенные в карты опроса экспертов. Метод применяется при

оценке таких показателей, как наглядность, полнота и доступность программной документации, легкость усвоения, возможность анализа, документированность, структурированность ПО и др.

Из описания методов измерения показателей (метрик) качества понятно, что на этапе проектирования ПО невозможно измерить ни одной характеристики еще не разработанного ПО, невозможно регистрировать моменты процесса выполнения еще не существующего ПО и невозможно воспринять органами чувств информацию о не разработанном ПО. Так, на этапе проектирования является возможность определять качество ПО только с применением расчетных и экспертных методов.

Согласно [7], метрика определяется как мера степени владения свойством, которая имеет числовое значение. Вообще, метрика ПО - это мера, которая позволяет получить числовое значение некоторого свойства программного обеспечения или его спецификаций. Итак, если речь идет о метрики сложности программ, то это меры, которые дают числовое значение сложности ПО, а метрики качества программ дают числовое значение качества ПО.

Показатели оценки сложности и качества проекта очень важны для получения объективных оценок по проекту. Как правило, данные показатели не могут быть вычислены на ранних стадиях работы над проектом, поскольку требуют, как минимум, детального проектирования. Однако эти показатели важны для получения прогнозных оценок длительности и стоимости проекта, поскольку непосредственно определяют его трудоемкость.

Несмотря на многочисленные исследования программных метрик, в этой области остается *ряд нерешенных вопросов*:

1) технология измерения качества еще не достигла зрелости - только 1.5% софтверных организаций пытаются оценить качество процессов и готового продукта количественно, с помощью метрик, и только 0.5% софтверных организаций пытаются улучшить работу, руководствуясь количественными критериями качества для выпуска бездефектных продуктов;

2) отсутствуют единые стандарты на метрики - создано более тысячи метрик, каждый поставщик "измерительной" системы предлагает свои способы оценки качества и соответственно метрики;

3) существует проблема сложности интерпретации величин метрик - значения метрик, полученные с помощью "измерительных" систем, неинформативны или мало информативны для пользователя, заказчика, а часто и для программиста;

4) метрики рассчитываются только для готового ПО - все "измерительные" системы ориентированы на расчет метрик программного кода, но часто существует необходимость в расчете метрик уже на этапе проектирования - метрик сложности и качества с точными значениями

для проекта ПО и метрик сложности и качества с прогнозируемыми значениями для разрабатываемого по проекту программного обеспечения.

Именно из-за нерешенности этих вопросов пока невозможно создать бездефектное высококачественное ПО. Сложность обоснования выбора и интерпретации метрик в процедурах принятия производственных решений, а также игнорирование этапов жизненного цикла ПО не позволяют полноценно использовать метрики для повышения качества ПО и уменьшения его сложности.

Метрики сложности и качества с точными значениями на этапе проектирования ПО. В [8-10] был проведен анализ метрик сложности и качества программного обеспечения с точки зрения возможности их применения на этапе проектирования ПО с получением точного или прогнозируемого значения. В результате такого анализа был выделен ряд метрик, которые уже на этапе проектирования имеют точные значения:

1) *метрика Чепина* - анализирует характер использования переменных из списка ввода, т.е. обрабатываемую информацию. Множество переменных с учетом их значимости вычисляется как: $Q = P + 2M + 3C + 0{,}5T$, где P - переменные для расчетов и вывода; M - модифицированные или созданные в программе переменные; C - управляющие переменные, T - не используемые в программе ("паразитные") переменные. Метрика Чепина дает оценку информационной прочности модуля;

2) *метрики связности* (связность (cohesion) - внутренняя характеристика программного модуля) - зависят от типа модуля или проекта. На основе [5, 11, 12] опишем следующие правила определения уровня связности модуля и силы связности (СС):

- если модуль - единичная проблемно-ориентированная функция, то уровень связности - функциональный (части модуля вместе реализуют одну функцию) и СС = 10;

- если действия внутри модуля связаны данными, и порядок действий внутри модуля важен, то уровень связности - информационный (выходные данные одной части используются как входные данные в другой части модуля) и СС = 9;

- если действия внутри модуля связаны данными, и порядок действий внутри модуля не имеет никакого значения, то уровень связности - коммуникативный (части модуля связаны данными - работают с одной структурой данных) и СС = 7;

- если действия внутри модуля связаны потоком управления, и порядок действий внутри модуля важен, то уровень связности - процедурный (части модуля связаны порядком выполняемых им действий, которые реализуют некоторый сценарий поведения) и СС = 5;

- если действия внутри модуля связаны потоком управления, и порядок действий внутри модуля не имеет никакого значения, то уровень

связности - временной (части модуля не связаны, но необходимы в один и тот же момент работы системы) и СС = 3; недостатком такого типа связности является сильная взаимосвязь с другими модулями, из-за чего проект становится очень чувствительным к внесению изменений;

- если действия внутри модуля никоим образом не связаны, но относятся к одной категории, то уровень связности - логический (части модуля объединены по принципу функционального сходства) и СС = 1; недостатками логической связности является сложное сопряжение и высокая вероятность внесения ошибок при изменении сопряжения ради одной из функций;

- если действия внутри модуля никоим образом не связаны, а также не относятся к одной категории, то уровень связности - по совпадению (в модуле отсутствуют явно выраженные внутренние связи) и СС = 0.

Возможны более сложные случаи, когда с модулем ассоциируются несколько уровней связности. В таком случае, учитывая статистическую и экспертную информацию о типе и области применения разрабатываемого ПО, модулю присваивается самый сильный уровень связности или самый слабый уровень связности.

Чем выше связность модуля, тем лучше результат проектирования, т.е. чем "чернее" ящик проекта (защитная оболочка модуля), тем меньше "рычагов управления" на нем находится и тем он проще. Такие типы связности, как связность по совпадению, логическая связность и временная связность - результат неверного планирования архитектуры, а процедурный тип связности - результат небрежного планирования архитектуры приложения;

3) *метрики сцепления* (сцепление (coupling) - внешняя характеристика модуля, которую желательно и необходимо уменьшать) - мера взаимозависимости модулей по данным. Количественно сцепление измеряется степенью сцепления (СЦ). На основе [5, 11, 12] выделим следующие типы сцепления:

- сцепление по данным (СЦ = 1) - один модуль вызывает другой модуль, все входные и выходные параметры вызываемого модуля - простые элементы данных;

- сцепление по образцу (СЦ = 3) - в качестве входных и выходных параметров используются структуры данных;

- сцепление с управлением (СЦ = 4) - один модуль явно руководит функционированием другого модуля с помощью флажков или переключателей, посылая ему управляющие данные;

- сцепление по внешним ссылкам (СЦ = 5) - модули ссылаются на один и тот же глобальный элемент данных;

- сцепление по общей области (СЦ = 7) - модули разделяют одну и ту же глобальную структуру данных;

- сцепление по содержанию (СЦ = 9) - один модуль прямо ссылается на содержание другого модуля (не через его точку входа).

4) *метрика Джилба (составляющая метрики - абсолютная модульная сложность программы)* - на этапе проектирования можно подсчитать количество межмодульных связей N_{ZV}. Метрика Джилба вычисляется также и по другой методике - количество условных операторов (L_{IF}) и операторов цикла (L_{LOOP}) составляет абсолютную логическую сложность программы $CL = L_{IF} + L_{LOOP}$.

5) *метрика Мак-Клура* - метрика, направленная на оценку архитектуры системы; мера сложности, основанная на количестве возможных путей выполнения программы, количестве управляющих конструкций и переменных. Вычисляется в 3 этапа:

- для каждой управляющей переменной i вычисляется значение ее функции сложности $C(i) = (D(i) * J(i))/n$, где $D(i)$ - величина, которая измеряет сферу действия переменной (для локальной переменной $D(i) = 0$, а для глобальной переменной - $D(i) = 1$), $J(i)$ - мера сложности взаимодействия модулей по этой переменной (сколько раз модули взаимодействовали с использованием этой переменной), n - количество модулей;

- для всех модулей определяются функции сложности $M(P) = fp * X(P) + gp * Y(P)$, где fp, gp - соответственно количество модулей, непосредственно предшествующих и непосредственно следующих за модулем P, $X(P)$ - сложность обращения к модулю (сколько раз происходит обращение к модулю P), $Y(P)$ - сложность управления вызовом из модуля P других модулей (сколько раз модуль P вызывает другие модули);

- общая сложность MP программного обеспечения: $MP = \sum_{P} M(P)$.

Данная метрика ориентирована на хорошо структурированное программное обеспечение, для которого возможно построить схему разбиения на модули. В каждом модуле предусмотрена одна точка входа и одна точка выхода, модуль выполняет одну функцию, вызов модуля осуществляется в соответствии с иерархической системой управления. Метрика Мак-Клура позволяет выбрать схему разбиения с меньшей сложностью еще до написания программы;

6) *метрика Кафура* - метрика, основанная на учете потоков данных. Для расчета этой метрики вводятся понятия локального и глобального потоков данных. Если модуль А вызывает модуль В, то это прямой локальный поток из А в В; если модуль В вызывает модуль А и модуль А возвращает значение в модуль В - это косвенный локальный поток из А в В; если модуль С вызывает модули А и В и передает результат из модуля А в модуль В - это локальный поток из А в В.

