СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

УДК 004.5:658.5

МЕТОД ПРОЕКТИРОВАНИЯ КОНТЕКСТНО-ЗАВИСИМЫХ АДАПТИВНЫХ ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ В СОСТАВЕ СИСТЕМЫ УПРАВЛЕНИЯ ПРОЕКТАМИ

М. А. Артемов*, Р. В. Беляев, А. А. Чиченин***

** Воронежский государственный университет*
*** Военный учебно-научный центр Военно-воздушных сил «Военно-воздушная академия имени профессора Н.Е. Жуковского и Ю.А. Гагарина» (г. Воронеж)*

Поступила в редакцию 12.07.2013 г.

Аннотация. В статье рассматривается метод проектирования контекстно-зависимых адаптивных пользовательских интерфейсов, применимый к современным методологиям проектирования и разработки ПО.
Ключевые слова: управление проектами, Пользовательские интерфейсы, Алгоритмы на графах.
Annotation. The paper discusses a method of designing context-aware adaptive user interfaces, applicable to modern software design and development methodologies.
Keywords: project management, UI, Graph based algorithms.

Введение. При разработке программного обеспечения часто возникает проблема определения перечня компонентов интерфейса, доступных тому или иному пользователю и выводимых в том или ином интерфейсе.

Основными причинами данных проблем являются:

1. Отсутствие целостной системы прав и обязонностей пользователя.

2. Отсутсвие целостной системы связей интерфейсов и функций, выполняемых программным продуктом.

Задачи. Проанализировать существующие методы проектирования приложений, выявить зависимости с современными методологиями разработки. Разработать метод проектирования программных продуктов и пользовательских интерфейсов в их составе, совместимый с «Agile» методологиями ведения проектов.

Краткий обзор «Agile». Данный класс методологий предназначен для разбиения сложных программных продуктов на модули с минимальной связанностью и внешними зависимостями, выявления порядка разработки этих модулей и, наконец, планирования необходимых затрат на разработку и тестирования.

Независимо от конкретной методологии, всегдавводится сущность «**компонент**» — совокупность модулей приложения, выполняющая группу тесно связанных функций системы.

Суть метода. Метод расширяет указанный ранее список сущностей «Agile» дополнительными сущностями:

• **Графический интерфейс** («Представление», «Виджет» или «Элемент»). Данные сущности входят в состав «компонентов». При этом дополнительно вводятся сквозные компоненты для всего программного продукта.

• **Контекст.** Сущность предназначена для определения функций, доступных пользователю интерфейса [8] в зависимости от внешних параметров и состояния других компонентов.

• **Функция.** Данная сущщность предназначена для аннотирования компонентов непосредственными действиями, которые могут быть осуществлены пользователями системы. Функция связывает сущности «Контекст», «Компонент» и «Роль».

• **Роль**. Сущность предназначена для группировки функций и позволяет определять права и обязанности различных пользователей.

При этом один пользователь может обладать несколькими ролями.

Все эти сущности представляются в виде связного графа, после чего становится возможным расчёт целого ряда различных характеристик:

1. Похожесть ролей пользователей для компонента. С целью выявления лишних ролей.

2. Похожесть компонентов. С целью выявления функций, которые должны быть реализованы централизовано.

3. Связность функций. С целью выявления групп, которые целесообразно выделять в компоненты.

4. Порядок разработки и тестирования, путём построения ориентированного графа, основывающегося на доступности функций в различных контекстах и изменения контекстов функциями.

5. Определение критического пути и последующая оценка трудозатрат.

6. Получение конечного автомата для проведения автоматического тестирования и вспомогательной информации при ручном тестировании.

В общем виде граф представлен на рисунке.

В частных случаях граф содержит гораздо больше вершин и связей, которые выставляются с учётом характера конкретного исследования. В любом случае опорной сущностью является функция, однако, можно получить подграф связей для любой из нужных сущностей, что позволяет, например:

- Получить список компонентов, реализующих определённую функцию.
- Оценить похожесть компонентов не только по набору реализуемых функций, но и по набору интерфейсов и их компонентов.
- Оценить похожесть ролей, по набору реализуемых функций, с учётом интерфейсов.
- Спланировать разработку отдельных компонентов в составе системы, и их тестирования, с учётом цепочки зависимостей.

