ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г.

Главный редактор
ГУРИЕВ М.А.

Редакционная коллегия:
АВДОШИН С.М.
АНТОНОВ Б.И.
БОСОВ А.В.
ВАСЕНИН В.А.
ГАВРИЛОВ А.В.
ДЗЕГЕЛЁНОК И.И.
ЖУКОВ И.Ю.
КОРНЕЕВ В.В.
КОСТЮХИН К.А.
ЛИПАЕВ В.В.
ЛОКАЕВ А.С.
МАХОРТОВ С.Д.
НАЗИРОВ Р.Р.
НЕЧАЕВ В.В.
НОВИКОВ Е.С.
НОРЕНКОВ И.П.
НУРМИНСКИЙ Е.А.
ПАВЛОВ В.Л.
ПАЛЬЧУНОВ Д.Е.
ПОЗИН Б.А.
РУСАКОВ С.Г.
РЯБОВ Г.Г.
СОРОКИН А.В.
ТЕРЕХОВ А.Н.
ТРУСОВ Б.Г.
ФИЛИМОНОВ Н.Б.
ШУНДЕЕВ А.С.
ЯЗОВ Ю.К.

Редакция:
ЛЫСЕНКО А.В.
ЧУГУНОВА А.В.

СОДЕРЖАНИЕ

ПАРТНЕР ВЫПУСКА IBM

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать"– **22765**, по Объединенному каталогу "Пресса России"– **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru E-mail: prin@novtex.ru

© **Издательство "Новые технологии", "Программная инженерия", 2010**

УДК 338.46; 004.9

М.А. Гуриев, д-р техн. наук, проф., IBM Восточная Европа Азия, главный редактор журнала "Программная инженерия"

E-mail: marat_ guriev@ru.ibm.com

Журнал "Программная инженерия" и наука о сервисах

Показан рост доли услуг в современной экономике. Обоснована целесообразность создания специальной "Науки о сервисах". Проиллюстрирована роль компании IBM в ее становлении. Сформулированы основные задачи журнала в связи с развитием новой науки.

Ключевые слова: *наука о сервисах, компания IBM, задачи журнала "Программная инженерия"*

Дважды в истории компания IBM обращалась к университетской общественности с инициативой внедрения новой дисциплины.

Первый раз примерно четверть века назад – это была известная и ставшая популярной "Наука о компьютерах" – *Computer Science*.

В 2004 г. известная компания представила академическим кругам проект новой дисциплины, которая называется "Наука о сервисах, управлении и инжиниринге" или в англоязычной версии "*Service Science Management and Engineering*". Используются также сокращения *Service Science* или SSME.

Для интереса к новой дисциплине имеются веские основания. Доля услуг в экономике неуклонно растет. В ВВП большинства развитых стран услуги составляют от 50 до 80 %. В то же время, в отличие от производства, инжиниринг сервисов до сих пор не является приоритетной областью исследований и подготовки кадров. Решению этих задач и был посвящен разработанный в исследовательском центре IBM – *Almaden Research Center* – курс "Науки об услугах, управлении и инжиниринге" (http://www.almaden.ibm.com).

Президент IBM Самуэль Палмизано (*Samuel Palmisano*) после подписания в ноябре 2006 г. меморандума о взаимопонимании между IBM и министерством образования Китая, предусматривающего внедрение программ новой дисциплины в более чем 50 ведущих университетах страны, так сформулировал задачи новой науки: "развитие навыков, опыта и бизнес-моделей, необходимых для глобальной экономики, в которой сектор услуг будет занимать лидирующие позиции".

Пол Хорн (*Paul Horn*), вице-президент и глава исследований IBM, рассматривает науку об услугах в качестве междисциплинарного направления, в котором наука, управление и инжиниринг взаимодействуют в целях повышения качества обслуживания. Развитие этой науки обеспечивает теоретические основы создания современной отрасли услуг, которая будет разрабатывать и внедрять технологические приложения, открывающие новые возможности для частных и государственных структур в решении стоящих перед ними задач.

Экономическая аргументация необходимости подобного подхода иллюстрируется статистическими данными, приведенными на рис. 1.

Легко заметить, что объем производимых услуг, начиная с некоторого времени, преобладает в экономике крупных ИТ компаний.

Надо отметить, что в контексте введения новой науки вовсе не противопоставляются услуги и производство. Наоборот, исследования, например, в области информационных технологий показывают, что именно производство техники и технологий является фундаментальной причиной, обеспечивающей качественный и количественный рост услуг. Включение в экономику информационных технологий, массово производимых автомашин, бытовой и другой техники неизбежно порождает необходимость поддержания процессов эффективной эксплуатации формируемой таким образом техносферы и, как следствие, образование новых и расширение действующих услуг.

Так, в современном мире резко возрастает число встраиваемых программируемых систем. Смартфоны, бытовая техника, "умные" дома, системы связи и навигации в транспорте, системы наблюдения и безопасности, и многие другие образуют постоянно расширяющуюся сервисную инфраструктуру с возможностями выхода и интеграции в глобальной сети.

Еще одной стороной данного явления представляется процесс перевода многих активов в услуги. К примеру, можно не приобретать автомобиль, а взять его напрокат, не ставить сложную вычислительную или графическую станцию, а заказать выполнение соответствующих видов работ в специализированных компаниях, и т.п.