Глобальный поток из модуля A в модуль B сквозь глобальную структуру данных D существует, если модуль A помещает информацию в структуру D, а модуль B использует информацию из структуры D. Тогда:

- информационная сложность процедуры вычисляется по формуле: $I = length \times (fan\_in \times fan\_out)^2$, где $length$ - сложность текста процедуры (измеряется по одной из метрик сложности), fan_in - количество локальных потоков внутрь процедуры плюс число структур данных, fan_out - количество локальных потоков из процедуры плюс число структур данных, которые обновляются процедурой;

- информационная сложность модуля относительно некоторой структуры данных вычисляется по формуле: $I = W \times R + W \times WrRd + WrRd \times R + WrRd \times (WrRd - 1)$, где W - количество процедур, которые обновляют структуру данных, R - количество процедур, которые читают информацию из структуры данных, $WrRd$ - количество процедур, которые читают и обновляют структуру данных;

7) *метрика обращения к глобальным переменным* - пара (модуль, глобальная переменная) - пара (p,r), где p - модуль, имеющий доступ к глобальной переменной r. В зависимости от наличия реального обращения к переменной r формируются 2 типа пар (p,r) - фактические и возможные. Характеристика Aup показывает, сколько раз модуль действительно получит доступ к глобальной переменной, а характеристика Pup - сколько раз он мог бы получить такой доступ. Тогда приближенную вероятность ссылки произвольного модуля на произвольную глобальную переменную можно вычислить как отношение: $Rup = \frac{Aup}{Pup}$. Чем выше такая вероятность, тем выше вероятность "несанкционированного" изменения глобальной переменной, что приводит к усложнению модификации программы;

8) *время модификации моделей* - метрика процесса разработки ПО;

9) *количество найденных ошибок при инспектировании моделей и прототипов подсистем, модулей, функций, требований* - указывает на проблемную подсистему, модуль, функцию, требование. Это метрика процесса разработки ПО.

Метрики сложности и качества с прогнозируемыми значениями на этапе проектирования ПО. В [8-10] выделены метрики этапа проектирования ПО с прогнозируемыми значениями:

1) *ожидаемая LOC-оценка (экспертная)* - по каждой функции эксперты предоставляют лучшее, худшее и вероятное значения, тогда ожидаемая LOC-оценка (LOC - количество строк исходного кода программы) вычисляется по формуле:

$$LOC_{ожi} = (LOC_{лучшi} + LOC_{худшi} + 4 \times LOC_{верi})/6 .$$

Показатель LOC зависит от языка программирования. Это оценочная и необязательная метрика. Она может привести к оптимизации количества строк кода, а не проекта в целом. Позволяет прогнозировать трудозатраты, время и стоимость разработки, а также необходимое количество программистов для выполнения проекта. Есть свободно распространяемые инструменты ожидаемой LOC-оценки [13]. LOC-оценка влияет на оценку величины изменений объема кода во времени;

2) *метрики Холстеда:*

- мера длины модуля $N = n_1 \log_2(n_1) + n_2 \log_2(n_2)$, где n_1 - количество различных операторов, n_2 - количество различных операндов;

- объем модуля как количество символов для записи всех операторов и операндов текста программы ($V = N \times \log_2(n_1 + n_2)$).

Метрики Холстеда вычисляются на основе анализа количества строк и синтаксических элементов исходного кода программы.

Основу метрик Холстеда составляют 4 измеряемые характеристики программы: $NUOprtr$ - количество уникальных операторов программы, включая имена подпрограмм (словарь операторов), $NUOprnd$ - количество уникальных операндов программы (словарь операндов), $NOprtr$ - общее количество операторов в программе, $NOprnd$ - общее количество операндов программы.

На основе этих характеристик вычисляются следующие оценки:

- словарь программы $HPVoc = NUOprtr + NUOprnd$;

- длина программы $HPLen = NOprtr + NOprnd$ - влияет на оценку величины изменений объема кода во времени;

- объем программы $HPVol = HPLen \times \log_2 HPVoc$ - влияет на оценку величины изменений объема кода во времени;

- сложность программы $HDiff = (NUOprtr/2) \times (NOprnd/NUOprnd)$;

- оценка усилий программиста при разработке программы $HEff = HDiff \times HPVol$ - указывает, насколько эффективна работа разработчика. Используется для определения сложности реализации определенного требования, функции, модуля, части проекта. Влияет на точность прогнозов оценки трудоемкости и на понимание того, насколько интеллектуально затратной для разработчика будет та или иная функция.

3) *метрика Маккейба* - цикломатическая сложность. Один из наиболее распространенных показателей оценки сложности программных проектов. Цикломатическое число Маккейба показывает необходимое количество проходов для покрытия всех контуров сильносвязанного

графа управления программы, т.е. количество тестовых прогонов программы, необходимых для проведения исчерпывающего тестирования. Позволяет провести оценку трудоемкости реализации отдельных элементов программного проекта, оценку трудоемкости тестирования программного проекта, скорректировать общие показатели оценки длительности и стоимости проекта, оценить связанные риски и принять необходимые управленческие решения.

Метрика Маккейба вычисляется для оценки сложности ПО, исходя из топологии внутренних связей. Вычисление метрики производится на основе графа управления программы (орграф, в котором вычислительные операторы или выражения представляются вершинами, а передачи управления между вершинами представляются дугами). Упрощенная оценка цикломатической сложности $V(G) = E - N + 2$, где E - количество дуг, а N - число вершин в управляющем графе ПО (логические операторы не учитываются).

Автоматизированное вычисление метрики Маккейба сводится к подсчету числа логических операторов и возможного количества путей выполнения программы.

Метрика Маккейба может быть вычислена для модуля, метода и других структурных единиц программного проекта. Она влияет на оценку изменений сложности во времени;

4) *метрика Джилба (составляющая - относительная сложность программы)* - отношение количества межмодульных связей к количеству модулей L_{mod}: $f = \frac{N_{zv}^4}{L_{mod}}$. По другой методике отношение абсолютной логической сложности программы к общему числу операторов L составляет относительную логическую сложность программы $cl = \frac{CL}{L}$.

Чем выше значение относительной модульной сложности проекта, тем большую степень имеет сцепление модулей проекта, а, следовательно, и более сложным является программный проект;

5) *прогнозируемое количество операторов программы* - $N_{прогн} = NF \times N_{eд}$, где NF - количество функций или требований в спецификации программы, $N_{eд}$ - единичное значение количества операторов (среднее число операторов, которые приходятся на одну функцию или требование);

6) *прогнозируемая оценка сложности интерфейсов компонентов ПО* - $C = \frac{NI}{NF \times NI_{eд} \times K_{сл}}$, где NI - общее количество переменных, передаваемых по интерфейсам между компонентами программы; $NI_{eд}$ - единичное значение количества переменных, передаваемых по

интерфейсам между компонентами (среднее количество переменных, передаваемых по интерфейсам, приходящихся на одну функцию или требование), выбирают на основе анализа статистических данных; $K_{сл}$ - коэффициент сложности разрабатываемой программы, который учитывает рост единичной сложности программы (сложности, приходящейся на одну функцию или требование спецификации) по сравнению со средним показателем сложности; его следует выбирать с учетом статистических данных и характеристик разрабатываемого ПО;

7) *общее время разработки ПО* - метрика процесса разработки ПО;

8) *время этапа проектирования ПО* - метрика процесса разработки ПО;

9) *ожидаемая стоимость разработки ПО*: $СТОИМОСТЬ_i = LOC_{ожс_i} \times СТОИМОСТЬ_{строки}$, стоимость строки является константой и не меняется от реализации к реализации;

10) *прогнозируемая стоимость проверки качества* - метрика процесса разработки ПО;

11) *прогнозируемая производительность разработки ПО* (на основе производительности аналогичных функций в аналогичных программных продуктах): $ПРОДУКТ_i = ПРОДУКТ_{ан_i} \times (LOC_{ан_i} / LOC_{ожс_i})$;

12) *прогнозируемые затраты на реализацию кода*: $ЗАТРАТЫ_i = (LOC_{ожс_i} / ПРОДУКТ_i)$

13) *прогнозируемый функциональный размер FP* - измеряет суть возможностей будущей программы. Для вычисления функционального размера: идентифицируются ожидаемые от программного приложения функции по критериям International Function Point Users Group (IFPUG) [14]. Метод определения функционального размера [15] опишем пошагово следующим образом:

- выделить функции приложения;

- для каждой потенциальной выделенной функции следует посчитать количество внешних входов EI, которые по-разному влияют на выполняемую функцию, количество внешних выходов EO для существенно различных алгоритмов и нетривиальной функциональности, количество внешних запросов EIN, количество внутренних логических файлов или уникальных логических групп пользовательских данных ILF, количество внешних логических файлов или уникальных логических групп пользовательских данных ELF;

- каждый из определенных на предыдущем шаге факторов умножается на коэффициент, который определяется сложностью данного фактора в программном проекте [15] (EI - 3 или 4 или 6 (от простого к сложному), EO - 4 или 5 или 7, EIN - 3 или 4 или 6, ILF - 7 или 10 или

15, *ELF* - 5 или 7 или 10). Эти произведения добавляются по каждой функции;

- определить значения для 14 общих характеристик проекта (от 0 до 5) [15]; веса указываются в форме диапазонов, отражающей неуверенность относительно функций; назначение весов требует определенного опыта в использовании метода функционального размера;

- вычислить уточненный функциональный размер по формуле: $Уточн.функц.размер = Приближенный\_функц\_размер \times [0.65 + 0.01 \times$

$\times (Сума\_общих\_характеристик]$

Если к проекту не выдвигаются никакие специальные требования (все общие характеристики равны 0), то неуточненный функциональный размер следует уменьшить на 35%, иначе следует увеличить на 1% на каждую единицу значений общих характеристик.