Основные шаги. В зависимости от конкретной задачи последовательность шагов может

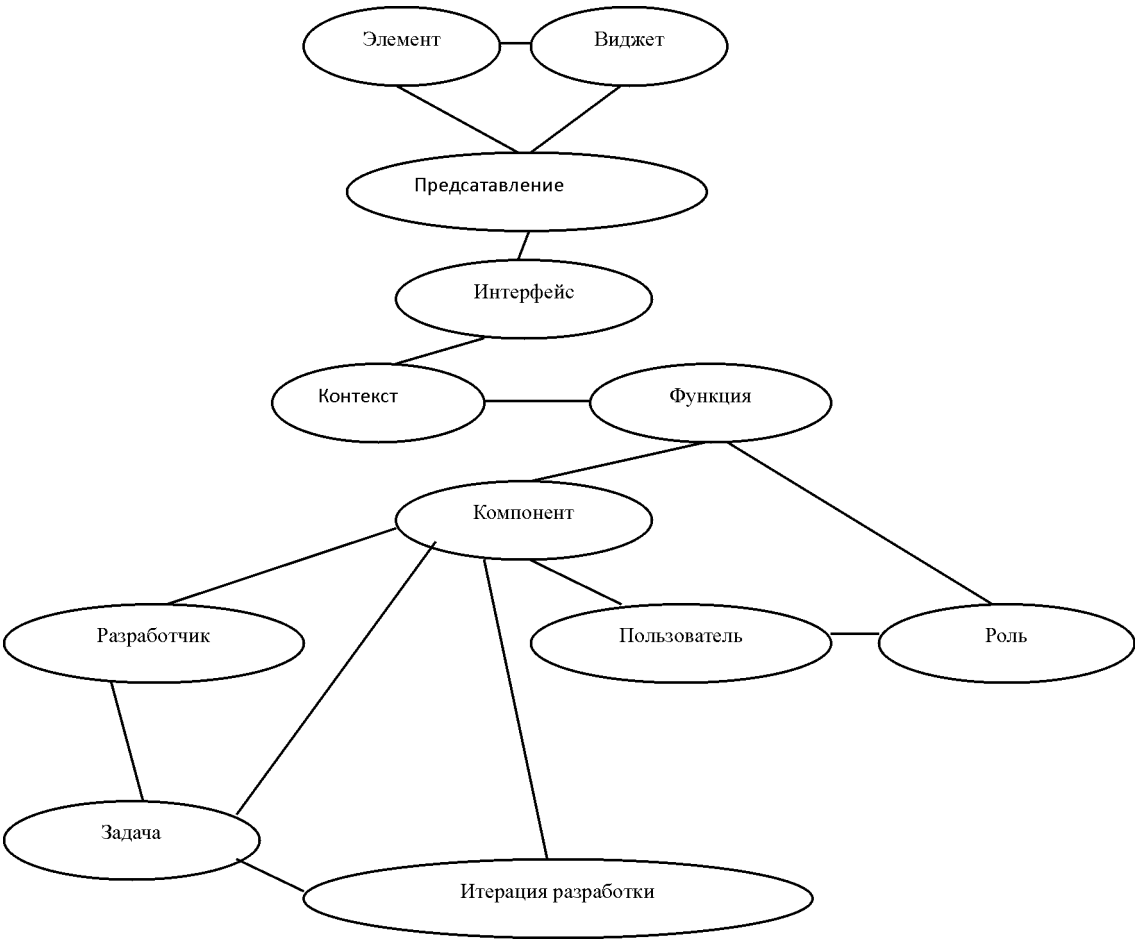

Рис. Граф для общего случая

меняться, но в общем случае она является следующей:

1. Определение линейного списка функций системы.

2. Определение базовых ролей и связей с функциями.

3. Определение линейного списка интерфейсов, необходимых для выполнения той или иной функции.