Существует и другой – методический аспект необходимости объединения "под одной крышей" трех независимых научных дисциплин. IBM является организацией, создающей крупномасштабные системы информационной технологии. Успех проектирования подобных структур в значительной степени зависит от искусства разработчиков и их умения в процессе проектирования "сблизить" бизнес-процессы и инфраструктуру ИТ, обеспечивающую их реализацию, и сделать их взаимодействие более гибким и оперативным. Совершенствование методов проектирования ИТ систем и технологий в настоящее время ведется в направлении повышения их функциональности, возможности многократного использования, и реинжиниринга – способности изменять состав и конфигурацию бизнес-процессов и средств автоматизации в реальном времени, адаптируясь к изменению окружения системы.

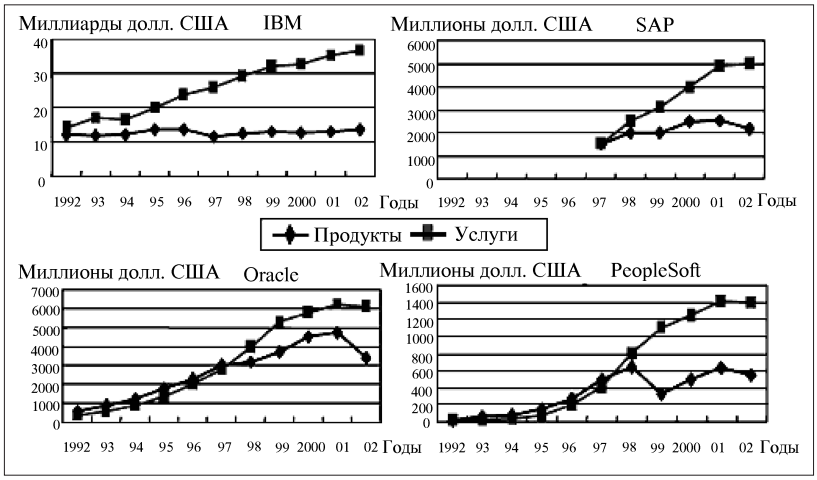

Рис. 1. Изменение соотношения между объемами продаж продуктов и услуг крупнейших производителей ИТ в течение 10 лет

Практическое осуществление данного направления оформилось в рамках сервис-ориентированной архитектуры (СОА), когда использование включаемого в систему программного обеспечения рассматривается в качестве сервиса. СОА, представляющая собой основу интеграционных процессов в Интернет поколения Web 2.0, одновременно является двигателем перевода коммерческого софта, на котором базируются ERP системы, в сервисы. А для интеграции сервисов различного назначения – функциональных, инфраструктурных, управленческих – требуются новые подходы к инжинирингу и управлению такими системами.

Основы курса отражают комплексный, междисциплинарный характер новой дисциплины, образованной на пересечении науки, бизнеса, технологий, социальной сферы, управления и экономики.

Дисциплина строится на принципах системного подхода, другими словами, основными объектами исследований являются системы обслуживания. К числу последних могут быть отнесены практически любые социосистемы: университеты, больницы, центры обработки заказов (кол-центры), центры хранения данных, семьи, города и даже государства в целом.

Дисциплина, созданная в исследовательском центре Альмаден, состоит из ряда модулей, перечисленных ниже.

• **Модуль 1.** Вступительная часть курса (введение в науку об услугах, отдельные примеры использования научных подходов применительно к сервисам, методология исследований в науке об услугах).

• **Модуль 2.** Сервисы (сервис-доминантный подход, управление качеством обслуживания, моделирование процессов предоставления услуг, инновационные услуги, промышленный анализ технологических услуг, цифровое управление бизнесом, управление знаниями).

• **Модуль 3.** Сервисные системы (сервис-ориентированная архитектура, технология web-сервисов, извлечение данных и поиск сервисов, технология обработки документов, управление проектами).

• **Модуль 4.** Управление обслуживанием.

• **Модуль 5.** Продуктивность и инновации в процессе обслуживания.

• **Модуль 6.** Введение в методологию науки об услугах.

• **Модуль 7.** Вызовы, культура обслуживания и социальные системы (организационная социология, человеческий фактор в процессах обслуживания, промышленная экономика, психология потребления и этика бизнеса, законодательные аспекты сервиса).

Разработчики новой дисциплины отдавали себе отчет, что сформированная структура курса не исчерпывает всех мыслимых направлений, и в рамках каждой из многочисленных приложений возможна своя структура, адекватно отображающая ее специфику. Возможности здесь многогранны, что отображается на рис. 2, в котором использована популярная аллегория слепых, исследующих на ощупь слона.

Приведенный рисунок отображает не только множество вариантов построения курса, но и показывает способ, как это может быть достигнуто. В правой части рисунка отображены специализации, связанные с предоставлением услуг. В левой – названия отдельных научных дисциплин, также сопряженных с данной проблемной областью. Комбинируя первые и вторые, можно получить множество прикладных курсов, например "Предоставление услуг контроля качества", и т.д.