Функциональный размер используется как относительная метрика для сравнения с предыдущими проектами, с его помощью можно вычислить количество строк кода, что позволяет определить общую трудоемкость и сроки проекта. Есть свободно распространяемые инструменты для вычисления функционального размера [16];

14) *прогнозируемая оценка трудозатрат по модели Боэма* [15, 17] - трудозатраты на разработку программных приложений растут быстрее, чем размер приложений. Для представления данного соотношения используется экспоненциальная функция со значением показателя, близким к 1,12: $Трудозатраты = a \times KLOC^b$, где $KLOC$ - количество тысяч строк кода, a,b - коэффициенты COCOMO [17]. Для органического (самостоятельного) программного проекта: $a = 2,4; b = 1,05$. Для встроенных программных проектов (интеграция аппаратного и программного обеспечения): $a = 3,6; b = 1,20$. Для промежуточных программных проектов (не органические, но и не жестко встроенные): $a = 3,0; b = 1,12$.

15) *прогнозируемая оценка длительности проекта по модели Боэма* [15, 17] - длительность проекта по модели Боэма растет экспоненциально вместе с приложенными к проекту усилиями. Однако в данном случае значение экспоненты меньше 1 и составляет около 0,35:

$Длительность = c \times Трудозатраты^d = c \times (a \times KLOC^b)^d = c \times a^d \times KLOC^{b \times d}$

c,d - коэффициенты COCOMO [17]. Для органического (самостоятельного) программного проекта: $c = 2,5; d = 0,38$. Для встроенных программных проектов (интеграция аппаратного и программного обеспечения): $c = 2,5; d = 0,32$. Для промежуточных программных проектов (не органические, но и не жестко встроенные): $c = 2,5; d = 0,35$.

Интеллектуальный метод оценивания результатов проектирования и прогнозирования характеристик качества программного обеспечения (ИМОП). Принимая во внимание результаты анализа методов и средств оценки сложности и качества ПО, очевидно, что необходимо разработать методы и средства оценивания результатов проектирования и прогнозирования характеристик качества ПО. Перспективным направлением исследований является разработка интеллектуальных методов и систем, которые:

1) будут вычислять расчетными и экспертными методами точные или прогнозируемые значения метрик программного обеспечения уже на этапе проектирования;

2) на основе полученных значений метрик будут предоставлять рекомендации и выводы о разрабатываемом программном обеспечении, а также будут анализировать и обрабатывать результаты метрических оценок.

Исходя из анализа метрик, значения которых доступны на этапе проектирования, можно получить оценки, характеризующие этап проектирования, который осуществляется конкретной софтверной компанией, и получить прогнозируемые оценки качества разрабатываемого ПО по результатам этапа проектирования. Полученные оценки результатов проектирования характеризуют уровень софтверной компании и серьезности отношения компании к данному заказу, а также предоставляют данные заказчику для выбора лучшей софтверной компании для разработки необходимого проекта и ПО. Прогнозируемые оценки характеристик качества разрабатываемого ПО предоставляют прогноз относительно качества реализации конкретной версии проекта и позволяют сравнить между собой различные версии проекта с такой точки зрения.

Суть интеллектуального метода оценки результатов проектирования и прогнозирования характеристик качества программного обеспечения [10, 18] заключается в оценивании результатов проектирования и прогнозировании характеристик качества ПО на основе метрического анализа. Для оценивания и прогнозирования сложности и качества ПО на основе метрического анализа следует решить задачу определения взаимосвязи между значениями метрик и такими характеристиками, как качество и сложность проекта ПО и уже разработанного ПО. Одним из средств, которое позволяет обобщить информацию и выявить зависимости между входными и результирующими данными, являются искусственные нейронные сети.

ИМОП использует искусственную нейронную сеть (ИНС), которая осуществляет апрокисмацию метрик, характеризующих ПО на этапе проектирования, в результате чего получаем оценку качества этапа проектирования и прогноз характеристик качества разрабатываемого ПО, т.е. ИМОП позволяет оценить проект и спрогнозировать характеристики

сложности и качества разрабатываемого ПО на основе точных или прогнозируемых значений метрик сложности и качества ПО этапа проектирования.

Для формирования входных данных ИНС понадобятся: множество метрик этапа проектирования с точными значениями (множество $TMP = \{ tmp_i \mid i = 1..9 \}$) и множество метрик этапа проектирования с прогнозируемыми значениями (множество $PMP = \{ pmp_j \mid j = 1..15 \}$).

Из полученных множеств TMP, PMP формируются векторы, которые подаются на вход искусственной нейронной сети (ИНС). Результатом работы ИНС является оценка сложности и качества проекта на основе точных метрик этапа проектирования и прогноз сложности и качества будущего программного обеспечения на основе прогнозируемых метрик этапа проектирования (рис.1).

ИНС для обработки метрик этапа проектирования ПО имеет 9 входов x' и 15 входов x, поскольку на входы x' подаются количественные значения метрик этапа проектирования с точными значениями, которых в предварительном анализе обнаружено 9, а на входы x подаются количественные значения метрик этапа проектирования с прогнозируемыми значениями, которых в предыдущем анализе обнаружено 15.

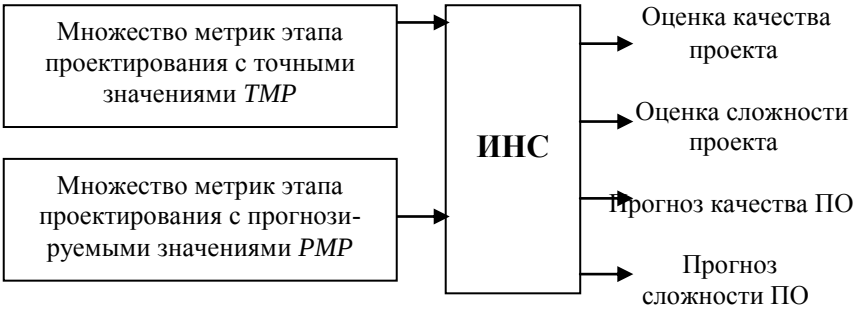

Рис.1. Нейросетевая составляющая ИМОП

Входные данные для ИНС подаются в виде следующих множеств:

- множество метрик сложности этапа проектирования с точными значениями $CMEV = \{ cmev_i \mid i = 1..4 \}$, где $cmev_i$ - количественное значение i-й метрики сложности с точным значением, если эта метрика определялась, иначе (-1);

- множество метрик качества этапа проектирования с точными значениями $QMEV = \{ qmev_j \mid j = 1..5 \}$, где $qmev_j$ - количественное

значение j-й метрики качества с точным значением, если эта метрика определялась, иначе (-1);

- множество метрик сложности этапа проектирования с прогнозируемыми значениями $CMPV = \{ cmpv_k \mid k = 1..6 \}$, где $cmpv_k$ - количественное значение k-й метрики сложности с прогнозируемым значением, если эта метрика определялась, иначе (-1);

- множество метрик качества этапа проектирования с прогнозируемыми значениями $QMPV = \{ qmpv_n \mid n = 1..9 \}$, где $qmpv_n$ - количественное значение n-й метрики качества с прогнозируемым значением, если эта метрика определялась, иначе (-1).

Результатами обработки этих множеств являются:

- оценка сложности проекта PCE в диапазоне [0, 1], где 0 - метрики сложности с точными значениями не определялись, около 0 - проект сложный для реализации и предполагает высокую стоимость реализации, а 1 - проект прост для реализации;

- оценка качества проекта PQE в диапазоне [0, 1], где 0 - метрики качества с точными значениями не определялись, около 0 - проект некачественный, а 1 - проект удовлетворяет требования заказчика по качеству;

- прогноз сложности разрабатываемого программного обеспечения SCP в диапазоне [0, 1], где 0 - метрики сложности с прогнозируемыми значениями не определялись, около 0 - будущее ПО будет иметь существенную сложность, а 1 - будущее ПО ожидается простым;

- прогноз качества разрабатываемого программного обеспечения SQP в диапазоне [0, 1], где 0 - метрики качества с прогнозируемыми значениями не определялись, около 0 - будущее ПО будет некачественным, а 1 - будущее ПО ожидается высокого качества.

Основой для получения оценки сложности проекта являются элементы множества $CMEV$. Основой для получения оценки качества проекта являются элементы множеств $CMEV$ и $QMEV$. Основой для получения прогноза сложности разрабатываемого ПО являются элементы множества $CMPV$, но учитываются и элементы множеств $CMEV$ и $QMEV$. Основой для получения прогноза качества разрабатываемого ПО являются элементы множеств $CMPV$ и $QMPV$, но учитываются и элементы множеств $CMEV$ и $QMEV$.

На основе анализа 4-х полученных результатов делается вывод о качестве и сложности проекта и ожидаемых качестве и сложности разрабатываемого по проекту программного обеспечения.

Итак, ИМОП состоит из следующих этапов: 1) подготовка метрик этапа проектирования с точными и прогнозируемыми значениями для представления на входы ИНС; 2) проверка, не выходят ли полученные

значения метрик за пределы диапазонов значений входов ИНС; 3) обработка значений метрик искусственной нейронной сетью, 4) анализ результатов функционирования ИНС; 5) формирование вывода о сложности и качестве проекта и разрабатываемого ПО на основе результатов ИНС.

