УДК 681.518:658.512

ОБ ОДНОМ ПОДХОДЕ К ОЦЕНКЕ ЭФФЕКТИВНОСТИ ПРИМЕНЕНИЯ ПОСТ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ ТЕХНОЛОГИЙ ПРИ СОПРОВОЖДЕНИИ ПРОГРАММНЫХ СИСТЕМ

Н. Ткачук, К. Нагорный

Национальный технический университет «Харьковский политехнический институт»,
ул. Фрунзе 21,.Харьков, 61002, Украина
tka@kpi.kharkov.ua, k.nagornyi@gmail.com

Рассмотрена проблема реализации сквозной функциональности (СФ) при сопровождении объектно-ориентированных программных систем (ПС). Приведен сравнительный анализ некоторых особенностей пост объектно-ориентированных технологий (ПООТ) разработки ПС и предложен подход, позволяющий получить комплексную оценку эффективности применения ПООТ для решения проблемы реализации СФ.

The problem of crosscutting functionality (CF) implementation by object-oriented software systems (SWS) development is considered, the comparative analysis of some post object-oriented technologies (POOT) are preformed, and the approach is proposed, which provides a comprehensive assessment of effective POOT usage by CF implementation

1. Актуальность решения задач сквозной функциональности в объектно-ориентированных программных системах

В настоящее время большинство ПС проектируются и разрабатываются на основе объектно-ориентированной парадигмы, в соответствии с которой ПС представляется как некоторый набор объектов предметной области [1]. Эти объекты обладают определенным набором операций, реализующих логически завершенную часть функциональности (бизнес-логики) ПС и являются экземплярами соответствующих классов. ПС, разработанную таким образом, достаточно легко сопровождать, если вносимые в нее изменения в соотвествующих системных требованиях (СТ) касаются именно вышеуказанного типа ее бизнес-логики.

Однако практика разработки и сопровождения сложных компонентных программных решений (КПР) показывает, что даже в хорошо спроектированной ПС, может присутствовать бизнес-логика, реализация которой распределена по многим программным классам. Такую функциональность, которую в современной литературе принято называть *сквозной функциональностью* (crosscuting functionality – см., напр. в [2]), зачастую невозможно включить в рамки отдельного класса, поэтому необходимость её изменения доставляет разработчику ПС множество неудобств. Для более наглядного представления проблем, которые связаны с феноменом СФ, рассмотрим следующие примеры:

1) допустим, что сопровождаемая ПС “XYZ” написана на объектно-ориентированном языке Java и состоит из 225 классов. Для удобства администрирования этой ПС в 170 классах реализована функция записи критической информации в системный журнал, другими словами, должна быть обеспечена функция логирования системных событий. Эта задача может быть решена с помощью настраевоемого программного компонента log4j, путем вызова соответствующего метода класса системного журнала. Таким образом, везде, где осуществляется логирование событий, получаем повторяющийся участок кода, с вызовом соответствующего метода логирования. Очевидно, что внести изменения в подобный код довольно сложно, т.к. разработчику необходимо вручную просмотреть 170 классов, найти все необходимые вхождения логирования и произвести нужную замену кода;

2) далее, предположим, что в ту же сопровождаемую систему “XYZ” необходимо добавить новую функцию выбора данных по некоторому признаку (например, реализовать фильтрацию списка персонала компании по фамилии выбранного сотрудника). Фамилия этого сотрудника может быть задана глобальной переменной, по которой производится настройка всех функций ПС, таким образом, что выбирая любой пункт меню, пользователь, минуя основной поиск по сотрудникам, сразу получает страницу с персональными данными по данному сотруднику. Пусть, например, такая функциональность должна быть реализована в 137 классах. Разработчику, сопровождающему данную ПС, придется соответствующий участок кода дописывать в 137 классов, не говоря уже о возможных изменениях в коде самой функции фильтрации данных.

Таким образом, код такой ПС “XYZ” можно представить в виде рис. 1, где белые столбцы – это классы объектно-ориентированного проектирования (ООП), а серые полоски – это реализация кода СФ.

Рис. 1 показывает, что подобный программный код весьма сложен для сопровождения, так как основная функциональность классов ПС тесно переплетается со СФ, которая играет важную но все же второстепенную роль в бизнес-логике всей ПС. Подобная ситуация влечет за собой для разработчика следующие проблемы [3, 4].

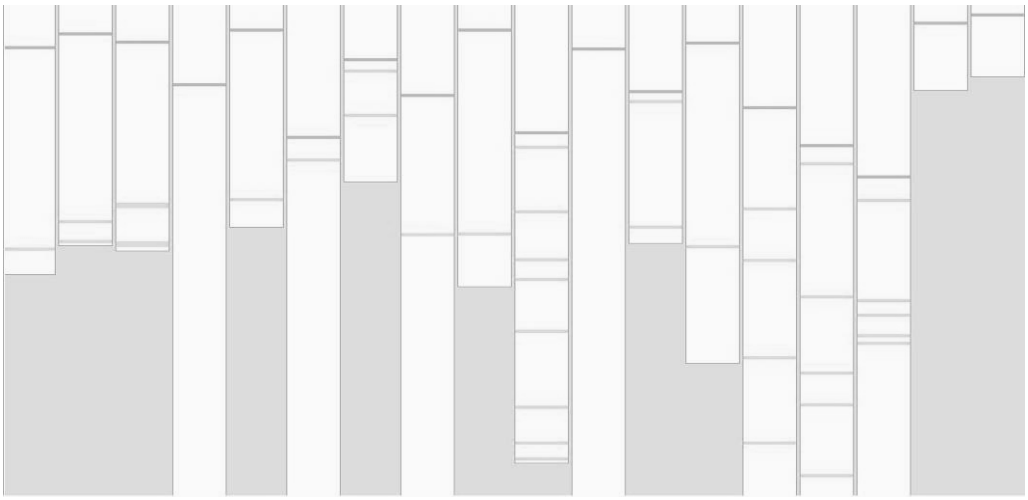

Рис. 1. Сквозная функциональность в ООП

1. Налицо избыточность программного кода;
2. Сложность внесения изменений в СФ, так как она распределена по всей ПС;
3. Невозможность повторного использования кода СФ из-за отсутствия модульности;
4. Затруднено понимание программного кода системы в целом.

