УДК 004.054

РАЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ КОМПОНЕНТОВ РАЗВИВАЮЩИХСЯ ПРОГРАММНЫХ СИСТЕМ НА ОСНОВЕ УЧЕТА ИХ ДИНАМИЧЕСКИХ СВОЙСТВ

© 2018 Н. А. Рындин, С. В. Сапегин

Воронежский государственный технический университет (г. Воронеж, Россия)
Воронежский государственный университет (г. Воронеж, Россия)

В статье рассматриваются вопросы оптимального проектирования компостов программных систем на основе учета их динамических свойств, т.е. учета временных факторов развития и модернизации этих компонентов при развитии своих программных систем. В основу положен итеративный подход к разработке программных систем. Выделяются основные базовые составляющие стратегии создания таких систем, заключающейся в достижении максимального экономического эффекта от отдельно взятого компонента системы (сервиса). Для оценки качества архитектуры системы вводятся ряд макропоказателей. Определяется жизненный цикл компонента системы и граничные условия его окончания. Приводится обобщенный алгоритм построения развивающейся программной системы и ее компонентов.

Ключевые слова: развивающиеся программные системы, жизненный цикл компонента, итеративный подход.

При разработке программных систем часто возникают ситуации, когда потребности пользователей и требования изначально определены недостаточно четко, либо сложность разработки настолько высока, что сопряжена со значительным риском. Именно для таких случаев изначально различными исследователями был разработан итеративный подход, заключающийся в последовательной разработке программной системы на основе имеющихся представлений с последующей апробацией и уточнением (или даже переформулированием) требований. Дальнейшее использование итеративного подхода для разработки проектов в различных предметных областях доказали его эффективность в тех случаях, когда цель разработки изначально плохо сформулирована, а проект проходит через большое количество изменений в требованиях.

Итеративный подход к разработке накладывает на проектирование структур программных систем весьма жесткие ограничения. Поскольку, изначальные оценки параметров проекта могут быть весьма неточными, количество итераций в процессе разработки может серьезно варьироваться. В свою очередь, это приводит к тому, что соображения гибкости и восприимчивости к изменениям в проектировании структуры программной системы становятся определяющими. Таким образом, рациональность разработки программных систем все меньше зависит от их общей структуры (которая должна быть максимально гибкой и простой), и все больше – от характера составляющих ее компонентов, каждый из которых обладает своими техническими и пользовательскими характеристиками, а также сроком службы и качеством предоставляемых информационных сервисов. Поэтому, в данных условиях немаловажное значение приобретают подходы, связанные с написанием конкретных компонентов ПО, их сопровождением и изменением, языками и инструментальными средствами, используемыми для разработки, а также с методиками рационального (с точки зрения обеспечения взаимодействия и архитектурной гибкости, а также минимизации затрат на разработку и изменение) выделения компонентов и их быстрой реализации [1].

Стратегия, реализуемая в рамках процесса разработки компонентов корпоративной ИС представляет собой решение задачи достижения максимального экономического эффекта от использования отдельно взятого компонента (сервиса). Формализацию этой

Рындин Никита Александрович – Воронежский государственный технический университет, к. т. н, доцент, hrimfaxi@icloud.com.
Сапегин Сергей Владимирович – Воронежский государственный университет, к. т. н., доцент.

задачи предлагается осуществлять в виде выражения

$$\int_{T_0}^{T_0+T} f(S(t); F(t)) \to max\ , \qquad (1)$$

где $S(t)$ – набор требований к компоненту ИС, $F(t)$ – реализованная функциональность, T – время реального использования компонента в системе, T_0 – момент начала использования компонента, f – функция, оценивающая соответствие функциональности компонента $F(t)$актуальным требованиям $S(t)$, $f \in [0,1]$. При этом, стратегии достижения результата могут комбинироваться из следующих базовых составляющих:

1.Формирование набора требований к ПО $S(t)$ наиболее удобным для реализации способом. Данная задача решается в рамках процессов «Бизнес-моделирование», «Управление требованиями», «Формирование технического задания», которые присутствуют в большинстве современных методик программной инженерии. С точки зрения отношений заказчика и исполнителя, способы формирования наиболее удобного для реализации набора требований можно разбить на следующие группы:

- применение комплекса методов и средств, направленных на структуризацию выдвинутых требований и предотвращение их неконтролируемой «эволюции» в рамках проекта (итеративное уточнение требований, прототипирование, сбор неформальных знаний о структуре проекта, проработка механизма изменения требований, разработка документов, фиксирующих требования на каком-либо уровне);
- облегчение формирования стратегии реализации (упорядочивание набора требований по степени важности, по легкости реализации, по наличию пользовательских зависимостей и т. д.);
- формирование встречных предложений (выяснение возможностей реализации некоторых требований уже существующими у разработчика компонентами и решениями, предложения по реструктуризации неключевых бизнес-процессов в сторону облегчения автоматизации).

2.Максимизация реализованной в рамках рассматриваемого компонента функциональности $F(t)$ к началу временного интервала использования разработанного компонента. Реализация функциональности происходит с учетом актуального набора требований $S(t)$ и прогноза насчет возможности его изменения, а также с учетом имеющихся технологий разработки и навыков команды разработчиков. Задача создания компонента с набором функциональности $F(t)$, максимально близким текущему набору требований $S(t)$ фактически является задачей организации эффективной командной разработки ПО, и широко рассмотрена в литературе за последние несколько десятилетий. Следует отметить, что большинство предлагаемых стратегий решения этой задачи базируются на эмпирических данных. Разработка и использование математических моделей для решения задачи организации разработки ПО в общем случае вызывает трудности, обусловленные следующими причинами:

- высокая значимость и непредсказуемость человеческого фактора;
- неэффективность формальных подходов к организации работы небольших команд;
- сильная зависимость результата от появляющихся технологий и методологий разработки.

3. Максимизация отрезка времени $[T_0, T_0 + T]$, в течение которого происходит использование разработанного компонента. Время использования компонента зависит от следующих факторов:

- проработанность требований, уменьшающая изменчивость набора S(t) и, как следствие, увеличивающая интервал времени T, в течение которого осуществляется использование компонента;
- использование в процессе разработки компонента методик, предусматривающих возможность внесения изменений (объектно-ориентированный подход, гибкая архитектура и т. п.);
- заранее определенная стратегия сопровождения и доработки компонента в процессе его использования (управление пользовательскими требованиями, документирование решений и технологий, учет человеческого фактора).

Для оценки качества архитектуры системы в целом можно использовать набор следующих макросистемных показателей:

- степень разнообразия технологий и методик в системе;
- степень интеграции компонентов системы;

- степень избыточности функциональности;

- степень несогласованности функций системы.

1.В качестве показателя, характеризующего степень разнообразия технологий и методик реализации компонентов корпоративной ИС можно использовать энтропию на множестве вариантов решений о разработке каждого компонента [2]. Системы с низкой энтропией вариантов решений характеризуются высокой «монолитностью», относительно жестко заданным набором используемых технологий. С одной стороны, это способствует более упорядоченной и непротиворечивой структуре корпоративной ИС, с другой стороны – увеличивает риск несоответствия компонентов пользовательским требованиям, что может привести к существенному сокращению срока использования системы в целом. Наоборот, высокая энтропия может как свидетельствовать о хорошо спроектированной и открытой для изменения архитектуре, так и сигнализировать о несвязности компонентов системы, отсутствии единых стандартов и, как следствие, об экспоненциальном увеличении стоимости разработки каждого последующего компонента.