За истекший срок новое направление получило широкое распространение в ведущих университетах мира. Некоторые из них заняли лидирующие позиции в исследованиях и привнесли инновационное видение новой дисциплины. К таким университетам относятся: Беркли, Карнеги–Мелонн, Технологический Институт штата Джорджия, Массачусетский технологический институт (США), Институт Фраунгофера, Университет Карлштада (ФРГ), университет Цинг Хуа (Тайвань), университет Кента (Великобритания).

Рис. 2. Аллегория науки о сервисах

Серьезное развитие преподавание новой науки получило в Чехии и Венгрии.

Некоторые из перечисленных вузов размещают на своих сайтах материалы, связанные с преподаванием дисциплины. Так разработки университета Беркли доступны по адресу: http://rosetta.sims.berkeley.edu:8085/sylvia/f06/view/202.complete

В Российской Федерации имеются не менее серьезные предпосылки развития рассматриваемой дисциплины:

- Доля сферы услуг в 1990 г. в ВВП РСФСР равнялась 35 %. К 1999 г. в ВВП России она выросла до 55 %, и на данный момент колеблется на уровне 65 %.
- Ведутся преподавания по специальности 080506 "Логистика и управление цепями поставок", имеющей непосредственное отношение к дисциплине. ГОС введен в 2006 г., действуют кафедры во многих вузах.
- Наличие образовательных учреждений, обучающих студентов предоставлению услуг (Российский государственный университет туризма и сервиса, Владивостокский государственный университет экономики и сервиса, Тольяттинский государственный университет сервиса), кафедры сервиса во многих вузах.
- Активное вынесение индустрии услуг в Интернет и развитие Web-сервисов.
- Невысокое качество услуг и необходимость повышения их качества.

Специфика каждого отдельного вуза и специальностей доминирует при разработке программ соответствующих курсов. На сегодняшний день во многих вузах читаются дисциплины "Проектирование сервисов", "Управление и инжиниринг сервисов", "Механизмы обслуживания и маркетинг", "Управление обслуживанием", "Измерение и моделирование услуг", "Источники предоставляемых услуг и стандартизация", "Маркетинг услуг и качество", "Интеллектуальная собственность и предоставление услуг". Сегодня в Интернете, набрав в качестве ключевого слова SSME, можно найти описания и других курсов, сопряженных с новой дисциплиной.

Применительно к области профессиональной деятельности журнала "Программная инженерия" это ставит некоторые новые задачи:

– исследование и описание технологий интегрированной разработки сложных систем, созданных на принципах архитектуры, основанной на модели (Model Driven Architecture), в которой создание программных продуктов тесно связано с проектированием информационных систем с использованием UML-2 и программных продуктов семейства IBM Rational и аналогов, поддерживаемых другими вендорами;

– построение технологий производства "облачных" сервисов;

– создание инструментов для быстрого реинжиниринга и изменения конфигурации бизнес-процессов и архитектуры IT-сервисов, класса "приборной доски" (*IT-services Dash board*);

– совершенствование механизма и языков хранения и поиска воспроизводимых сервисов, а также встраивания их в существующие системы обслуживания.

Перечисленные направления, конечно, не исчерпывают все открывающиеся в рамках новой науки возможности, и хочется надеяться, что авторы и читатели журнала будут их пополнять и внесут свой оригинальный вклад в ее развитие.

УДК 004.41

В.В. Липаев, д-р техн. наук, проф., гл. науч. сотр., Институт системного программирования РАН

E-mail: alexlip@mail.ru

Проблемы программной инженерии: стандартизация, человеческий фактор, подготовка кадров

Рассмотрены основные научные, методологические и технологические проблемы программной инженерии, которые возникают на различных этапах жизненного цикла современных сложных комплексов программ. Представлены проблемы стандартизации процессов в программной инженерии, влияния на них человеческого фактора, организации коллективов исполнителей работ и обучения специалистов. Выделены и сформулированы свыше двадцати наиболее актуальных частных задач, представлены пути и методы и их решения.

Ключевые слова: *программная инженерия, сложные комплексы программ, стандартизация, человеческий фактор, подготовка кадров*

1. Проблема стандартизации в программной инженерии

Задачи выбора стратегии обеспечения стандартами программной инженерии. В 1980-е годы стандартизацией процессов программной инженерии в США активно занялись в Институте программной инженерии – SEI, Институте инженеров по электротехнике и электронике – IEEE. Затем в решение вопросов их обобщения, корректировки, согласования с разными странами и утверждения стандартов включилась международная организация по стандартизации – ISO. Как признание важности этого направления с середины 1990-х годов [3] в России была принята **стратегия перевода на русский язык международных стандартов ISO, их адаптация и утверждение в качестве Государственных стандартов.**

Основные положения программной инженерии в 1990-е годы формализовались в наборе (около 50) систематизированных международных стандартов, охватывающих практически все процессы жизненного цикла сложных программных средств. Эти стандарты сконцентрировали и обобщили опыт, а также технологии разработки и сопровождения крупных программных продуктов высокого качества в различных отраслях народного хозяйства и оборонной техники. Несколько десятков таких стандартов допускают целенаправленный отбор необходимых унифицированных процессов в зависимости от характеристик и особенностей конкретного проекта, а также формирование на их базе проблемно-ориентированных профилей стандартов для определенных типов проектов и/или предприятий. В условиях, когда сложно организованные средства и системы автоматизации активно внедряются во все, включая критические для государства, сферы деятельности, продолжение работ на этом направлении является крайне важной задачей.