Все эти проблемы приводят к снижению эффективности и без того достаточно трудоемкого процесса сопровождения сложных ПС.

Для количественной оценки *удельного веса СФ* в бизнес-логике некоторой ПС, можно, например, предложить следующий простейший показатель: отношение количества тех классов ПС, в которых присутствует СФ, к общему количеству ее классов, т.е. этот *коэффициент удельного веса* СФ (*Crosscutting Ratio* – CR) можно представить как

$$CR = \frac{C_{cf}}{C_{cf} + C}, \tag{1}$$

где C_{cf} – количество всех классов ПС, в которых присутствует СФ, C – количество всех классов ПС, в которых она не присутствует.

При этом значение $CR = 0$ говорит о том, что ПС совсем лишена СФ, а $CR = 1$ – о том, что в ПС все классы используют СФ.

Для вышеприведенных примеров $CR = \frac{170}{170 + 55} = 0,76$, и, соответственно, удельный вес СФ в системе XYZ довольно велик, это означает что средствами объектно-ориентированного программирования вносить изменения в подобную ПС будет довольно сложно.

Одним из способов решения подобной проблемы может быть использование *пост объектно-ориентированных* технологии (ПООТ) программирования, которые за последние 7–10 лет получили достаточно интенсивное развитие. Некоторые их особенности будут кратко рассмотрены далее.

2. Краткий обзор существующих ПООТ как средств реализации сквозной функциональности ПС

К наиболее известным на настоящий момент ПООТ относятся: аспектно-ориентированная технология (aspect-oriented technology), свойство-ориентированная технология (feature-oriented technology) и контекстно-ориентированная технология (context-oriented technology) – см., например, в [2, 5 ,6]. При этом, для анализа особенностей этих подходов при реализации СФ в некотором КПР по сравнению с объектно-ориентированным подходом (ООП), будем понимать под *базовой функциональностью* (БФ) этого КПР совокупность его методов, сгруппированных в ООП-классы и реализующих основную бизнес-логику.

1. Аспектно-ориентированный подход (АОП) – этот подход к разработке КПР предложен инженерами исследовательского центра Xerox PARC и на сегодняшний день реализован во многих языках программирования, таких как Java: AspectJ [7], C++, .NET, Python, JavaScript и др. АОП позволяет выделять СФ в отдельные модули, называемые *аспектами* (aspect), которые средствам определения *точек пересечения* (pointcut) с методами БФ и с помощью т.н. *инъекций* (injection) вводится в исходный код ПС. Схематично такое взаимодействие показано на рис. 2, а, где белыми вертикальными прямоугольниками C1, C2, C3 показаны ООП-классы, реализующие БФ, а серые ромбы A1, A2, A3 – аспекты, в которых сконцентрирована СФ.

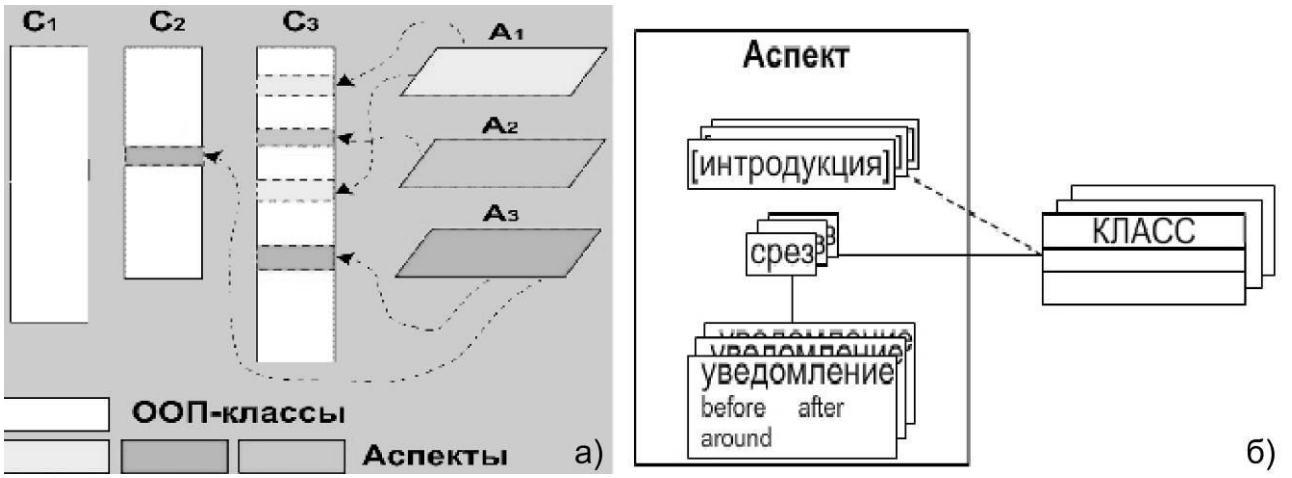

Рис. 2. Аспектно-ориентированный подход: а – концептуальная схема; б – технология реализации