2.Измерение степени интегрированности компонентов системы может производиться путем использования следующих показателей:

- среднее количество связей между компонентами системы:

$$C_{cp} = \frac{\sum_{i=1}^{n} C_i}{n}, \quad (2)$$

где C_i – число связей -го компонента с другими;

- среднее количество компонентов, участвующих в реализации бизнес-процесса:

$$B_{cp} = \frac{\sum_{j=1}^{m} B_j}{m}, \quad (3)$$

где B_j – число компонентов системы, задействованных в -м бизнес-процессе;

Помимо вычисления средних величин иногда бывает полезно оценить динамику распределения величин C_i и B_j , с целью выявления узких мест системы, определения целей рефакторинга ИС и реинжиниринга бизнес-процессов.

3. Степень избыточности функциональности – это показатель, характеризующий относительное количество неиспользуемой функциональности в системе. В зависимости от архитектуры системы и целей оценки, можно выделить следующие параметры оценки:

- доля интерфейсов, не используемых для реализации бизнес-процессов (для архитектур соответствующего типа);

- доля процедур и функций, не используемых в процессе работы ИС (включая «временно» отключенные в процессе финальной настройки функциональности);

- доля процессов, для которых существует реализация в КИС, но которые продолжают выполняться без привлечения компонентов ИС.

4.Степень несогласованности бизнес-функций системы – показатель, характеризующий разнородность реализации различных бизнес-процессов системы, несогласованность данных, логики поведения компонентов системы. Пусть V_i - i-й выявленный факт несоответствия между реализациями одного и того же действия в составе двух различных процессов системы. Определим $f(V_i)$ как метрику, определяющую степень значимости несоответствия -го факта V_i . Задачу интеграции и реинжиниринга ИС в таком случае можно формализовать в виде

$$\sum_{i=1}^{n} f(V_i) \to min, \quad (4)$$

где n – общее количество выявленных несоответствий в системе. В благоприятных условиях необходимо стремиться к полному устранению выявленных несоответствий, однако при интеграции различных приложений в корпоративных ИС бывают ситуации, в которых полное устранение несоответствий значительно превышает бюджет проекта, либо просто нецелесообразно по экономическим мотивам. В любом случае, принятие решения по каждой обнаруженной несогласованности, как и метрика $f(V_i)$ целиком зависит от мнения экспертов, занимающихся развитием архитектуры и интеграцией компонентов в рамках КИС.

В целом, задача оптимальной разработки и эффективной эксплуатации КИС сводится не только к проектированию статичной архитектуры, обладающей требуемыми характеристиками гибкости и производительности, но и к определению оптимальной траектории развития

корпоративной ИС, а также к удержанию корпоративной ИС на этой траектории [3].

Задачу разработки программного компонента в общем случае можно сформулировать следующим образом. Пусть есть набор требований к компоненту S_i, причем его состав можно представить в виде:

$$S_i = \big(1 + A(t)\big) \cdot S_{i.tech} + (1 + B(t) + C) \cdot S_{i.user} + S_{D(t)} \quad (5)$$

где $S_{i.tech}$ – технологии, используемые в работе компонента (включая технологии взаимодействия с другими компонентами системы); $S_{i.user}$ – пользовательские требования к компоненту; $A(t), B(t)$ – множители, характеризующие изменение требований к компоненту в течение его срока работы; C – составляющая согласования требований к компоненту между различными пользователями компонента корпоративной ИС; $S_{D(t)}$ – набор требований, определяющих процесс совмещения различной функциональности в компоненте (условие существования множителей $A(t), B(t)$. Таким образом, задачу о разработке компонента ПО в общем виде можно представить, как достижение за ограниченное время набора требований S_x, максимально близкого к некоторому идеальному набору требований S_{ideal}. В общем случае, достижение в процессе разработки самого набора требований S_{ideal} невозможно по следующим причинам:

1. Время разработки компонента ПО ограничено;

2. Набор требований к разрабатываемому компоненту ПО часто формируется не одним пользователем, а целой группой (или даже несколькими группами), причем каждый участник группы может выдвигать требования, плохо согласующиеся с требованиями остальных членов группы;

3. Требования к компоненту ПО как со стороны пользователей, так и со стороны взаимодействующего ПО, меняются с течением времени;

4. Анализ свойств процесса достижения пользовательских требований во многих случаях, как правило, не учитывает субъективного характера самого процесса, т. е. квалификации разработчиков, работающих над компонентом ПО.