Задача освоения и применения международных стандартов программной инженерии. Практическое применение профилей стандартов, аккумулирующих мировой опыт создания различных типов крупных комплексов программ, способствует значительному повышению производительности труда специалистов и качества создаваемых ими программных продуктов. Эти стандарты определяют процессы модификации и возможности повторного применения программных компонентов и комплексов, их расширяемость и переносимость на различные аппаратно-программные платформы. Последнее обстоятельство непосредственно отражается на росте экономической эффективности технологий и процессов создания различных программных продуктов и систем. Для регламентирования процессов жизненного цикла такие профили

стандартов должны адаптироваться и конкретизироваться применительно к определенным классам и функциям выполняемых проектов, процессов и компонентов создаваемых в их рамках программных средств (ПС). При этом должна сохраняться концептуальная целостность применяемой совокупности стандартов и их эффективное, положительное влияние на процессы и результаты, качество, надежность и безопасность программных продуктов при реальных ограничениях на использование ресурсов проектов.

Значительное внимание в методологии программной инженерии уделено фрагментам и компонентам стандартов ISO, которые необходимы для того, чтобы обеспечить высокое качество и безопасность применения программных продуктов. Менеджмент программных проектов должен быть ориентирован на базовые стандарты серии ISO 9000:2000 и стандарты менеджмента CMM / CMMI [4] при управлении проектированием ПС. Однако положения этих стандартов не контролируют некоторые важные компоненты процессов в жизненном цикле ПС: характеристики качества ПС; интерфейсы Открытых систем; функциональную и информационную безопасность; комплекс технологической и эксплуатационной документации. Акцент методологии программной инженерии на крупные проекты предполагает сохранение и возможность применения ее базовых методов, процессов и стандартов при обеспечении жизненного цикла относительно небольших по объему кода проектов. Комплексное, скоординированное применение стандартов в процессе создания, развития и применения ПС позволяет исключать многие виды потенциально возможных или де-факто присутствующих в ПС дефектов, дают возможность значительно ослабить их влияние. Таким образом уровень качества создаваемых ПС становится предсказуемым и управляемым, непосредственно зависящим от ресурсов, выделяемых на его достижение, а главное – от системы качества и эффективности технологии, используемых на всех этапах жизненного цикла ПС.

Задачи внедрения стандартов в программную инженерию. К сожалению, многие отечественные специалисты в области программирования привыкли видеть в использовании стандартов рутину, сковывающую их творчество. Быстрое усложнение и рост размеров комплексов программ приводит к созданию крупных программистских коллективов с профессиональным разделением труда, в которых необходимо регламентировать и координировать деятельность команды специалистов над единым проектом. Особенно трудно осваиваются и внедряются в практику: формальное описание стандартных промышленных процессов жизненного цикла ПС, процедуры оценивания характеристик и документирования качества крупных программных продуктов. Обещания руководителей потенциальных разработчиков, которые фиксируются в контрактах с заказчиками и предполагают создание ПС высокого качества в согласованные сроки, во многих случаях не выполняются. Причиной этого, как правило, является различие договорившихся сторон в понимании требуемого качества, а также неумение оценить ресурсы, необходимые для его достижения. В результате качество программной продукции зачастую остается низким, оно не поддается достоверной оценке и оказывается неконкурентоспособным на международном рынке. Для эффективной деятельности коллектива специалистов, направленной на выполнение определенного проекта, на базе стандартов должен создаваться комплект регламентирующих документов программной инженерии. Каждый из таких документов должен быть предназначен для конкретных пользователей в жизненном цикле ПС. В них должны быть отражены:

- содержание и описание применяемых положений и разделов стандартов с позиции его конкретного пользователя;
- параметры адаптации разделов стандартов и содержание дополнительных нормативных документов;
- методика и сценарии корректного применения всех обязательных и рекомендуемых положений стандартов;
- требования к содержанию отчетов о результатах контроля и тестирования компонентов системы на соответствие обязательным положениям стандартов в процессе их жизненного цикла.

Детализацию требований и положений стандартов следует проводить с ориентацией на унификацию конкретных процессов, работ и документов программного продукта определенного функционального назначения. Можно выделить следующие основные группы специалистов, которые должны использовать такие стандарты:

- руководители крупного проекта системы и ее основных функциональных компонентов;
- системные аналитики, создатели спецификаций требований, проектов компонентов и алгоритмов решения функциональных задач;
- программисты-разработчики программных модулей, компонентов, структур, типов и содержания данных;
- интеграторы функциональных программных компонентов, тестирующие и отлаживающие крупные функциональные компоненты или ПС в целом;
- специалисты сопровождения и управления конфигурацией версий программных продуктов;
- испытатели и сертификаторы программных продуктов;
- разработчики технологий, инструментальных средств, методических, руководящих и инструктивных документов, обеспечивающих реализацию утвержденных стандартов для конкретного проекта ПС.