Реализация и исследование нейросетевой составляющей ИМОП. В результате анализа известных архитектур искусственных нейронных сетей была выбрана архитектура для анализа метрик этапа проектирования ПО и прогнозирования характеристик качества ПО.

Для выбора архитектуры для анализа метрик этапа проектирования ПО и прогнозирования характеристик качества ПО был проведен анализ известных архитектур искусственных нейронных сетей [19]. Регрессионные радиальные базисные сети используются для анализа числовых рядов. Вероятностные радиальные базисные сети предназначены для решения вероятностных задач, в частности, задач классификации. Карта Кохонена предназначена для кластеризации данных. Сети для классификации входных векторов используются для кластеризации и классификации. Сети Элмана и Хопфилда - это сети с динамическими обратными связями, ориентированные на обработку динамических моделей, учитывающих предысторию процессов. Поскольку задаче анализа метрик и прогнозирования характеристик качества ПО не присущи свойства числового ряда, отсутствуют обратные связи и нет необходимости в классификации и кластеризации данных, то для решения такой задачи выберем многослойный персептрон. При использовании нейросети другого типа для решения этой задачи ее природа будет искусственно искажаться, в результате чего результаты работы ИНС будут неподходящими.

В качестве ИНС анализа метрик и прогнозирования характеристик качества ПО используется многослойный (трехслойный) персептрон (рис.2).

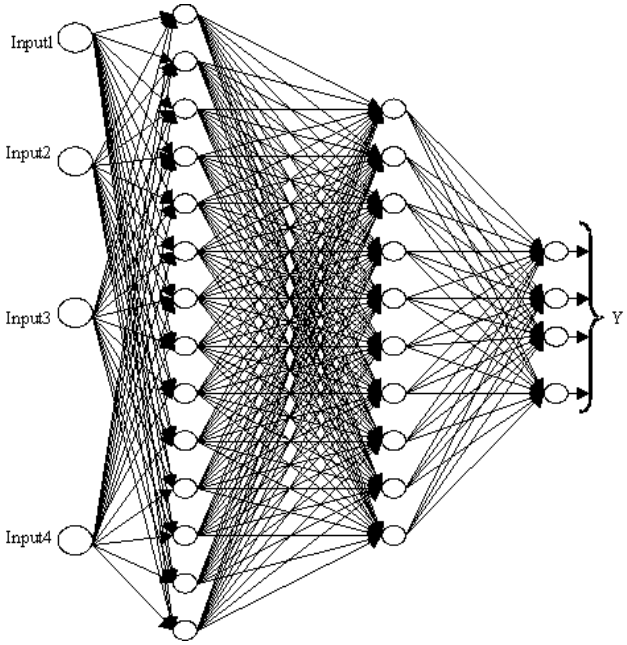

Рис. 2. Архитектура ИНС для анализа метрик и прогнозирования характеристик качества ПО

Прежде чем приступить к реализации НС, следует определить диапазоны ее входных значений. Для определения диапазонов значений используемых метрик этапа проектирования введем определенные предположения и наложим ограничения на проекты и ПО [18, 20], которые будут анализироваться с помощью интеллектуального метода оценивания результатов проектирования и прогнозирования качества программного обеспечения (ИМОП): 1) 100 переменных для расчетов и вывода, 100 модифицированных или созданных в модуле переменных, 100 управляющих переменных и 100 неиспользуемых в модуле ("паразитных") переменных для каждого модуля программы; 2) ПО имеет не более 50 модулей; 3) каждый модуль ПО связан с другими 49 модулями одной связью; 4) в каждом модуле есть не более 50 процедур, которые обновляют структуру данных, не более 50 процедур, которые читают информацию из структуры данных, и не более 50 процедур, которые читают и обновляют структуру данных; 5) каждый модуль реально получает доступ к глобальной переменной столько раз, сколько может получить такой доступ; 6) ПО содержит не более 50000 строк исходного кода; 7) на постановку задачи требуется 10% времени, на проектирование - 35% времени, на программирование - 35% времени и на тестирование, отладку и проверку качества - 20% времени; 8) 25%

времени проектирования расходуется на построение и модификацию моделей; 9) максимальное количество ошибок моделей и прототипов одного модуля не должна превышать 100; 10) количество уникальных операторов программы, включая имена подпрограмм, не превышает 25000, общее количество операндов программы не превышает 50000, количество уникальных операндов программы не превышает 400; 11) каждая строка программы - это логический оператор или оператор цикла, т.е. количество логических операторов или операторов цикла не превышает 50000; 12) количество операторов программы не превышает 50000; 13) каждая переменная модуля передается по его интерфейсу; 14) максимальной ожидаемой стоимостью разработки ПО в долларах США будем считать количество строк исходного кода, поделенную пополам; 15) 50% времени этапа тестирования, отладки и проверки качества занимает проверка качества ПО; 16) количество внешних входов каждого модуля - 49, количество внешних выходов каждого модуля - 49, количество внешних запросов каждого модуля - 49; 17) каждый модуль имеет максимум 50 внутренних логических файлов и использует 50 внешних логических файлов; 18) тип проекта не задан.

Используя вышеприведенные формулы для расчета метрик, получим следующие диапазоны значений метрик этапа проектирования программного обеспечения (таблица 1).

Таблица 1. Интервалы значений метрик ПО этапа проектирования

№	Метрики с точными значениями	Метрики с прогнозируемыми значениями
(1)	(2)	(3)
1	Метрика Чепина: -1, 0..32500	Ожидаемая LOC-оцінка: -1, 0..50000
2	Метрика Джилба (абсолютная): -1, 0..2450	Метрика Холстеда: -1, 0..1562500
3	Метрика Мак-Клура: -1, 0..120050	Метрика Мак-Кейба: -1, 0..2402
4	Метрика Кафура: -1, 0..497500	Метрика Джилба (относительная): -1, 0..1
5	Метрика связности: -1, 0..10	Количество операторов программы: -1, 0..50000
6	Метрика сцепления: -1, 0..9	Оценка сложности интерфейсов: -1, 0..1
7	Метрика обращения к глобальным переменным: -1, 0..1	Общее время разработки ПО: -1, 0..520 (рабочих дней)
8	Время модификации моделей: -1, 0..46	Время этапа проектирования: -1, 0..182 (рабочих дней)

(1)	(2)	(3)
9	Количество обнаруженных ошибок при инспектировании моделей и прототипов: -1, 0..5000	Продуктивность разработки ПО: -1, 0..5 (минут на одну строку кода)
10		Стоимость проверки качества ПО: - 1, 0..20000 (грн)
11		Стоимость разработки ПО: -1, 0..200000 (грн)
12		Стоимость реализации кода программы: -1, 0..70000 (грн)
13		Функциональный размер (FP): -1, 0..2945
14		Оценка трудозатрат по модели Боэма: -1, 0..394 (человекомесяцев)
15		Оценка длительности проекта по модели Боэма: -1, 0..520 (рабочих дней)

Если ввести другие предположения и ограничения на проекты и ПО, обрабатываемые с помощью ИМОП, то получим другие диапазоны значений входных данных ИНС. После этого нужно будет внести соответствующие коррективы в Matlab-программу, которая моделирует данную ИНС, а потом выполнить повторное обучение ИНС.

Таблица 1 показывает, что существует большой разброс диапазонов значений входных данных - значения входных векторов различаются в десятки, сотни, тысячи и даже миллионы раз.

ИНС реализована в пакете Matlab [18, 20-22]. Для создания шаблона ИНС использовалась функция `network`. Были определены количество входных векторов (`net.numInputs = 4`), слоев (`net.numLayers = 4`), элементов каждого входного вектора ИНС:

```
net.inputs{1}.size=4;
net.inputs{2}.size=5;
net.inputs{3}.size=6;
net.inputs{4}.size=9.
```

Матрица связности для смещений:

```
net.biasConnect=[1;1;1;1].
```

Матрицы связности для входов, слоев, выходов и целей заданы следующим образом:

```
net.inputConnect = [1 1 1 1; 0 0 0 0; 0 0 0 0; 0 0 0 0];
```

```
net.layerConnect = [0 0 0 0; 1 0 0 0; 0 1 0 0; 0 0 1 0];
net.outputConnect = [0 0 0 1];
net.targetConnect = [0 0 0 1].
```

Диапазоны значений входных векторов:

```
net.inputs{1}.range = [0 32500; 0 2450; 0 120050; 0 497500];
net.inputs{2}.range = [0 10; 0 9; 0 1; 0 46; 0 5000];
net.inputs{3}.range = [0 50000; 0 1562500; 0 2402; 0 1; 0 50000; 0 1];
net.inputs{4}.range = [0 520; 0 182; 0 200000; 0 20000; 0 5; 0 70000; 0 2945; 0 394; 0 24].
```

Количество нейронов в слоях ИНС:

```
net.layers{1}.size=24;
net.layers{2}.size=14;
net.layers{3}.size=8;
net.layers{4}.size=4.
```

В качестве функции инициализации для каждого слоя ИНС использована функция Нгуен-Видроу (`initnw`). Для 1-3-го слоев в качестве активационной функции выбрана функция гиперболического тангенса, для 4-го (выходного) слоя активационной является пороговая линейная функция.

Представления ИНС предоставлены пакетом Simulink (рис.3 - 6).