Исходя из приведенной концепции, задачу достижения максимального обобщенного эффекта от использования отдельного программного компонента системы можно определить, как

$$\int_{T_0}^{T_0+T} f(S(t); F(t)) \to \max, \quad (6)$$

где $S(t)$ – набор требований, $F(t)$ – реализованная функциональность, T – время использования компонента, T_0 – момент начала использования, f — функция, оценивающая соответствие функциональности компонента $F(t)$ актуальным требованиям $S(t)$, $f \in [0,1]$. В качестве простейшего варианта функции f можно использовать выражение вида:

$$f = \frac{D(S(t),F(t))}{|S(t)|}, \quad (7)$$

где D – мощность симметричной разности множеств $S(t)$ и $F(t)$ в момент времени t, $|S(t)|$- мощность множества $S(t)$. Практика показывает, что зависимость расстояния между множествами $S(t)$ и $F(t)$ (т. е. степени соответствия функциональности компонента требованиям) в случае процесса интенсивной (целенаправленной) разработки имеет -образный характер. Это обусловлено тем, что в начале цикла разработки ресурсы затрачиваются не столько на реализацию функций, сколько на выстраивание архитектуры программного средства. Соответственно, варианты функции f, более соответствующие статистическим данным, следует искать среди семейств -функций различной кривизны. Схематичный график функции f, иллюстрирующий наиболее распространенный жизненный цикл такого компонента, представлен на рисунке 1

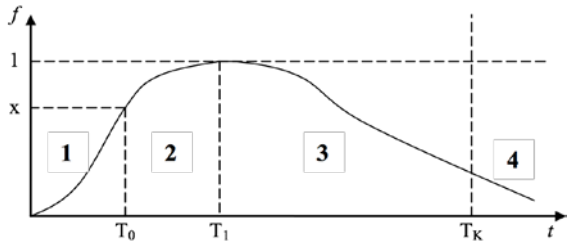

Рисунок 1. Жизненный цикл компонента.

Жизненный цикл компонента заканчивается решением о выводе его из эксплуатации. Основными причинами вывода программного компонента из эксплуатации являются:

1. Снижение уровня используемой функциональности до некоего критического порога (компонент становится фактически бесполезным в новых условиях);

2. Несоответствие компонента реалиям архитектуры (при смене операционной сис-

темы, модели данных, среды взаимодействия служб, стандартов и т. д.);

3. Замена компонента на новую версию, либо полное распределение функционала службы между другими, введенными в эксплуатацию, компонентами.

Следует заметить, что на стадии проектирования каждого конкретного компонента предсказать в точности его поведение достаточно сложно, так как характер и степень воздействия на него в процессе его развития могут оказаться совершенно непредсказуемыми. Однако, исходя из принципов подхода к анализу и проектированию отдельных компонентов на основе определения их жизненного цикла, а также исследования устойчивости траектории их развития, можно сформулировать общий алгоритм рационализации процесса разработки программных систем. При этом, задача построения систем в целом рассматривается, как комплекс задач разработки и модификации отдельных компонентов ПС, состоящих из следующих подзадач [4]:

1. Максимизация соответствия отдельных компонентов программной системы пользовательским требованиям в течение всего времени использования компонента в составе ПС.

2. Определение состава и количества подсистем ПС, группирующих компоненты, а также оптимальное распределение компонентов между ними.

3. Координирование жизненных циклов отдельных компонентов ПС с целью максимизации «покрытия» функциональностью системы пользовательских потребностей на протяжении всего времени существования ПС в целом.