Учет психологии – важный фактор стандартизации процессов в программной инженерии. Стандарты регламентируют и, как следствие, стимулируют дисциплину труда коллектива специалистов при создании и применении крупных программных продуктов требуемого качества при ограниченных ресурсах. В целях эффективного внедрения этих стандартов необходимо

их изучение и освоение на практике, корректное применение отдельных требований и рекомендаций всеми участниками выполнения проекта, направленного на создание и применение ПС. По мере возрастания размеров проектов и ответственности за качество и безопасность применения программных продуктов, так же как и в любых других отраслях промышленного производства совершенствуются технологии их создания, повышаются требования к полноте и корректности применения стандартов.

Годами сложившийся богатый набор международных стандартов программной инженерии в значительной степени обеспечивает современный, регламентированный жизненный цикл комплексов программ. Этот набор продолжает успешно развиваться и совершенствоваться, что требует от коллективов разработчиков дополнительных усилий для анализа номенклатуры, текущего состояния и адаптации таких стандартов под особенности реальных проектов. При этом объективно возникают психологические трудности изучения и применения стандартов, обусловленные конфликтами между традиционным индивидуальным "хаотическим" (или "свободным") программированием небольших компонентов, и регламентированным коллективным производством крупных программных продуктов. Однако и это следует внедрять в сознание исполнителей таких программных проектов, только при втором подходе будет сохраняться концептуальная целостность применяемой совокупности стандартов и их эффективное, положительное влияние на процессы и результаты, на качество, надежность и безопасность программных продуктов при наличии реальных ограничений на использование необходимых для этого ресурсов.

2. Проблема влияния человеческого фактора в программной инженерии

Методология и стиль руководителей коллективов специалистов-разработчиков на практике значительно различаются. В процессе разработки программных продуктов творчество руководителей коллективов и отдельных специалистов – поиск методов, алгоритмов, альтернативных решений и способов осуществления заданных требований, а также формирование и декомпозиция этих требований составляют значительную часть всех человеческих затрат. Индустриализация разработки программных продуктов позволяет автоматизировать многие нетворческие, технические и рутинные операции и этапы, а также облегчает творческие процессы за счет селекции, обработки и отображения информации, необходимой для принятия творческих решений. Результатом такого подхода является возможность значительного сокращения доли затрат на творческий труд в непосредственных затратах специалистов на проектирование и производство комплексов программ.

Очень быстро увеличивается сложность и ответственность отдельных задач, связанных с обработкой информации и управлением, которые возлагаются на ЭВМ, что вызывает рост требований к качеству, надежности функционирования и безопасности применения программных продуктов. Каждое предприятие вынуждено тратить значительные средства на поиск, обучение и переподготовку специалистов. Возрастают требования к их профессиональной квалификации, появляется необходимость обучения специалистов ряду важных для программной индустрии профессий, связанных с созданием крупных программных продуктов, в том числе:

- организации и регламентированной работе больших профессиональных коллективов специалистов над целостным продуктом;
- распределению сотрудников разной профессиональной специализации по производственным этапам, компонентам и видам работ в жизненном цикле комплексов программ;
- планированию и методам работы в условиях ограниченных ресурсов, по графикам в реальном времени, с заданными сроками, контролем качества и документированием результатов;
- тестированию, испытаниям и обеспечению гарантии качества, надежности компонентов и программного продукта в целом.

Перечисленные профессиональные навыки способствуют формированию, совершенствованию и применению современной методологии и регламентированной инженерной дисциплины в процессе производства сложных программных продуктов для различных сфер жизнедеятельности общества. Быстрый рост сложности и повышение ответственности за качество таких продуктов привели к появлению новых требований к квалификации и дифференцированному подходу к обучению специалистов в области программной инженерии. В современных условиях им недостаточно навыков процедурного программирования небольших компонентов. Необходимы глубокие знания системотехники, технологии и стандартов проектирования, а также производства программных продуктов в определенной тематической области их применения. Эти специалисты должны владеть новыми интеллектуальными профессиями. Они призваны обеспечивать высокое качество программных продуктов, а также контроль, испытания и достоверность реально достигнутого качества на каждом этапе разработки и совершенствования комплексов программ. Накопленный опыт создания крупных программных систем и острый дефицит востребованных для выполнения таких работ специалистов привели к необходимости принципиального изменения и расширения методов и программ их обучения и воспитания. Крупномасштабное производство программных продуктов различных классов, разделение труда специалистов по профессиональной квалификации при разработке программ, структура и организация больших коллективов, а также экономическая сторона таких производств стали важнейшей частью процессов выбора, обучения и стимулирования специалистов для сопровождения всех этапов жизненного цикла сложных программных продуктов.

Личная профессиональная квалификация, психологические характеристики специалистов и организация крупного коллектива в основном определяют качество и трудоемкость при производстве программных продуктов. Разработка сложных программных комплексов, особенно на начальных и завершающих этапах, характеризуется наиболее высокой долей творческого труда. Дефекты, трудоемкость и длительность отдельных операций и частных работ существенно зависят от индивидуальных психологических особенностей их исполнителей, от функционального назначения и характеристик конкретного проекта. Принципиальной особенностью планирования производства комплексов программ является необходимость активного участия руководителей-менеджеров при отборе и подготовке профессиональных специалистов–создателей программных продуктов.