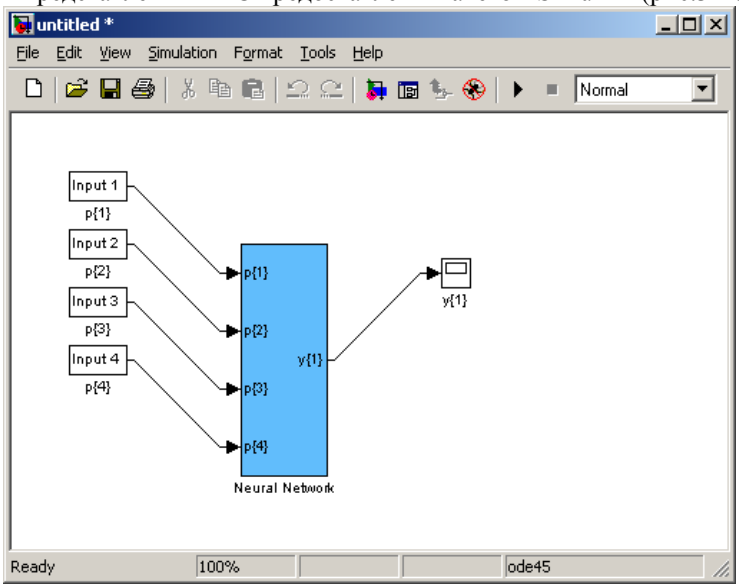

Рис.3. Архитектура ИНС в пакете Simulink

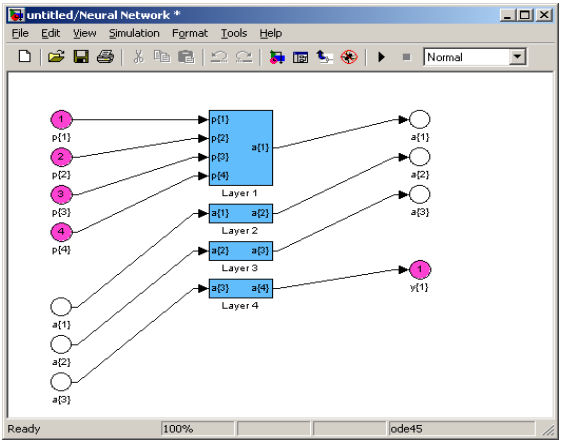

Рис.4. Структурная схема слоев ИНС

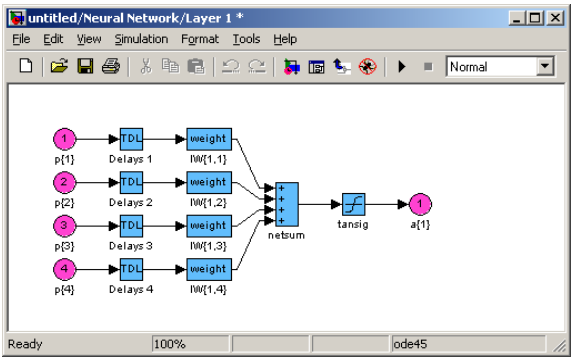

Рис.5. Структурная схема первого слоя ИНС

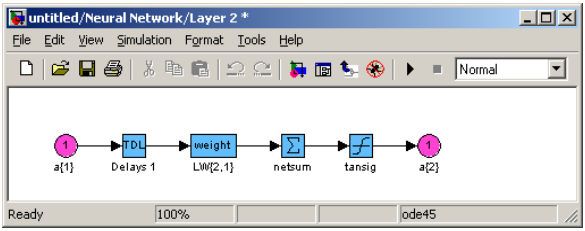

Рис.6. Структурная схема второго слоя ИНС

Структура третьего и четвертого слоев аналогична структуре второго слоя.

Для обучения полученной ИНС последовательность обучающих векторов (обучающая выборка) задана в виде:

`c={[32500;0;0;0] [30875;0;0;0] […];` - обучающие векторы для входа Input1;

`[0;0;0.15;0;0] [0;0;0.2;0;0] […];` - обучающие векторы для входа Input2;

`[3450;0;0;0;0;0] [5900;0;0;0;0;0] […];` - обучающие векторы для входа Input3;

`[0;0;0;0;1.7;0;0;0;0] [0;0;0;0;1.8;0;0;0;0] […];` - обучающие векторы для входа Input4.

Целевой вектор определен как:

`m={[0.05;0.02;0.01;0.01] [0.1;0.04;0.02;0.02] […]}`.

Качество решений, предоставляемых ИНС, существенно зависит от качества информации, используемой для ее обучения. Поэтому актуальной задачей при реализации ИМОП является подготовка учебной и тестовой выборок с данными о значениях метрик этапа проектирования ПО.

Процессы обучения и тестирования ИНС немасштабируемыми выборками по алгоритму градиентного спуска с выбором параметра скорости настройки (`traingda`) показаны на рис.7 (нижняя линия отображает обучение, а верхняя линия - тестирование ИНС). Обучающая выборка состоит из 1935 векторов, тестовая выборка - из 324 векторов.

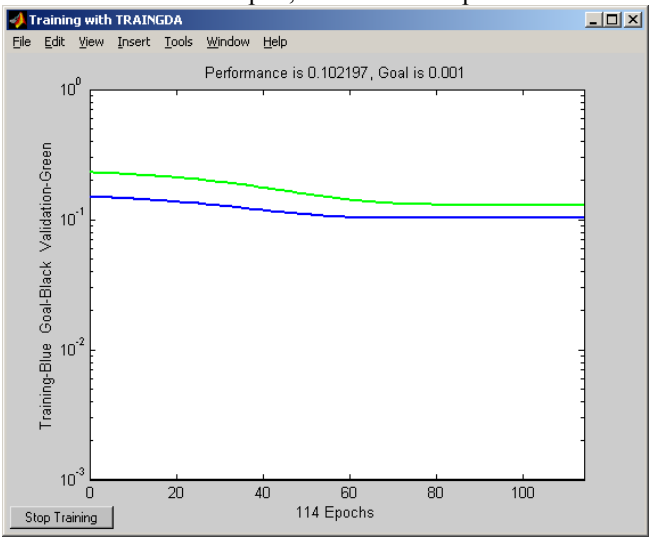

Рис.7. Процессы обучения и тестирования ИНС немасштабируемыми выборками

В результате обучения получим ИНС, погрешность обучения которой составляет примерно 0,102197.

Протестируем работу нейросети на двух выборках, которые отличаются лишь значением одной из метрик, которая имеет малый диапазон значений и на основе которой рассчитывается качество проекта.

Первая выборка:

c1={[1625;170;6050;30000]; - вектор для входа Input1;

[10;1;**0**;2;250]; - вектор для входа Input2;

[5900;156251;240;0.1;5000;0.1]; - вектор для входа Input3;

[52;19;20000;2000;0.6;7000;280;43; 3]}; - вектор для входа Input4.

Вектор выхода после обработки первой выборки:

m1={[0.95;0.97;0.901;0.91]}.

Подадим на входы нейросети вторую выборку:

c2={[1625;170;6050;30000];
[10;1;**1**;2;250];
[5900;156251;240;0.1;5000;0.1];
[52;19;20000;2000;0.6;7000;280;43; 3]};

Вектор выхода после обработки второй выборки:

m2={[0.95;0.97;0.901;0.91]}.

Как видно, вектор выхода нейросети абсолютно не изменился - и для первой, и для второй выборок он говорит о высоком качестве проекта и ПО, хотя значение метрики обращения к глобальным переменным изменилось с минимального значения, которое свидетельствует о высоком качестве проекта, на максимальное значение, которое свидетельствует о низком качестве проекта.

Рассмотрим другой пример.

Подадим на входы нейросети выборку:

c1={[29250;2210;108050;449500];
[1;7;0.9;41;4500];
[45100;**1406251**;2160;1;45000;0.9];
[468;163;180000;18000;4.6;63000; 2660;361;21]};

Вектор выхода после обработки такой выборки:

m1={[0.1;0.11;**0.104**;0.101]}.

Подадим на входы нейросети другую выборку, которая снова-таки будет отличаться значением только одной из метрик, которая имеет большой диапазон значений и на основе которой прогнозируется сложность разрабатываемого ПО.

c2={[29250;2210;108050;449500];
[1;7;0.9;41;4500];
[45100;**78126**;2160;1;45000;0.9];
[468;163;180000;18000;4.6;63000; 2660;361;21]};

Вектор выхода после обработки этой выборки:

m2={[0.1;0.11;**0.54**;0.101]}.

В этом случае вектор выхода нейросети существенно изменился - для первой выборки он свидетельствует о высокой сложности проекта и ПО, для второй выборки - о высокой сложности проекта и средней сложности разрабатываемого ПО, хотя изменилось значение лишь одной метрики - метрики Холстеда - с максимального значения, которое свидетельствует о высокой сложности разрабатываемого ПО, на близкое к минимальному значение, которое свидетельствует о простоте разрабатываемого ПО.

Проведены также другие эксперименты, в которых существенно изменялись значения метрик с малыми диапазонами значений, но выходные векторы ИНС оставались неизменными, а также эксперименты, в которых существенно менялись значения метрик с большими диапазонами значений, что приводило к существенному изменению определенных значений выходных векторов. Итак, если подавать на входы ИНС данные в немасштабируемом виде, то ИНС становится нечувствительной ко входам с малыми диапазонами значений, т.е. метрики связности, сцепления, обращения к глобальным переменным, времени модификации моделей, Джилба (относительная), прогнозируемой оценки сложности интерфейсов, производительности разработки ПО не влияют на результаты работы ИНС, а определяющими при расчете результатов нейросети являются входы (метрики) с большими диапазонами значений.