4. Организация рационального взаимодействия компонентов системы для повышения способностей системы к автоматизации как стандартных, так и эксклюзивных пользовательских потребностей.

5. Реструктуризация ПС с целью устранения избыточной функциональности для повышения гибкости системы и улучшения ее эксплуатационных характеристик.

В общем виде процесс построения ПС, как системы компонентов, можно формализовать в виде последовательности действий, показанной на рисунке 2.

Рисунок 2. Алгоритм построения ПС.

В рамках первого этапа осуществляется первоначальная декомпозиция программной системы, разбиение общего набора требований на отдельные подсистемы, модули и библиотеки, обслуживающие различные потоки данных и задачи, отдельные подразделения и т. д. Результатом этого этапа является модель процессов автоматизируемого объекта, сгруппированных в пакеты и контуры по логическому признаку.

На этапе выделения информационных, функциональных и пользовательских зависимостей осуществляется разбиение подсистем требований на отдельные задачи, анализ информационных связей между службами, оптимизация их структуры.

Этап масштабирования подзадач связан с техническим анализом структуры программной системы, решением задач балансирования нагрузки между узлами разрабатываемого приложения, выбора технологии взаимодействия служб исходя из соображений развертывания системы, надежности и отказоустойчивости ее работы, среднего времени отклика на запрос и т. д. Особо следует отметить важность выбора способа обмена сообщениями между взаимодействующими сервисами в случае реализации распределенной системы. При этом, в общем виде способ обмена сообщениями будет являться одной из комбинаций вариантов асинхронного и синхронного взаимодействия.

Заключительным этапом процесса построения программной системы является

реализация ее отдельных компонентов и служб в соответствии с разработанной архитектурой, их тестирование и запуск в эксплуатацию.

Таким образом, на основе предложенной формализации был разработан алгоритм рационального с точки зрения сокращения трудоемкости и расходов проектирования программных компонентов, отличающийся учетом динамического характера как пользовательских потребностей, так и структуры разрабатываемых ПС, позволяющий осуществлять качественное и быстрое развитие программных комплексов.

ЛИТЕРАТУРА

1. Рындин, А. А Автоматизация проектирования корпоративных информационных систем на основе методов многовариантной интеграции / А. А. Рындин, С. В. Сапегин. – Воронеж, издательство ВГТУ, 2013. – 237 с.

2. Рындин, А. А. Особенности разработки развивающихся программных систем / А. А. Рындин, С. В. Сапегин // Вестник Воронежского государственного технического универсистета. – 2010. – Т. 10. – № 5. – С. 5-8.

3. Рындин, А. А. Многовариантная интеграция: теория и приложения в САПР: монография / А. А. Рындин. – Воронеж, издательство «Кварта». 2018. – 362 с.

4. Making a Formal Case for the Development of Components of Modern Enterprise Information System. Indian Journal of Science and Technology / A. Ryndin, S. Sapegin. С. 10-15.

RATIONAL DESIGN OF COMPONENTS DEVELOPING SOFTWARE SYSTEMS BASED ON ACCOUNTING THEIR DYNAMIC PROPERTIES

© 2018 N. Ryndin, S. Sapegin

Voronezh State Technical University (Voronezh, Russia)
Voronezh State University (Voronezh, Russia)

The article considers the issues of optimal design of composts of software systems based on consideration of their dynamic properties, i.e. taking into account the time factors of development and modernization of these components in the development of their software systems. The main basic components of the strategy of creating such systems are outlined, which consists in achieving the maximum economic effect from an individual component of the system (service). To assess the quality of the architecture of the system, a number of macroindicators are introduced. The life cycle of the system component and the boundary conditions for its termination are determined. A generalized algorithm for constructing an evolving software system and its components is given.

Key words: developing software systems, component life cycle, iterative approach.