Трудности на этом направлении, как правило, возникают потому, что менеджеры и разработчики комплексов программ, как правило, не знают даже основ психологии специалистов-исполнителей, которые необходимы для производства сложной интеллектуальной продукции. Психологи при этом имеют недостаточное представление о природе и свойствах объектов разработки – программных продуктов, а также особенностей технологических процессов их производства и применения. Создание программных средств как производственной продукции существенно повысило актуальность обоснования, прогнозирования и оценивания роли человеческого фактора для характеристики качества процессов производства. Широкий спектр количественных и качественных показателей, которые с разных сторон характеризуют содержание компонентов и комплексов программ, объективно не высокий уровень достоверности при анализе их свойств и оценке значений, являются причиной трудностей и неопределенностей при попытке описать и измерить процессы производства сложных программных продуктов, учитывая при этом факторы психологической готовности и профессиональной компетенции их создателей.

Технологии регламентированного проектирования и производства крупных программных продуктов большими коллективами специалистов принципиально и в плане учета психологического фактора отличаются от технологий индивидуальной разработки небольших программ или комплексов программ свободным методом. К числу таких отличий относятся следующие:

- руководители больших коллективов специалистов должны быть способны выполнять роль лидеров, объединяющих и координирующих знания, навыки и труд над программным продуктом специалистов с разной профессиональной квалификацией и психологическими характеристиками;
- взаимодействие специалистов, творческая и психологическая совместимость в коллективе должны обеспечивать планируемое производство целостного программного продукта в реальном времени в заданные сроки и требуемого качества;
- при формировании коллектива и выполнении совместных работ необходимо учитывать и использовать особенности каждого специалиста в коллективе, которые отличаются профессиональной квалификацией и психологическими характеристиками;
- следует учитывать, что, как правило, сложно выделить персональное авторство и ответственность за реализацию отдельных функций и/или фрагментов, определяющих характеристики, качество, дефекты и риски программного продукта;
- нужно иметь в виду, что качество поставляемого программного продукта зависит от качества труда почти каждого специалиста и его персональной квалификации, однако не всегда можно выделить конкретного специалиста, ответственного за выявленные критические дефекты, ошибки и риски неблагоприятных событий при применении программного продукта.

В программных продуктах практически отсутствуют затраты на физические материалы и комплектующие компоненты. Все их достоинства и недостатки практически полностью определяются интеллектуальным трудом специалистов, их квалификацией, психологическими характеристиками и умением слаженно, ответственно работать в большом коллективе. В программной инженерии особое значение имеет параметр "реальное время" как ограниченный, непополняемый человеческий ресурс, который проявляется в следующих базовых группах процессов создания комплекса программ:

- значение реального времени входит в реализацию жизненного цикла, во все производственные процессы и определяет планы и сроки создания программных продуктов, тем самым с позиции заказчика и пользователей этот человеческий фактор определяет принципиальную необходимость и возможность создания продукта к определенному сроку;
- реальное время определяет внутренние процессы исполнения комплекса программ, качество, надежность и безопасность функционирования программного продукта в реальном времени в соответствии с его назначением, в зависимости от поступающей из внешней среды информации;
- реальное время определяет процессы во внешней среде, в системе управления и обработки информации или у пользователей, влияет на динамику реализации функций программного продукта при его применении.

При планировании размеров, сложности и трудоемкости производства конкретных программных продуктов интуитивные оценки заказчиков, руководителей и специалистов, как правило, существенно отличаются, в силу ошибок и тех, и других. Некоторые из перечисленных заинтересованных лиц зачастую психологически оптимистичны, и комплекс программ им кажется меньше по размеру, что приводит к нереальным оценкам числа функций, объема и сложности продукта и его компонентов. Следствием таких оценок являются большие ошибки при планировании затрат времени, качества и стоимости создания про-

граммных продуктов. Типичны ситуации, когда отставание сроков внедрения промышленных систем управления и обработки информации полностью зависит от неготовности и/или недостаточного качества разрабатываемых для них программных продуктов.

Как следствие роста сфер применения и ответственности функций, выполняемых программами, резко возросла необходимость гарантировать высокое качество программных продуктов, потребность регламентировать и корректно формировать требования к характеристикам подлежащих разработке комплексов программ, к их реализации и проверке достоверности выполнения требований. Сложность анализируемых объектов – комплексов программ и излишняя самоуверенность ряда руководителей и специалистов в собственной "непогрешимости", как правило, приводят к тому, что реальные характеристики качества функционирования и безопасности при применении программных продуктов остаются неизвестными не только для заказчиков и пользователей, но также и для самих разработчиков. Отсутствие в документах четкого декларирования понятий и требуемых значений характеристик качества продуктов зачастую вызывает конфликты между заказчиками-пользователями и разработчиками-поставщиками из-за разной трактовки одних и тех же характеристик.