Очевидно, что следует определенным образом обрабатывать входные данные ИНС. Выполним обработку учебной и тестовой выборок ИНС с помощью масштабирования Matlab-функцией `premnmx` [19] - препроцессорная обработка обучающей выборки путем приведения значений элементов векторов входа и цели к интервалу [-1 .. 1].

Рис.8 отображает процессы обучения и тестирования ИНС масштабируемыми (функцией `premnmx`) выборками по алгоритму градиентного спуска с выбором параметра скорости настройки (`traingda`).

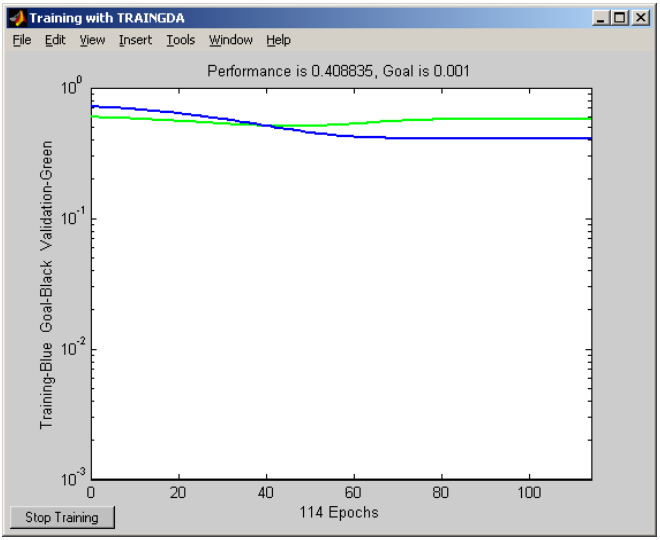

Рис.8. Процессы обучения и тестирования ИНС масштабируемыми (функцией `premnmx`) выборками

Погрешность обучения, отражающая реакцию ИНС на входные данные, составляет примерно 0,408835.

Протестируем работу нейросети на двух выборках, которые отличаются только одним значением - одной из метрик с малым диапазоном значений, на основе которой рассчитывается качество проекта. Первая выборка:

```
c1={[1625;170;6050;30000];
[10;1;0;2;250];
[5900;156251;240;0.1;5000;0.1];-
[52;19;20000;2000;0.6;7000;280;43; 3]};
```

Вектор выхода после обработки первой выборки:

```
m1={[0.95;0.97;0.901;0.91]}.
```

Подадим на входы нейросети вторую выборку:

```
c2={[1625;170;6050;30000];
[10;1;1;2;250];
[5900;156251;240;0.1;5000;0.1];
[52;19;20000;2000;0.6;7000;280;43; 3]};
```

Вектор выхода после обработки второй выборки:

```
m2={[0.95;0.547;0.901;0.91]}.
```

Результаты функционирования ИНС для первой выборки свидетельствует о высоком качестве проекта и ПО, а выходной вектор нейросети для второй выборки свидетельствует о среднем качестве проекта, хотя изменилось значение лишь одной метрики - метрики

обращения к глобальным переменным - с минимального значения, свидетельствующего о высоком качестве проекта, на максимальное значение, свидетельствующее о низком качестве проекта.

Рассмотрим другой пример.

Подадим на входы ИНС выборку:

```
c1={[29250;2210;108050;449500];
[1;7;0.9;41;4500];
[45100;1406251;2160;1;45000;0.9];
[468;163;180000;18000;4.6;63000; 2660;361;21]};
```

Вектор выхода после обработки такой выборки:

```
m1={[0.1;0.11;0.104;0.101]}.
```

Подадим на входы ИНС другую выборку, которая вновь будет отличаться только одним значением - одной из метрик с большим диапазоном значений, на основе которой прогнозируется сложность разрабатываемого ПО:

```
c2={[29250;2210;108050;449500];
[1;7;0.9;41;4500];
[45100;78126;2160;1;45000;0.9];
[468;163;180000;18000;4.6;63000; 2660;361;21]};
```

Вектор выхода после обработки этой выборки:

```
m2={[0.1;0.11;0.104;0.101]}.
```

В этом случае выходной вектор нейросети абсолютно не изменился - он свидетельствует о высокой сложности проекта и ПО, хотя существенно изменилось значение метрики Холстеда - с максимального значения, которое свидетельствует о высокой сложности разрабатываемого ПО, на близкое к минимальному значению, которое свидетельствует о простоте разрабатываемого ПО .

Проведены также другие эксперименты, в которых существенно изменялись значения метрик с малыми диапазонами значений, что приводило к изменениям определенных величин выходных векторов, а также эксперименты, в которых существенно изменялись значения метрик с большими диапазонами значений, что совершенно не влияло на выходные векторы нейросети.

Таким образом, при масштабировании входных значений нейросети функцией `premnmx` входы с малыми диапазонами значений значительно влияют на результаты функционирования ИНС, а входы с большими диапазонами значений вызывают появление нейронов, которые не учитываются при расчете результирующего вектора ИНС. Следовательно, теряется часть значимой информации, поскольку метрики с большими диапазонами значений (метрики Чепина, Мак-Клура, Кафура, LOC-оценка, Холстеда, прогнозируемого количества операторов программы, ожидаемой стоимости разработки ПО, прогнозируемой стоимости проверки качества ПО, прогнозируемых расходов на реализацию кода) не влияют на результаты работы нейросети.

Исследования показали, что при обработке немасштабируемых входных выборок ИНС нечувствительна к входам с малыми диапазонами значений, а входы с большими диапазонами определяют результаты работы ИНС. При обработке масштабируемых входных выборок именно входы с малыми диапазонами значений являются определяющими, а входы с большими диапазонами значений не влияют на работу нейросети, т.е. встроенная в пакет Matlab функция масштабирования входов нейросети `premnmx` не подходит для обработки значений метрик программного обеспечения. Встроенная в Matlab функция нормирования `prestd` может обрабатывать входные значения, которые имеют нормальный закон распределения, следовательно, также не подходит для обработки значений метрик ПО. Встроенная функция факторного анализа `prepca` также не может обрабатывать значения метрик ПО, поскольку для ее применения входные значения не должны коррелировать между собой.

Итак, ни одна из встроенных функций пакета Matlab не подходит для обработки значений метрик ПО этапа проектирования, но подготовка входной информации для данной ИНС необходима. Следовательно, необходимо разработать новую или модифицировать одну из встроенных функций подготовки входной информации ИНС таким образом, чтобы она равнозначно учитывала входные данные как с большими, так и с малыми диапазонами для предотвращения потери значимой информации.

Интеллектуальная система оценивания и прогнозирования сложности и качества программного обеспечения (ИСОП). Для оценивания результатов проектирования и прогнозирования характеристик сложности и качества ПО на основе обработки метрик этапа проектирования с точными и прогнозируемыми значениями разработана интеллектуальная система оценивания и прогнозирования сложности и качества программного обеспечения (ИСОП) [23-25].

На вход ИСОП подаются количественные значения метрик этапа проектирования с точными и прогнозируемыми значениями, а результатом работы являются выводы о сложности и качестве проекта и разрабатываемого ПО. Структурная схема ИСОП представлена на рис.9.

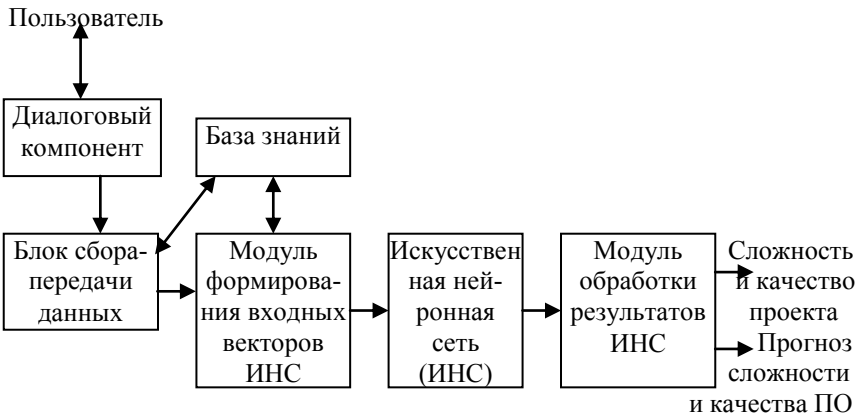

Рис.9. Структурная схема ИСОП

ИСОП состоит из следующих компонентов:

- диалоговый компонент;
- блок сбора-передачи данных;
- база знаний;
- модуль формирования входных векторов для искусственной нейронной сети;
- искусственная нейронная сеть (ИНС);
- модуль обработки результатов ИНС.

Диалоговый компонент визуализирует работу блока сбора-передачи данных, отображает работу системы и выдает пользователю сообщение в понятной для него форме.

Блок сбора-передачи данных считывает информацию пользователя о количественных значений точных и прогнозируемых метрик этапа проектирования ПО, сохраняет полученную информацию в базе знаний и передает ее в модуль формирования входных векторов ИНС.

База знаний содержит количественные значения точных и прогнозируемых метрик этапа проектирования ПО, входные векторы ИНС и правила обработки результатов работы ИНС.