Далее, специальный программный *компоновщик* (weaver) связывает основные методы, для которых в аспектах определены *точки соединения* (join-point) и производит инъекции методов СФ. Более подробная структура аспекта показана на рис. 2, б. Аспект состоит из *среза* (point-cut), *уведомления* (advice), и *интродукции* (inner declaration). Задача среза – определить точки соединения аспекта и методов БФ, другими словами, те строки кода в основных методах, куда должен будет введен код уведомления. Уведомление – это программный код на ООП-языке (например, на Java), реализующий СФ, который присоединяется к методам БФ. Уведомления бывают трех типов: *before* – уведомление выполняется до вызова основного метода; *after* – уведомление выполняется после вызова основного метода; *around* – уведомление выполняется вместо основного метода БФ. Также АОП позволяет внедрять в ООП-классы новые поля и методы, которые можно определять в аспектах, в результате чего объекты данного класса будут иметь поля и методы, определенные в соответствующем аспекте.

2. *Свойство-ориентированный подход* (СОП) – это подход к разработке КПР, который основан на алгебраической модели Gen Voka [2] и реализован на платформе Java в таких инструментальных средствах как: AHEAD Tool Suit [8], Feature IDE [9] и в некоторых др. СОП позволяет расширять классы БФ дополнительной функциональностью, сосредоточенной в *модулях расширения* (refines). Концептуально такое взаимодействие показано на рисунке 3, а, где белыми вертикальными прямоугольниками C_1, C_2, C_3 показаны ООП-классы, в которых реализована БФ, а серые горизонтальные прямоугольники – это СФ, которая группируется в логические *слои* (layr) обозначенные пунктиром: L_1, L_2, L_3. Технологический аспект реализации СОП (на примере платформы Java) приведен на рис. 3, б. Функциональное расширение содержится в jar-файле, где указывается, какой именно класс БФ должен быть ним дополнен (refines class). Далее, указывается метод класса БФ, который дополняется (refines method). Затем необходимо определить *конфигурационный файл* (equestion**),** где указываются все слои, которые должны войти в новую модификацию класса БФ. В итоге после компоновки из набора jak-файлов получается новая модификация ООП-класса. Таким образом, Feature IDE не является компилятором, а скорее используется как компоновщик новой модификации ООП-класса, который в последствии необходимо компилировать. Следует отметить, что в настоящее время ведется активная работа по объединению АОП и СОП подходов [2, 10].

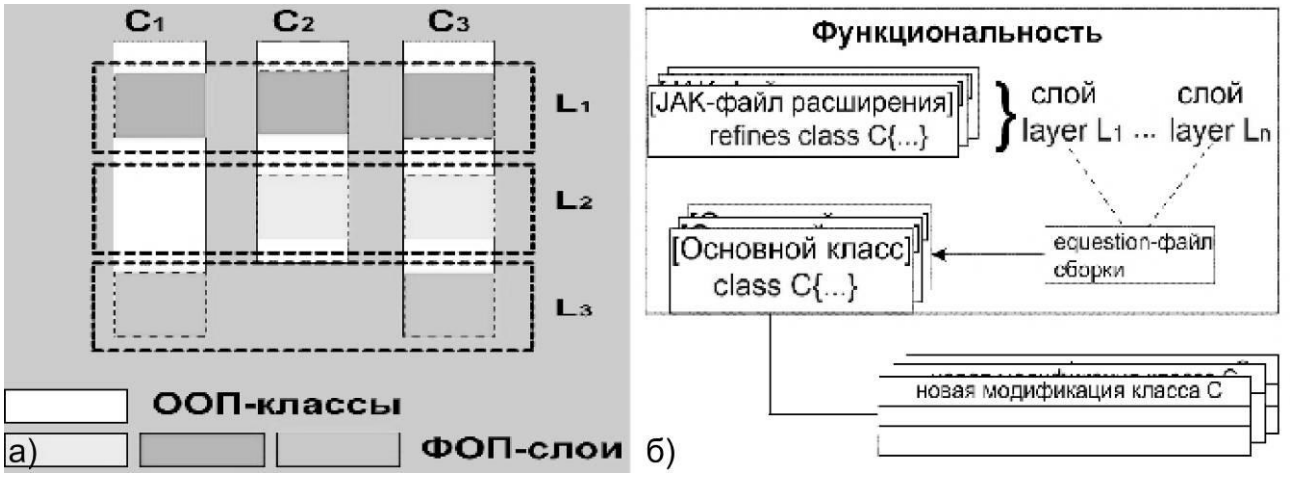

Рис. 3. Свойство-ориентированный подход: а – концептуальная схема; б – технология реализации

3. *Контекстно-ориентированный подход* (КОП) – подход к разработке КПР, позволяющий активировать/деактивировать классы СФ, в зависимости от некоторого контекста времени выполнения КПР. Такая СФ называется *вариациями поведения* (behavioral variations) КПР [11, 12] и известны следующие реализации КОП: ContextJ*, ContextJ для java; ContextL для Lisp, ContextS для SmallTalk [13].

Концептуально КОП-взаимодействие показано на рис. 4, а, где белыми вертикальными прямоугольниками C_1, C_2, C_3 показаны ООП-классы, в которых реализована БФ. Серые горизонтальные прямоугольники – это СФ, которая подобно СОП, группируется в логические слои, обозначенные пунктиром L_1, L_2, L_3. Вертикальным пунктиром K_1, K_2, K_3 показан контекст выполнения ООП-классов, светлыми кругами показана активация слоя, темными – деактивация.