Лидеры – руководители групп разработчиков, участвующих в производстве крупных программных продуктов, и высококвалифицированные специалисты в составе таких групп играют особую роль при формировании полноценных, корректных требований, которые должны осуществляться совместно с заказчиком и другими заинтересованными лицами, с участием экспертов по тематике области назначения продукта. Необходим подбор и эффективная организация работы специалистов для проверки корректности исходных требований на создание крупных программных продуктов. При формировании крупных производственных коллективов очень важно уметь учитывать психологические особенности и профессиональную компетентность их лидеров и специалистов. Для этого руководителям и специалистам – участникам работ по проектированию и производству крупных программных продуктов полезно знать и использовать основы психологии и, как следствие, учитывать характеристики людей, участвующих в производстве сложных продуктов. Для гарантии качества и безопасности применения программных продуктов руководителям и специалистам целесообразно знать и учитывать статистические свойства дефектов и ошибок в комплексах программ, их типы и источники, факторы, влияющие на их проявление и возможность обнаружения, а также их негативные последствия.

Многообразие классов сложных комплексов программ, обусловленное различными функциями и сферами применения систем управления и обработки информации, определяет формальные трудности, связанные с практическим использованием методов и процедур доказательства соответствия создаваемых и поставляемых программных продуктов условиям контрактов и требованиям заказчиков и потребителей. По мере расширения сфер применения и увеличения сложности систем выделились инфраструктуры и составляющие их объекты [5, 6], в которых дефекты и недостаточное качество комплексов программ могут наносить ущерб, значительно превышающий положительный эффект от их использования. В некоторых критических системах (например, управления атомными электростанциями, крупными банками или системами вооружения) проявления рисков неблагоприятных событий при использовании программных продуктов недопустимы. Подобные риски при применении комплексов программ могут определять безопасность функционирования объектов, предприятий и даже страны. Как следствие, резко повышается психологическая и юридическая ответственность коллективов, конкретных руководителей и специалистов за качество создаваемых ими программных продуктов.

Задачи организации структуры и состава коллектива специалистов в программной инженерии. Крупномасштабное проектирование ПС различных классов, разделение труда специалистов по квалификации при разработке программ и данных, структура и организация коллективов, а также экономика таких разработок стали важнейшей частью проблемы выбора и обучения специалистов для обеспечения всего жизненного цикла сложных программных продуктов. В жизненном цикле сложных комплексов программ участвуют специалисты различной квалификации и степени ответственности за результаты своей деятельности. Для организации эффективной структуры коллектива разработчиков ПС следует учитывать согласованность конкретных целей специалистов, участвующих в проекте, их психологическую совместимость, способность к дружной, коллективной работе, наличие опыта взаимодействия в составе определенного коллектива, а также другие объективные и субъективные факторы, характеризующие участников проекта. Большое значение при этом может иметь личная мотивация и психологические особенности поведения разных специалистов при комплексной работе над сложным проектом.

Уровень квалификации заказчика и неопределенности отдельных положений технического задания на разработку ПС могут сильно влиять на суммарные затраты и длительность создания комплекса программ. Первоначально техническое задание зачастую оказывается недостаточно квалифицированным и, как правило, подвергается в дальнейшем многократным изменениям. Изменения технического задания заказчиком и объем переделок непосредственно отражаются на производительности труда специалистов. Особенно сильно недостаточная обоснованность технического задания может быть обусловлена попытками заказчика форсировать сроки разработки. Этому же может способствовать разница между заказчиком и разработчиком в квалификации, уровне понимания целей разработки и необходимых затрат на реализацию возни-

кающих дополнительных требований. Даже при испытаниях заказчик зачастую обнаруживает, что решаются не совсем те задачи и не совсем так, как ему нужно, вследствие чего необходима значительная переработка уже созданных программ. Даже квалифицированные заказчики вынуждены иногда корректировать техническое задание на любых этапах разработки, что может влиять на снижение производительности на 10...20 %. Представители заказчика, участвующие в проекте, должны обучаться формализации автоматизируемых функций систем и технологических процессов, для которых предназначены соответствующие программные продукты, а также иметь представление об эффективных путях реализации таких функций.

При разработке сложных программ большими коллективами значительно повышается роль квалификации руководителей проекта, что непосредственно отражается на производительности труда всего коллектива. Известны случаи, когда только уровень квалификации руководителей крупных проектов изменял суммарные затраты на разработку в несколько раз, как в большую, так и в меньшую сторону. Некоторые методы организации структуры коллектива и процессов разработки позволяют сокращать негативное влияние этого человеческого фактора. Однако формализовать и учесть влияние особенностей конкретного руководителя разработки и ведущих специалистов на реализацию комплекса программ пока трудно.

Разработчики должны иметь в составе своего коллектива квалифицированных менеджеров, проблемно-ориентированных системных архитекторов, способных переводить не всегда полные и корректные функциональные требования заказчиков в конкретные спецификации и технические требования к комплексам программ и к их компонентам. Специалисты по проектированию сложных ПС (системные архитекторы), прежде всего, должны иметь хорошую подготовку по системному анализу алгоритмов и комплексов программ в определенной проблемной области, по методам оценки эффективности проектов, по организации и планированию крупных разработок компонентов программ и баз данных. Им необходима высокая квалификация по архитектурному построению, комплексной отладке и испытаниям ПС определенных классов, умение организовать коллектив для решения общей целевой задачи системы. Такая организация позволит уже на ранних этапах исключать или сокращать дефекты, обусловленные различием представления ими целей и задач проектов, а также их показателей качества.