Искусственная нейронная сеть осуществляет апрокисмацию метрик ПО этапа проектирования и дает количественную оценку сложности и качества проекта и прогнозируемые значения характеристик сложности и качества разрабатываемого ПО.

Модуль формирования входных векторов ИНС готовит значения метрик из базы знаний к подаче на входы ИНС.

Система ИСОП разработана на языке программирования Delphi 7.

Первое интерфейсное окно системы выглядит следующим образом (рис.10). Оно имеет справочный характер, поскольку показывает пользователю все метрики, которые обрабатываются системой ИСОП.

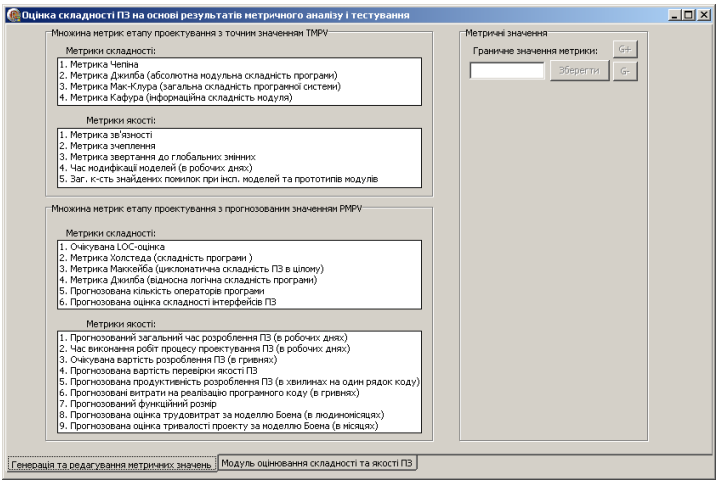

Рис.10. Первое интерфейсное окно системы ИСОП (справочное)

При выборе определенной метрики в правой части окна отображаются предельные значения метрики, предыдущие значения данной метрики, которые были введены при пользовании системой, и значение соответствующего выходного значения системы как реакция именно на эту метрику (рис.11). Если пользователем вводятся несколько метрик определенного типа или же какое-то количество метрик различных типов, то они коррелируют между собой, и исходное значение системы учитывает все введенные значения метрик.

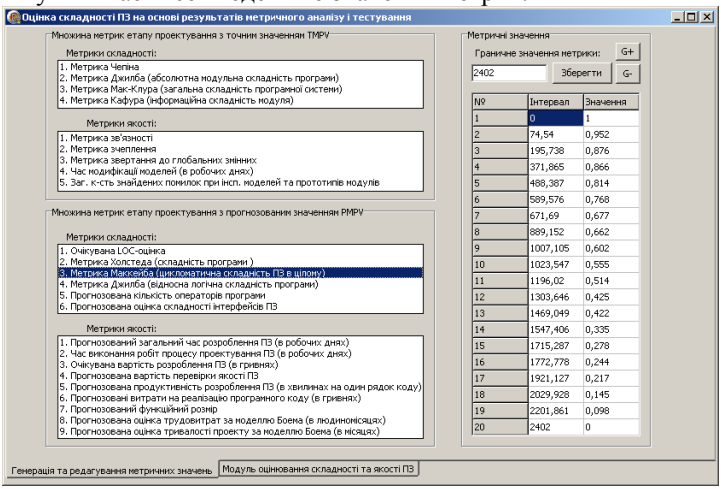

Рис.11. Интерфейсное окно системы с информацией о диапазоне выбранной метрики и с ранее введенными значениями выбранной метрики

Для ввода значений метрик и получения выводов системы следует перейти на вкладку "Редактирование метрических значений", которая в начале работы с системой имеет вид, показанный на рис.12. В левой части окна нужно вводить значения метрик, которые определялись для проекта, или -1, если данная метрика не определялась. После ввода всех метрик следует нажать кнопку "Получить оценку" для получения выводов системы ИСОП.

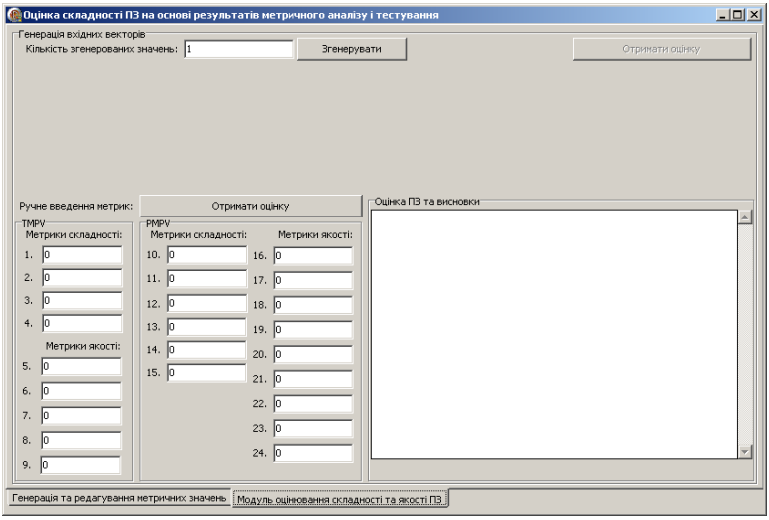

Рис.12. Интерфейсное окно ИСОП "Редактирование метрических значений" до введения значений метрик

В качестве входных данных ИСОП будем использовать результаты метрического анализа 3-х проектов, разработанных софтверной компанией "СТУ-Электроникс" г.Хмельницкого (таблица 2).

Таблица 2. Результаты метрического анализа этапа проектирования

№ пр.	Метрики сложности и качества с точными значениями на этапе проектирования	Метрики сложности и качества с прогнозируемыми значениями на этапе проектирования
(1)	(2)	(3)
1	Метрика Чепина – 22750 Метрика Джилба (абсолютная) – 1730 Метрика Мак-Клура – 84050 Метрика Кафура – не определялась Метрика связности – 3	Ожидаемая LOC-оцінка – 35300 Метрика Холстеда – не определялась Метрика Маккейба – 1680 Метрика Джилба (относительная) - 0,7

(1)	(2)	(3)
	Метрика сцепления – 7 Метрика вызова глобальных переменных – не определялась Время модификации моделей – 33 Количество обнаруженных ошибок при инспектировании моделей и прототипов – не определялась	Ожидаемое количество операторов программы – 35000 Ожидаемая оценка сложности интерфейсов – не определялась Общее время разработки ПО – 364 рабочих дня Время выполнения работ этапа проектирования – 127 рабочих дня Ожидаемая стоимость разработки ПО – 140000 гривен Ожидаемая стоимость проверки качества ПО – не определялась Прогнозируемая продуктивность разработки ПО – не определялась Прогнозируемые затраты на реализацию программного кода – 49000 гривен Ожидаемый функциональный размер FP - 2100 Прогнозируемая оценка трудозатрат по модели Боэма – 283 человекомесяцев Прогнозируемая оценка длительности проекта по модели Боэма – не определялась
2	Метрика Чепина – 16250 Метрика Джилба (абсолютная) – 1250 Метрика Мак-Клура – 60050 Метрика Кафура – 257000 Метрика связности – 5 Метрика сцепления – 4 Метрика вызова глобальных переменных – 0,5 Время модификации моделей – 24 Количество обнаруженных ошибок при инспектировании моделей и прототипов – 2500	Не определялись

(1)	(2)	(3)
3	Не определялись	Ожидаемая LOC-оценка – 10800 Метрика Холстеда – 312501 Метрика Маккейба – 480 Метрика Джилба (относительная) - 0,2 Количество операторов программы – 10000 Ожидаемая оценка сложности интерфейсов – 0,2 Общее время разработки ПО – 104 рабочих дня Время выполнения работ этапа проектирования – 37 рабочих дня Ожидаемая стоимость разработки ПО – 40000 гривен Ожидаемая стоимость проверки качества ПО – 4000 гривен Прогнозируемая продуктивность разработки ПО – 1,1 минута на одну строку кода Прогнозируемые затраты на реализацию программного кода – 14000 гривен Ожидаемый функциональный размер FP - 560 Прогнозируемая оценка трудозатрат по модели Боэма – 80 человекомесяцев Прогнозируемая оценка длительности проекта по модели Боэма – 5 месяцев

Модуль формирования входных векторов ИНС создает следующие векторы для входов ИНС (таблица 3).

Таблица 3. Пример входных векторов ИНС

№	**Input1 (x_1')**	**Input2 (x_2')**	**Input3 (x_1)**	**Input4 (x_2)**
1	{[22750;1730; 84050;-1];	[3;7;-1;33;-1];	[35300;-1; 1680;0.7; 35000;-1];	[364;127;140000; -1;-1;9000;2100; 283;-1]}
2	{[16250;1250; 60050;257000];	[5;4;0.5;24; 2500];	[-1;-1;-1;-1; -1; -1];	[-1;-1;-1;-1;-1;-1; -1;-1;-1]}
3	{[-1;-1;-1;-1];	[-1;-1;-1;-1; -1];	[10800; 312501;480; 0.2;10000;0.2];	[104;37;40000; 4000;1.1;14000; 560;80;5]}

ИНС обрабатывает входные векторы и выдает результаты, на основе которых, согласно правилам базы знаний, модуль обработки результатов работы ИНС делает определенные выводы (таблица 4).