4. Определение контекстов интерфейсов при выполнении функции, и изменения контекстов функциями.

5. Построение графа связей между определёнными компонентами.

6. Фиксирование параметров системы, или задание пределов из изменения.

7. Проведение расчётов и выявление возможных шагов для оптимизации графа по свободным параметрам.

8. Реверсивное формирование линейных списков сущностей для использования в системах управления проектами и в качестве документации или части технического задания.

Предположим, что необходимо произвести оценку разработки компонента авторизации на сайте, определить очерёдность разработки и набор разрабатываемых интерфейсов. Данный пример не позволит всесторонне продемонстрировать сильные стороны подхода и предназначен для пояснения механизма подготовки данных.

Сначала выделим основные функции для этого компонента:

• F1 — Авторизация зарегистрированного пользователя.

• F2 — Регистрация нового пользователя.

• F3 — Восстановление забытого пароля.

a) Отправка запроса на восстановление.

b) Подтверждение права на восстановление и создание нового пароля.

• F4 — Изменение пароля.

• F5 — Деавторизация пользователя.

Определим перечень ролей и доступные им функции:

• R1 — Неавторизованный пользователь (F1, F3a, F3b).

• R2 —Незарегистрированый пользователь (F2).

• R3 —Авторизованный пользователь (F4, F5).

В данном случае контексты интерфейсов полностью соответствуют ролям, построим список интерфейсов:

• I1 — Интерфейс входа на сайт, с возможностью регистрации и восстановления пароля (R1, R2).

• I2 — Интерфейс регистрации (R1, R2).

• I3 — Интерфейс восстановления пароля (R1, R2).

• I4 — Интерфейс подтверждения восстановления пароля (R1).

• I5 — Интерфейс изменения пароля (R3).

• I6 — Интерфейс деавторизации (R3).

На базе этих данных можно получить граф связности, определить перечень задач для дизайнеров и программистов, создать тестовые сценарии для команды контроля качества.

Продолжим уточнение структуры компонента, определяя элементы интерфейсов:

• E1 — поле ввода имени пользователя (I1, I2, I3).

• E2 — поле ввода пароля (I1, I2, I3, I4, I5).

• E3 — кнопка отправки формы с данными (I1, I2, I3, I4, I5, I6).

Предлагаемый метод позволяет создавать задачи автоматически и даже назначать их в соответствии с типом. Таким образом, по перечню функций будут созданы и назначены задачи для программистов, а по перечню интерфейсов и их элементов — задачи для дизайнеров.

Календарно-сетевое планирование. В теории календарно-сетевого планирования и управления важнейшими компонентами моделируемой системы являются работы (операции), события и ресурсы (разработчики). Сама система представляется в виде ориентированного графа без контуров. Дуги в этом графе соответствуют работам, а вершины — событиям начала и окончания работ (одной или нескольких параллельных).

Полученная ранее модель представления задач и ресурсов полностью соотвествует модели календарно-сетевого планирования и обладает дополнительной информацией, позволяющей автоматизировать формирование графа работ и событий. Кроме того, эта информация может быть использована и в качестве дополнительных ограничений при проведении оптимизационных расчётов.

Продолжительность проекта определяется суммарной продолжительностью работ, находящихся на критическом пути, или продолжительностью работ, которые не могут быть вы-

полнены только поочерёдно. В зависимости от того, с какой точностью можно определить длительность задач, возможны следующие ситуации:

• Продолжительность задач определена точно, при этом для оптимизации достаточно наиболее эффективно распределить некритические задачи с учётом информации о разработчиках [5].

• Продолжительность задач определена интервально, при этом можно определить множество исходов. Критичность задач при интервальной продолжительности можно разделить по трём уровням:

— Критические, для которых интервальные отклонения начала и окончания равны нулю.

— Полукритические, для которых отклонение начала равно нулю (нет резервов до наступления события), а отклонение окончания положительно (есть резервы на дальнейшие за событием работы, однако в условиях неопределённости этих резервов может не оказаться).

— Некритические, для которых отклонение начала положительно (есть гарантированные резервы до наступления события).