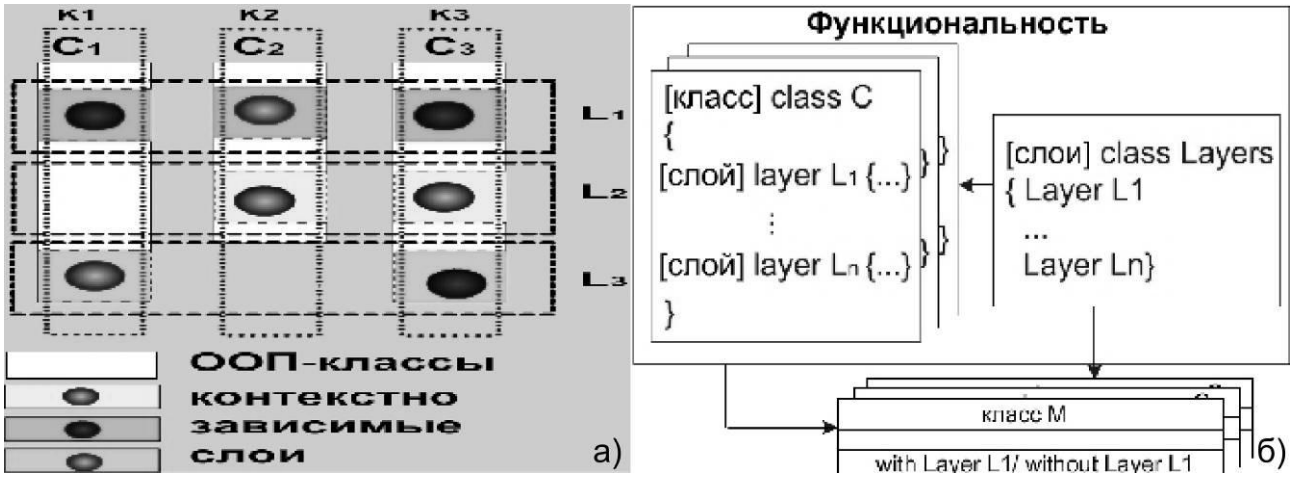

Рис. 4. Контекстно-ориентированный подход: а – концептуальная схема; б – технология реализации

Технология применения КОП для Java приведена на рис. 4, б. В отдельном файле объявляются все слои СФ, которые впоследствии включаются в класс БФ, используя зарезервированные слова *import layer*. Отличие КОП от вышеописанных подходов состоит в том, что СФ определяется по принципу *слой внутри класса* (layer in class). Каждый такой слой может расширять один метод соответствующего класса БФ. Кроме того, указывается слой, который вызывается по умолчанию, если никакие другие слои не активированы.

В табл. 1 приведен результат экспертной оценки вышерассмотренных ПООТ по нескольким их свойствам, при этом используются следующие значения этих оценок: «+» – данное свойство явно присутствует в соответствующей ПООТ; «–» – данное свойство отсутствует; «+/–» – свойство присуще данной ПООТ частично.

Таблица 1

Результаты сравнения отдельных ПООТ

Свойство технологии	ПООТ		
	АОП	СОП	КОП
Моделирование поведения ПС на более высоком уровне абстракции (по сравнению с ООП)	+	+	+
Простота сборки новых модификаций ПС	+	+	+
Реализация гомогенной СФ	+	+/–	+/–
Реализация гетерогенной СФ	+/–	+	+
Возможность внесения изменений в класс при отсутствии исходного кода	+	–	–
Реализация слоев СФ отдельно от модифицируемого класса БФ	+	+	–
Динамическое (контекстно-зависимое) подключение/отключение слоев	–	–	+
Возможность использования нескольких подходов одновременно	+/–	+/–	–
Наличие CASE-средств (текущая реализация)	+	+/–	+/–

Даже поверхностный анализ результатов этого сравнения показывает то, что для принятия решения о целесообразности и эффективности использования той или иной ПООТ для расширения функциональных возможностей существующей ПС, в частности, на этапе ее сопровождения, необходимо учитывать ряд других дополнительных факторов, которые и будут рассмотрены в предложенном подходе.

3. Подход к оценке эффективности применения ПООТ

В работе [14] была предложена концепция многомерного информационного метапространства проектирования и реинжиниринга ПС, которая, в частности, предполагает построение нескольких взаимосвязанных локальных подпространств, где происходит накопление и анализ данных о системных требованиях, моделей и методов проектирования ПС. Предлагаемый далее подход также использует эту концепцию.

3.1. Многомерная модель оценки эффективности применения ПООТ. В рамках этого подхода, в постановке задачи данного исследования, прежде всего, необходимо определить те показатели (или критерии), которые образуют соответствующие оси координат (или измерения) в многомерной модели оценки эффективности ПООТ. Принимая во внимание некоторые концептуальные и технологические особенности ПООТ, вышерассмотренные, а также с учетом предложенной в [15] информационной модели проектирования адаптивных ПС на основе ПООТ, к таким показателям целесообразно отнести следующие:

a) уровень *структурной сложности* сопровождаемой ПС;
b) показатель *динамики изменения* системных требований (СТ), которые должны быть учтены при сопровождении ПС;
c) коллекция рассматриваемых ПООТ, упорядоченная по *степени сложности* реализации архитектурных элементов ПООТ; (2)
d) оценка *эффективности применения* той или иной ПООТ при сопровождении унаследованной ПС, в качестве которой на данном этапе нашего исследования рассматривается коэффициент удельного веса СФ (см. выражение (1)).