Таблица 4. Результаты ИНС и выводы о проанализированных проектах

№	**Значение $Y_1(OSP)$ и вывод ИСОП**	**Значение $Y_2(OQP)$ и вывод ИСОП**	**Значение $Y_3(PSPZ)$ и вывод ИСОП**	**Значение $Y_4(PQPZ)$ и вывод ИСОП**
1	0.31	0.30	0.34	0.29
	Проект сложный для реализации	Проект низкокачест-венный	Будущее ПО ожидается высокослож-ным	Будущее ПО ожидается низкокачест-венным
2	0.50	0.51	0	0
	Проект имеет среднюю сложность	Проект имеет среднее качество	Метрики сложности с прогнозируе-мыми значени-ями не опреде-лялись	Метрики качества с прогнозируе-мыми значени-ями не опреде-лялись
3	0	0	0.83	0.80
	Метрики сложности с точными значениями не определялись	Метрики качества с точными значениями не определялись	Будущее ПО ожидается достаточно простым	Будущее ПО ожидается высококачест-венным

Как видно из таблицы 4, для проекта № 2 не определялись метрики сложности и качества с прогнозируемыми значениями, поэтому система не смогла сделать прогноз сложности и качества разрабатываемого по

проекту программного обеспечения. Для проекта № 3 не определялись метрики сложности и качества с точными значениями, поэтому система не смогла оценить сложность и качество проекта. Анализ проектов с определенными всеми группами метрик показал, что оценки сложности и качества проекта и оценки сложности и качества будущего ПО практически одинаковы. Согласно полученным результатам, простым и качественным является проект № 3. Программное обеспечение, разработанное по проекту № 3, тоже будет достаточно простым и высококачественным.

Пример обработки метрик реализованной системой ИСОП показан на рис.13. В левой части окна введены значения метрик для проекта 1 (таблица 3), в правой части окна - сформированы выводы системы (таблица 4).

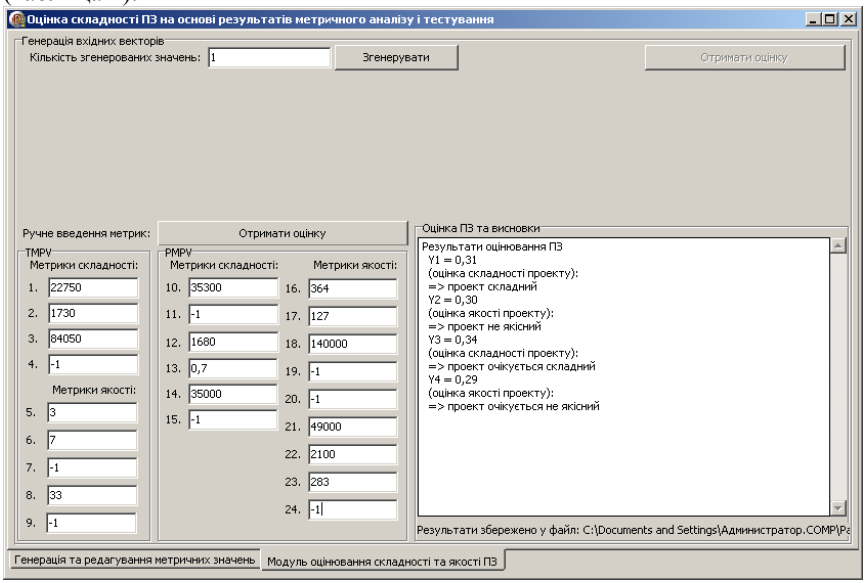

Рис.13. Интерфейсное окно с введенными значениями метрик для проекта 1 и сгенерированными системой выводами

Выводы ИСОП позволяют сравнить между собой различные версии проектов в ситуации, когда стоимость и длительность проектов примерно одинаковы.

Например, рассмотрим данные о 2-х проектах, разработанных софтверной компанией "СТУ-Электроникс" г.Хмельницкого (таблица 5).

Таблица 5.Характеристики вариантов реализации проекта

N	**Характеристики Y_1, Y_2**	**Характеристики Y_3, Y_4**	**Стоимость**	**Время разработки**
1	Y_1=0,85 Y_2=0,86	Y_3=0,81 Y_4=0,87	100000 грн	250 рабочих дней
2	Y_1=0,22 Y_2=0,25	Y_3=0,28 Y_4=0,27	105000 грн	260 рабочих дней

Очевидно, что на основе лишь стоимости и времени разработки софтверная организация может принять ошибочный вывод относительно выбора варианта реализации проекта, и именно выводы ИСОП помогут заказчику сделать правильный выбор проекта.

Предложенная интеллектуальная система оценивания и прогнозирования сложности и качества программного обеспечения дает данные заказчику для выбора проекта необходимого ПО и позволяет сравнить между собой различные версии проекта, то есть дает возможность принять мотивированное и обоснованное решение по выбору проекта и его реализации на основе не только стоимостных и временных характеристик, но и с учетом характеристик сложности и качества проекта и разрабатываемого программного обеспечения.

Литература

1. Скляр В.В. Оценка качества и экспертиза программного обеспечения. Лекционный материал. - Харьков: НАУ "ХАИ", 2008. - 204 с.
2. Майерс Г. Надежность программного обеспечения: Пер. с англ. – М.: Мир, 1980. – 360 с.
3. Myers G.J. The Art of Software Testing. – New York: John Wiley and Sons, 1979. – 312 pp.
4. Липаев В.В. Программная инженерия. Методологические основы. - М.: ТЕИС, 2006. - 608 с
5. Орлов С.А. Технологии разработки программного обеспечения. Разработка сложных программных систем: Учебник для ВУЗов - СПб.: Питер, 2004. - 527 с.
6. ГОСТ 28195-89. Оценка качества программных средств. Общие положения
7. IEEE Standard Glossary of Software Engineering Terminology /IEEE Std 610.12-1990
8. Поморова О.В., Говорущенко Т.О. Аналіз методів та засобів оцінки якості програмних систем // Радіоелектронні і комп'ютерні системи – Харків: НАУ "ХАІ", 2009 – № 6, с.148-158
9. О.В.Поморова, Т.О.Говорущенко, С.Я.Тарасек. Аналіз та опрацювання метрик якості програмного забезпечення на етапі проектування // Вісник Хмельницького національного університету – Хмельницький: ХНУ, 2010. - № 1, с54-63
10. Поморова О.В., Говорущенко Т.О. Інтелектуальний метод оцінювання результатів проектування та прогнозування характеристик якості програмного забезпечення // Радіоелектронні і комп'ютерні системи – Харків: НАУ "ХАІ", 2010 – № 6, с.211-218
11. Yourdon E., Costantine L. Structured Design: fundamentals of a discipline of computer program and systems design - Englewood Cliffs, NJ: Prentice-Hall, 1979. - 513 p.
12. Page-Jones M.The Practical Guide to Structured Systems Design - Englewood Cliffs, NY: Yourdon Pressl, 1988. - 609 p.
13. http://www.construx.com
14. International Function Point User's Group, http://www.ifpug.org/
15. Брауде Э. Технология разработки программного обеспечения - СПб:Питер,2004 -655 с.
16. International Function Point User's Group reference to function point spread-sheets, http://ifpug.org/home/docs/freebies.html
17. Boehm B. Software Engineering Economics - NJ: Prentice Hall, 1981. - 392 p.
18. Поморова О.В., Говорущенко Т.О., Онищук О.С. Оцінювання результатів проектування та прогнозування характеристик якості

програмного забезпечення // Вісник Хмельницького національного університету - Хмельницький: ХНУ, 2011 - №2, с.168-178

19. Медведев В.С., Потемкин В.Г. Нейронные сети Matlab 6 / Под общей редакцией к.т.н. В.Г.Потемкина – М.: Диалог-Мифи, 2002. – 496 с.

20. Поморова О.В., Говорущенко Т.О., Козак М.В. Реалізація та дослідження нейромережної складової методу оцінювання та прогнозування якості програмного забезпечення // Науковий вісник Чернівецького університету. Комп'ютерні системи та компоненти, 2011. Т. 2. Вип. 1.- Чернівці, ЧНУ, 2011 - с.19-26

21. O.Pomorova, T.Hovorushchenko. Research of Artificial Neural Network's Component of Software Quality Evaluation and Prediction Method // - USA, NJ 08855-1331: IEEE Operations Center, 2011 - vol.2, p. 959-962 (IEEE Catalog Number: CFP11803-PRT)

22. O.Pomorova, T.Hovorushchenko. Intelligence Method of Software Quality Evaluation and Prediction // Scientific basis of modern technology: Experience and Prospects. Monograph: edited by Shalapko Y.I., Dobrzanski L.A. - Khmelnitskiy, 2011 - pp. 92-104

23. O.Pomorova, T.Hovorushchenko. The Intelligence System of Software Complexity and Quality Evaluation and Prediction // Proceedings of the 1-st International Workshop "Critical Infrastructure Safety and Security (CrISS-DESSERT-2011)" - Kharkiv: National Aerospace University named after N.E.Zhukovsky "KhAI", 2011 - pp.257-264

24. Говорущенко Т.О. Інтелектуальна система оцінювання і прогнозування складності та якості програмного забезпечення // Комп'ютерно-інтегровані технології: освіта, наука, виробництво - Луцьк: Луцький національний технічний університет, 2011 - №5, с.47-54

25. Говорущенко Т.О. Інтелектуальна система автоматизації аналізу та опрацювання метрик програмного забезпечення // Матеріали 18 Міжнародної конференції з автоматичного управління "Автоматика-2011" - Львів: Видавництво Львівської політехніки, 2011 - с.301-302