• Известны вероятности распределения продолжительности операций. В данном случае, с использованием методов имитационного моделирования, например модифицированного метода Монте-Карло, можно определить множество наиболее вероятных исходов и провести дальнейший анализ исходя из этих множеств.

• Имеется нечёткая информация о продолжительности операций. Она может быть получена на базе эксперных оценок, имитационного моделирования, упомянутого ранее, или даже гибридных методов.

Обозначим сетевой граф как: (V, E), $|V| = m$.

Каждую операцию как: $(i; j)$, а её продолжительность как t_{ij}.

Нечёткую информацию определим как: $\mu_{\tilde{t}_{ij}}(t_{ij})$, где $\mu_{\tilde{t}_{ij}}(\cdot) : \Re^1_+ \to [0;1]$ — функция принадлежности нечёткой продолжительности операций $(i; j)$, $i, j \in V$.

Из функции принадлежности нечётких полных резервов $\mu_{\Delta\tilde{t}_i}(x)$ [4] получим степень принадлежности i-го события (где $i \in V$) критическому пути:

$\mu_i = \mu_{\Delta\tilde{t}_i}(0) \in [0;1]$, таким образом, можно выделить перечень операций, для которых $1 - \mu_i < \varepsilon_j \in [0;1]$, где ε_j — является одним из уровневых критериев критичности.

Отметим, что в общем случае, можно определить произвольное множество уровней критичности, но для соответствия интервальным методам, мы выделим только три значения ε:

– ε_{++} для критических событий,
– ε_{+-} для полукритических событий,
– ε_{--} для некритических событий.

Для определения непосредственных значений ε в простом случае можно использовать экспертные оценки, однако наиболее качественные результаты вероятнее всего получить проведя анализ уже существующих историй проектов. Значения могут варьироваться в зависимости от размера проекта, количества разработчиков, типа задач и других факторов, поэтому значения необходимо подбирать используя механизмы статистического анализа.

Генетическое планирование. Все упомянутые выше ситуации позволяют получить перечень возможных распределений работсразличной степенью критичности. Тем не менее, данные подходы требуют наличия заранее определённого порядка выполняемых работ. Отчасти этот порядок фиксируется ролями исполнителей и становится известным после распределения работ по ресурсам, отчасти только после ручного определения.

В крупных проектах или в проектах, в которых необходимо произвести сложное планирование (например, допустимо увеличение критического пути, при сохранении эффективной нагрузки на всех участников в течение недели), ручное определение порядка работ является нецелесообразным и даже невозможным. В некоторых случаях, заранее невозможно даже определить подробный перечень работ по исполнителям.

Генетический алгоритим позволяет подобрать множество возможных графов работ, с оптимальными критическими путями и заранее заданными параметрами оптимизации эффективно перебирая возможные варианты. В данном случае используется модификация генетического алгоритма с динамическим размером генотипа, изначально адаптируемым под ограничивающие условия, заданные вручную.

В генотип системы входят:

• Матрица переходов, определяющая возможные задачи, работы и события. Часть из комбинаций фиксируется заранее, таким обра-

зом определяются части критического пути, порядок следования задач на которых не может быть изменён.

• Список участников проекта (максимальное и минимальное число участников могут быть заданы заранее).

• Список ролей (участники могут быть частично закреплены за ролями заранее).

• Список компонентов получаемых на выходе.

• Длительности работ или их оценки (интервальные, вероятностные или неточные).

Для селекции используются уже упомянутые ранее критерии (допускается использование и других параметров):

• Длина критического пути.

• Суммарные трудозатраты.

• Равномерность нагрузки на разработчиков.

• Фиксированные на критическом пути события.

Для каждого поколения осуществляются расчёты, в зависимости от типа неопределённости продолжительности. Расчёты порождают сразу множество возможных решений. В отличие от стандартного генетического алгоритма отбор производится именно с этим множеством результатов, а не с первоначальным поколением. После проведения отбора выделяются наиболее подходящие варианты и из них производится генерация нового поколения.Для этого, среди всех решений случайно выбираются пары, производится половинный обмен генома, среди соответствующих генов, и добавляется незначительная мутация. Заметим, что каждая группа генов подвержена мутации в различной степени, кроме того она увеличивается в зависимости от степени похожести двух базовых групп генов. Непосредственные величины, определяющие мутации могут быть подобраны как статистически, так и импирически, в зависимости от конкретных задач.