Таким образом, задача разработки многомерной модели оценки эффективности применения ПООТ в процессах сопровождения ПС сводится к представления системы показателей (2) в некотором структурированном виде, который обеспечил бы возможность качественного представления и анализа их взаимосвязей, а затем – и в разработке экспертных процедур для получения количественных оценок эффективности применения той или иной ПООТ в зависимости от определенного типа структурной сложности ПС и характера динамики изменения СТ в процессах ее сопровождения.

3.2. Структурирование типов ПС. На первом этапе разработки многомерной модели оценки эффективности применения ПООТ проводится структуризация типов ПС с учетом показателей a) – b) из системы (2), а именно: “Уровень структурной сложности ПС” и “Динамика изменения СТ”. При этом предполагается, что этот последний показатель должен, в общем случае, учитывать как частоту внесения изменений в СТ, так и адекватно оценивать соответствующую трудоемкость реализации отдельных типов СТ при сопровождении ПС. На данном этапе исследования мы не ставим задачу предложить конкретный алгоритм для количественной оценки показателей a) – b), а лишь рассматриваем качественный подход к решению этой задачи, который показан на рис. 5. По оси ординат при этом откладывается определенный экспертным путем показатель динамики изменений СТ, а по оси абсцисс – показатель структурной сложности ПС, оба эти параметра в конечном итоге могут быть представлены в нормированном виде, т.е. их значения принадлежат интервалу $[0 \le 1]$. Тогда всю координатную площадь можно разбить на 4 квадранта, которые соответствуют следующим парам значений этих показателей:

I квадрант – «Структурная сложность ПС низкая» и «Динамика изменения СТ мала»;
II квадрант – « Структурная сложность ПС высокая» и « Динамика изменения СТ мала»;
III квадрант – «Структурная сложность ПС высокая» и «Динамика изменения СТ значительна»;
IV квадрант – «Структурная сложность ПС низкая» и «Динамика изменения СТ значительна».

Тогда с помощью такой системы координат каждую из ПС, которые должны быть модернизированы в процессе их сопровождения с применением той или иной ПООТ, можно отнести к соответствующей группе (I)-(IV), в рамках каждой из которых проведение последующего анализа показателей c), d) из выражения (2) будет корректным.

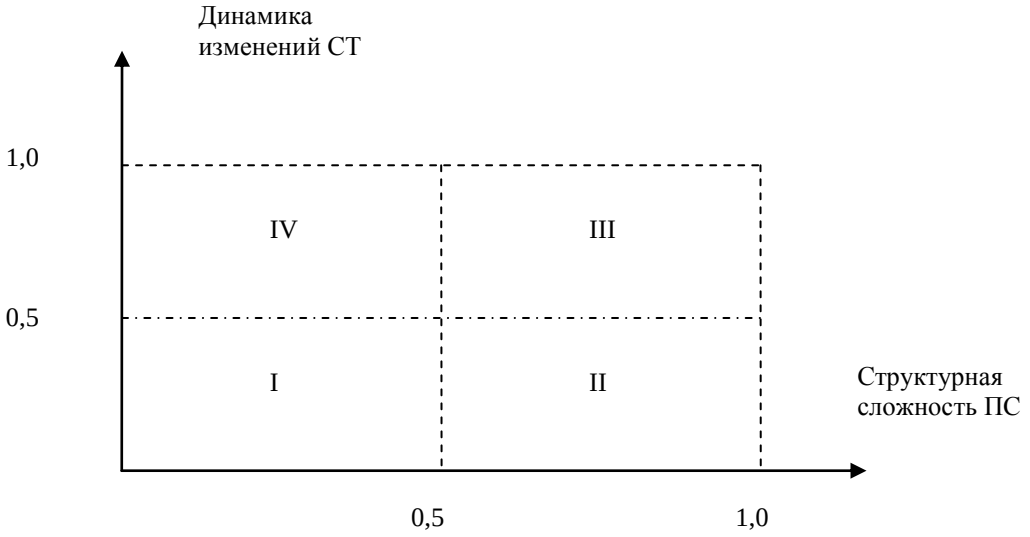

Рис. 5. Система координат для экспертной оценки типов ПС

Следует отметить, что для оценки сложности унаследованной ПС можно применить, например, такие методы как: метод функциональных точек (function points analysis – FPA), метод объектных баллов (object points analysis OPA) [16], множество различных количественных метрик сложности ПС [17] и др.

3.3. Оценка сложности реализации архитектурных примитивов на основе ПООТ. Для определения оценок (c) из перечня (2) предлагается проанализировать некоторые базовые архитектурные конструкции (в дальнейшем – архитектурные примитивы), которые могут быть получены с использованием различных ПООТ при дополнении их соответствующих объектно-ориентированных спецификаций. При этом очевидно, что для получения в дальнейшем корректной количественной оценки их необходимо представить в единой нотации. В качестве такой нотации нами предлагается выбрать один из языков описания программных архитектур (architecture description language – ADL), так как этот подход: 1) не зависит от конкретных средств программной реализации; 2) позволяет описывать как статическую структуру архитектурных примитивов, так и интерфейсы взаимодействия их элементов.

Основными абстракциями большинства ALD (см. напр., в [18–20]) являются компоненты, порты и коннекторы, а также такие дополнительные атрибуты ADL-спецификаций как роли, которые, в свою очередь, определяют тот или иной тип взаимодействия компонентов посредством коннекторов. Эти понятия определяются следующим образом:

- *компонент* (component) – сложный, потенциально состоящий из множества элементов, вычислительный блок или хранилище данных в программной системе (ПС). Компонент обладает интерфейсами – портами, для связи с его окружением;
- *порт* (port) – интерфейс взаимодействия компонента для связи с его окружением (другими компонентами ПС, внешней средой). Порт может предоставлять как простой интерфейс, например вызов одного метода, так и сложный, например, последовательные вызовы коллекции методов. В рамках данного исследования авторами предлагается расширить понятие порта:

–простой порт (Single Port) – интерфейс взаимодействия компонента для связи с его окружением. Например, метод класса.

–переключаемый порт (Case Port) – имеет возможность работы с разными коннекторами, например, метод класса может иметь дополнительный параметр булевого типа, значения которого указывают методу для какой среды его будут использовать.

- *Коннектор* (connector) – архитектурный блок, связывающий порты между разными компонентами. Коннектор моделирует взаимодействие между портами и правила, которые управляют этим взаимодействием с помощью ролей.
- *Роль* (role) – определение участников взаимодействия моделируемого коннектором.
- *Взаимодействие* (interaction) – свойство коннектора, которое определятся с помощью ролей. В рамках данного исследования авторами предлагается расширить понятие взаимодействия [17] следующим образом:

–*бинарное взаимодействие* (binary interaction) – у коннектора есть только 2 фиксированные роли;

–множественное взаимодействие (multiply interaction) – у коннектора имеется более чем 2 одновременно возможные роли. Например, т.н. *коннектор оповещения* (event broadcast connector) имеет 1 входную роль и произвольное число ролей-получателей;

–*переключаемое взаимодействие* (case interaction) – множественное взаимодействие, при котором у коннектора ролей более чем 2, однако одновременно возможные использование только 2 ролей, т. е. роли данного коннектора попарно взаимоисключающие.

В рамках определенных абстракций ADL под *архитектурным примитивом* ПООТ (АП ПООТ), будем понимать минимальную реализацию взаимодействия объектно-ориентированного подхода с ПООТ на уровне класса (см. схемы на рис. 2–4). Поэтому под компонентом понимается любая структурная единица ООП-класса и ПООТ-модуля: аспекта (*aspect*), функциональности (*feature*), контекста (*context*).

На рис. 5 показан архитектурный примитив, отображающий взаимодействие АОП с базовым ООП-классом. Белые крестообразные компоненты относятся к ООП-конструкциям, таким как: класс, поле, метод. Серыми прямоугольниками изображены основные компоненты АОП-конструкций: уведомления (*advice*) и интродукции (*inner declaration*).

Аналогичным образом могут быть построены и проанализированы 2 других архитектурных примитива для, соответственно, свойство-ориентированного и контекстно-ориентированного подходов (ввиду ограниченного объема данной статьи, они здесь не приведены).

Для расчета сложности представленных архитектурных примитивов ПООТ предлагается использовать формулы (3)–(6), которые, соответственно, имеют вид.

$$Component = 0.6 * \#POOT + 0.4 * \#OOP, \quad (3)$$

где $Component$ – коэффициент сложности компонентов архитектурного примитива, в которой учитывается количество компонентов БФ: $\#OOP$ и количество компонентов ПООТ: $\#POOT$, умноженных на соответствующие весовые коэффициенты, определенные экспертным путем.

$$Connector = 0.2*\#BinaryConnector + 0.3*\#MultyConnector + 0.5*\#CaseConnector, \quad (4)$$

где $Connector$ – коэффициент сложности коннекторов архитектурного примитива, в котором учитывается количество бинарных коннекторов: $\#BinaryConnector$, количество мультиконнекторов: $\#MultyConnector$ и количество кейс-коннекторов: $\#CaseConnector$, умноженных на соответствующие весовые коэффициенты, также определенные экспертным путем.

$$Port = 0.8*\#SinglePort + 0.7*\#CasePort, \quad (5)$$

где $Port$ – коэффициент сложности портов архитектурного примитива, в котором учитывается количество простых портов: $\#SinglePort$ и количество кейс-портов: $\#CasePort$, умноженных на соответствующие весовые коэффициенты.

$$Complexity = Component + Connector + Port, \quad (6)$$

где $Complexity$ – суммарный коэффициент сложности архитектурного примитива.