На протяжении всего процесса отдельно ведётся список наиболее удачных «особей», которые потом и выводятся в качестве наиболее оптимальных вариантов ведения проекта с указанием параметров. Ранжируя полученные варианты по части селекционных параметров можно получить группы решений, наиболее подходящих под те или иные параметры, сохраняя достаточную подходимость другим.

Выводы. Данный метод позволяет осуществлять раннее проектирование логики приложения, независимо от используемых при разработке технологий, что может существенно улучшить качество разарабываемой системы как на ранних стадиях. За счёт минимизации количества контекстов и ролей можно на ранних стадиях определить набор необходимых интерфейсов и избежать дублирования кода.

С другой стороны метод может быть применён к уже существующим системам или даже семействам систем с целью определения их похожести. В отличие от существующих методов, опирающихся только на абстрактное понятие «функция», метод позволяет сравнивать системы на более детальном уровне, с учётом компонентов и ролей пользователей.

Использование методов имитационного моделирования позволяет получить оценки трудозатрат с заданной точностью, а последующее использование этих результатов в алгоритме генетического планирования позволяет не только оптимизировать существующий календарный план, но и автоматически генерировать несколько оптимальных вариантов и вести планирование проекта в процессе его течения, выявляя отклонения и подбирая способы выхода из сложившейся ситуации.

СПИСОК ЛИТЕРАТУРЫ

1. Манифест и принципы гибкой разработки) — (http://www.kv.by/archive/index2008354201.htm).

2. Microsoft Solutions Framework— (http://ru.wikipedia.org/wiki/Microsoft_Solutions_Framework).

3. Адаптированный Microsoft Solution Framework — (http://www.microsoftproject.ru/articles.phtml?aid=69)

4. *Акимов В.А., Балашов В.Г., Заложнев А.Ю.* Метод нечёткого критического пути. — (http://www.masters.donntu.edu.ua/2013/fknt/drykin/library/method_nech_cr_p.pdf)

5. Управление проектами: справочное пособие / Под ред. И.И. Мазура, В.Д. Шапиро. – М.: Высшая школа, 2001. – 875 с.

6. *Гладков Л.А.* Генетические алгоритмы: Учебное пособие. / Л.А. Гладков, В.В. Курейчик, В. М. Курейчик. — 2-е изд. — М: Физматлит, 2006. — С. 320.

7. *Рутковская Д.* Нейронные сети генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский. — 2-е изд. — М: Горячая линия-Телеком, 2008. — С. 452.

8. *Артемов М.А.* Универсальные принципы проектирования пользовательских интерфейсов / М. А. Артемов, А. А. Чиченин. — Вестн. Воронеж. гос. ун-та. Сер. Системный анализ и информационные технологии.— Воронеж, 2011. — № 2. С. 115 – 119.

Артемов Михаил Анатольевич – заведующий кафедрой математического обеспечения и администрирования информационных систем Воронежского государственного университета, доктор физико-математических наук, профессор. E-mail: atremov_m_a@mail.ru

Беляев Роман Владимирович – Военно-воздушная академия имени профессора Н.Е. Жуковского и Ю.А. Гагарина. 9 960 100 81 61

Чиченин Александр Александрович – аспирант кафедры программного обеспечения и администрирования информационных систем факультета прикладной математики информатики и механики Воронежского государственного университета. E-mail: achichenin@dataart.com

Artemov Mikhail A. – Head of Department Software & Information System Administering, Voronezh State University, doctor of Physics-math. Sciences, Professor. E-mail: atremov_m_a@mail.ru

Belyaeb R. V. – Prof. Zhukovski and Gagarin Air Force Academy.

Chichenin Alexander A. – Post-Graduate Student, Department Software & Information System Administering, Voronezh State University. E-mail: achichenin@dataart.com