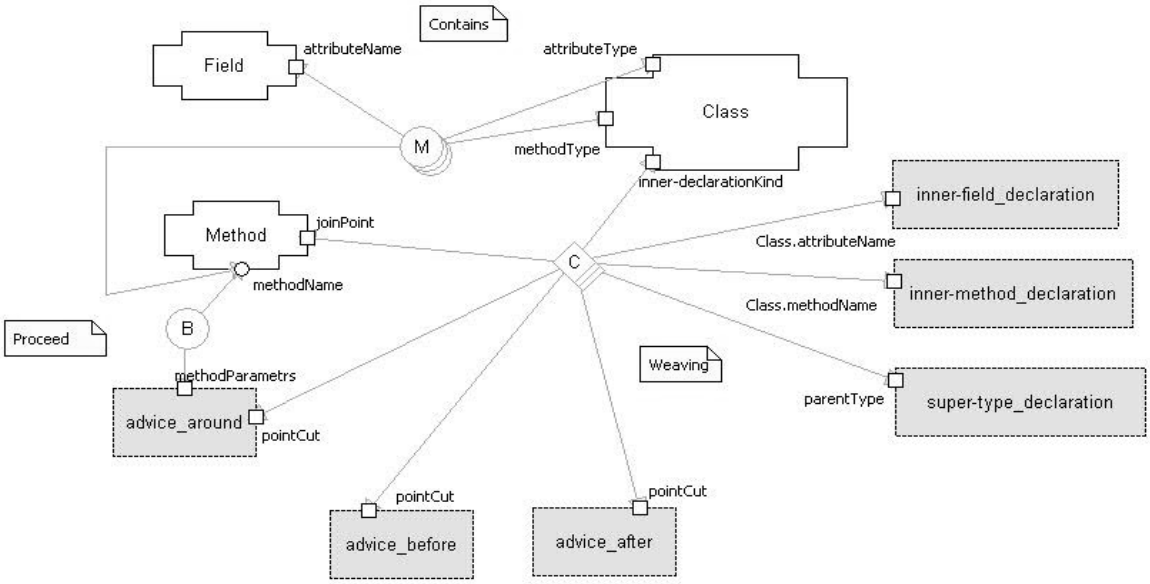

Рис. 5. ADL-спецификация аспектно-ориентированного архитектурного примитива

Результаты соответствующих расчетов сложности архитектурных примитивов, для различных ПООТ приведены в табл. 2.

Таблица 2

Результат расчета сложности архитектурных примитивов для вариантов АОП, СОП и КОП

Вид ПООТ	Коэффициент сложности компонента	Коэффициент сложности коннектора	Коэффициент сложности порта	Суммарный коэффициент сложности архитектурного примитива
АОП	4,8	1	4,3	10,1
СОП	3,6	1	3,9	8,5
КОП	2,8	0,7	4,1	7,6

На основе оценок сложности, представленных в табл. 2, можно сделать вывод об удельной трудоемкости применения отдельных ПООТ при реализации СФ в ООП-системах, что должно быть также учтено при определении комплексной эффективности использования ПООТ.

3.4. Определение комплексной эффективности использования ПООТ. Учитывая результаты, полученные в п.п. 3.2 – 3.3, а также принимая во внимание конечную цель построения многомерной модели оценки эффективности применения ПООТ (см. п. 3.1) можно предложить следующую ее графическую интерпретацию, которая позволяет определить механизм принятия решений при выборе той или иной ПООТ в процессах сопровождения различных типов унаследованных ПС и, в частности, при решении задач реализации в них сквозной функциональности (СФ). Этот подход показан на рис. 7 в виде 3-х мерного информационного пространства. Его структура представляется следующим образом:

- по оси абсцисс расположены типы ПС (см. их определение выше в п. 3.2), их позиции отвечают последовательности квадрантов I–IV (см. рис. 5);
- по оси ординат расположены пост объектно-ориентированные технологии, проранжированные в соответствии с суммарным коэффициентом сложности их архитектурных примитивов (см. выкладки в п. 3.3);
- по оси аппликат располагается оценка эффективности применения ПООТ, в контексте данной работы следует понимать *коэффициент удельного веса* СФ для того или иного типа ПС (см. раздел 1, выражение (1).

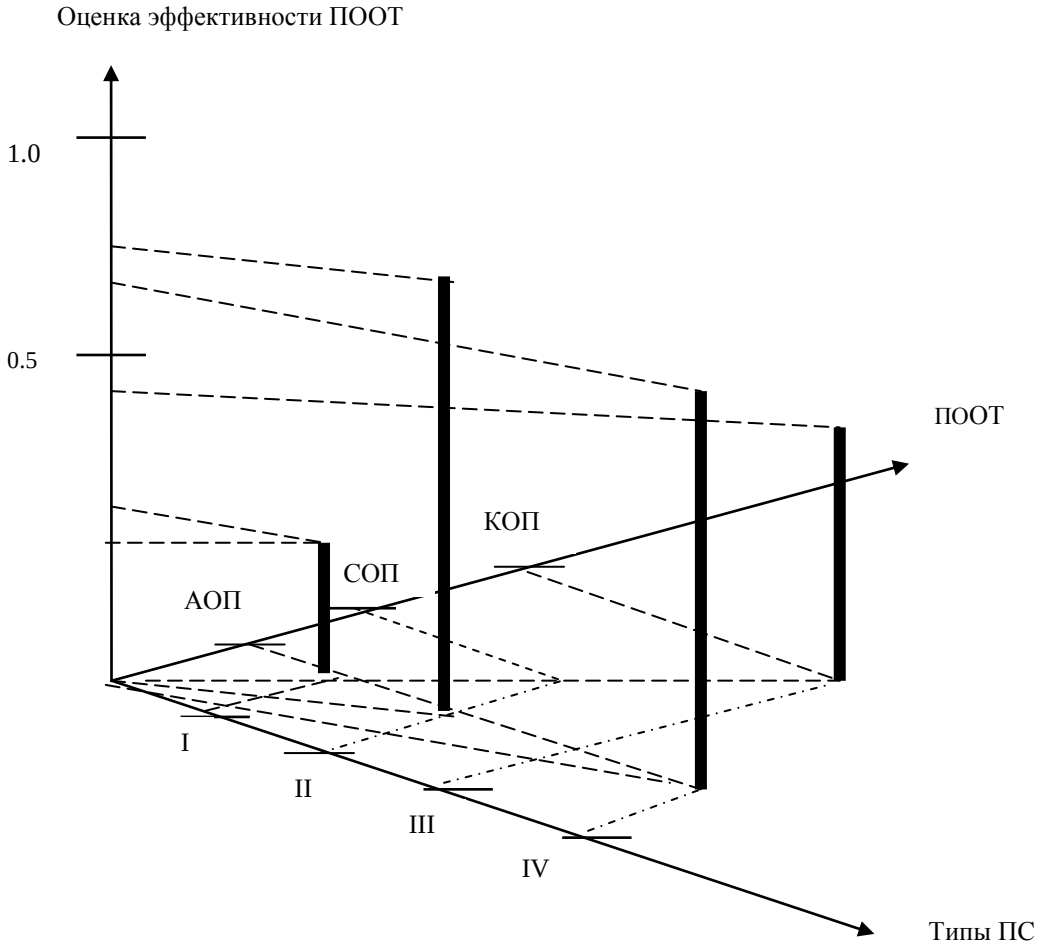

Рис. 7. 3-х мерное пространство для определения эффективности применения ПООТ

Следует отметить, что такая графическая интерпретация предложенного подхода позволяет не только наглядно представить результат применения различных аналитических расчетных методов и экспертных процедур, но и может быть использовано как прототип соответствующего визуального интерфейса для перспективного CASE-средства, которое должно быть разработано для автоматизации предложенного подхода.

4. Выводы и направления дальнейших исследований

В данной работе:

1) показана актуальность проведения исследований по проблемам определения эффективности применения различных пост объектно-ориентированных технологий (ПООТ) в процессе сопровождения программных систем;

2) предложен новый подход к решению этой задачи, который одновременно учитывает несколько факторов влияния и позволяет структурировать их взаимосвязи в соответствующем многомерном информационном пространстве;

3) определены основные этапы разработки комплексной методики, которая позволит в дальнейшем получить количественные оценки эффективности применения различных ПООТ с использованием знание-ориентированных экспертных методов обработки соответствующей информации.

Полученные результаты предполагается использовать для выбора эффективного варианта применения ПООТ в процессах модернизации программных систем различного назначения, реализованных, в частности, на платформе J2EE, которые разрабатываются и сопровождаются в настоящее время на кафедре НТУ "ХПИ" в рамках научно-технического сотрудничества с отечественными и зарубежными партнерами.

1. 1 *Соммервилл И.* Инженерия программного обеспечения: Пер. с англ. – М.: Издательский дом "Вильямс", 2002.
2. *Apel.S.* The Role of Features and Aspectsin Software Development. Dissertation, Otto-von-Guericke University Magdeburg, 2007 (in German).
3. *Schwanninger C., Wuchner E., Kircher M.* Encapsulating Crosscutting Concerns in System Software: Proc. of the Third AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software, Northeastern University, 2004.
4. *Hilsdale E., Hugunin J.* and others: Aspect-Oriented Programming with Aspect J: Palo Alto Research Center Incorporated, California 2002.
5. *Laddad R.* AspectJ in Action. Practical aspect-oriented programming: Manning Publications Co., 2003.
6. *Hirschfeld, R.; Costanza, P.; Nierstrasz, O.* Context-oriented Programming. In: "Journal of object technology", 2008. – №. 3. – P. 125–151.
7. http://www.eclipse.org/aspectj/ 02.2010
8. *Batory D.* Feature-Oriented Programming and the AHEAD Tool Suite: Proc. of the 26-th International Conference on Software Engineering, Texas USA, 2004
9. http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/ 02.2010
10. *Apel S., Leich T. Rosenmüller M., Saake G.* Combining Feature-Oriented and Aspect-Oriented Programming to Support Software Evolution. In: Proceedings of the 2nd ECOOP Workshop on Reflection, AOP and Meta-Data for Software Evolution (RAM-SE), School of Computer Science, University of Magdeburg, 2005. P. 3–16.
11. Hirschfeld, R., Costanza, P., Nierstrasz, O:Context-oriented Programming, In: Journal of Object Technology, 2008. – N 3/ – P. 125–151.
12. Appeltauer, M., Hirschfeld, R., Haupt, M,. Masuhara, H: ContextJ: Context-oriented Programming with Java – 2009.
13. http://soft.vub.ac.be/~pcostanz/contextj.html (просмотрено 03.02.2010).
14. Ткачук М.В. Моделі, методи та інформаційні технології адаптивної розробки і реінжинірингу інформаційно-управляючих систем // Автореф. дис. на здоб. вчен. ступеня д-ра техн. наук. – НТУ «ХПІ», Харків, 2006.
15. *Ткачук М.В., Нагорний К.А.* Про один підхід до структурної адаптації програмних систем на основі пост об'єктно-орієнтованих технологій // Тез. докл. IX Междунар. науч.-техн. конф. «Системный анализ и информационные технологии», НТУУ «КПИ», Киев, 26–30 мая 2009. – С. 577.
16. *Андон Ф.И., Коваль Г.И., Коротун Т.М.* и др. Основы инженерии качества программных систем. – 2-е изд. – К.: Академпериодика. – 2007.
17. *Демирский А.А.* Оценка структурной сложности программных средств в промышленности на ранних стадиях жизненного цикла // Автореф. дис. ... канд. техн. наук. – Тверской гос. техн. университет, Тверь, РФ. – 2009.
18. *David Garlan, Robert Monroe, David Wile.* ACME: An Architecture Description Interchange Language // Computer Science Department Carnegie Mellon University January 14, 1997.
19. *Basem Suleiman, Vladimir Tosic, Eldar Aliev*. Non-Functional Property Specifications for Wright ADL // School of Computer Science and Engineering, University of New South Wales, Australia 2008.
20. *Hang Tao1, Cao Yan-ping.* Measuring the complexity of product line architecture with vADL // School of Science, Xi'an University of Architecture and Technology, Xi'an 710055, China, Sep. 2008.