Принципиальным путем улучшения экономических характеристик при разработке сложных ПС является исключение "творчества" на тех этапах, где возможны типовые, стандартные решения и использование апробированных заготовок программных компонентов, не требующих при их применении высококвалифицированного творческого труда. Основой такого подхода является применение унифицированной технологии, готовых, испытанных программных компонентов и стандартизированной архитектуры определенных классов ПС. Использование готовых апробированных модулей может почти исключать творческий труд по их программированию, автономной отладке и документированию. На этих этапах творческие усилия необходимы для отбора и контроля готовых компонентов, а также для разработки новых, отсутствующих среди уже существующих и апробированных. Однако практически полностью сохраняется творческий труд при системном анализе, при комплексировании компонентов и их комплексной отладке, а также во время испытания программного продукта в целом.

3. Проблема организации коллективов исполнителей проектов и обучения специалистов в области программной инженерии

Задачи организации коллективов исполнителей проектов в области программной инженерии. Для реализации крупных проектов, направленных на создание комплексов программ, как правило, применяются следующие две схемы организации коллективов исполнителей:

- формирование для выполнения каждого крупного проекта "жесткой" организационной структуры целостного коллектива с полным составом необходимых для его выполнения специалистов под единым, централизованным руководством лидера проекта;
- выделение руководителя (главного конструктора) и небольшой группы интеграторов, по заданиям которых "узкими" специалистами выполняются частные работы по отдельным компонентам, специалисты при этом организационно не включаются в состав единого коллектива для реализации конкретного крупного проекта.

Первая схема предпочтительна, если предприятие реализует небольшое число крупных проектов-заказов и имеет возможность для каждого из них комплектовать единую, организационно замкнутую "команду", которая полностью реализует проект и несет ответственность за его качество. Однако при таком подходе возможны простои отдельных специалистов из-за несинхронного исполнения заданий и ожидания результатов на последовательных запланированных этапах разработки модулей и компонентов другими специалистами.

Вторая схема может иметь преимущества для предприятия с большим числом относительно небольших проектов, близких по содержанию и функциональному назначению компонентов. В этом случае большинство специалистов одновременно участвуют в нескольких заказах по локальным заданиям лидеров и интеграторов различных проектов, и их потенциал может использоваться более полно. Однако задачи интеграторов при этом усложняются и требуют более высокой квалификации. Несмотря на то, что за качество проекта в целом также несут ответственность руково-

дитель (лидер) и группа интеграторов, данная схема усложняет взаимодействие с поставщиками компонентов, а также руководство и контроль их качества.

В обеих схемах для реализации мероприятий по планированию и управлению жизненным циклом концептуально целостных, крупных ПС и обеспечения их качества необходимы организационные действия системных архитекторов, направленные на подбор и обучение коллектива исполнителей разных категорий и специализаций. Практически в каждом успешном проекте должен быть лидер. Лидерами производства программного продукта могут быть: менеджер продукта, менеджер проектирования, руководитель проекта. Лидер должен иметь талант и высокий уровень квалификации, а также иметь навыки, позволяющие ему:

- руководить процессом выявления, конкретизации и формирования требований заказчика проекта;
- осуществлять проверку спецификаций программного средства, чтобы удостовериться, что они соответствуют реальной концепции, представленной детальными функциями;
- квалифицированно вести переговоры с представителями заказчика, с персонами, осуществляющими руководство проектом, с пользователями и разработчиками, определять и поддерживать должный баланс между запросами заказчика и возможностями команды разработчиков выполнить проект, используя выделенные заказчиком ресурсы, в течение запланированного на реализацию проекта времени;
- рассматривать "конфликтующие" пожелания, поступающие от различных участников проекта и находить компромиссы, необходимые для определения набора функций, которые в наибольшей степени удовлетворяют представителей всех сторон, заинтересованных в успешном выполнении проекта.

Чтобы добиться успеха в большом проекте, необходима четкая координация действий членов "команды", которая должна работать по общей, заранее принятой методологии, чтобы реализовать комплекс требований и обеспечить качество программного продукта эффективно организованной командой разработчиков. При этом одним из наиболее важных факторов является то, что члены "команды" имеют различные талант, профессиональные навыки и квалификацию. По этой причине при обучении специалистов важно учитывать возможную структуру большой команды. Руководство крупным проектом ПС должны осуществлять один или два лидера – менеджера. К их числу относятся:

- менеджер проекта – специалист, обеспечивающий связь между заказчиком и выполняющей проект командой исполнителей; его задача – определять и согласовывать способы удовлетворения требований заказчика;
- менеджер-архитектор комплекса программ, который управляет коммуникациями и отношениями между участниками проектной команды, является координатором процессов создания компонентов, разрабатывает базовые функциональные спецификации требований и управляет ими, ведет график реализации проекта и отчитывается за его состояние, инициирует принятие критичных для хода проекта решений.

Разделение труда специалистов этой категории в крупных проектных коллективах приводит к необходимости дифференцированного подхода к их обучению, к оценке квалификации и выбору деятельности. Суть такого подхода заключается в следующем:

- спецификаторы требований должны подготавливать описания функций и алгоритмов соответствующих компонентов с уровнем детализации, достаточным для корректной разработки программистами текстов программ и их интерфейсов;
- разработчики программных компонентов (программисты) должны создавать тексты программных компонентов, удовлетворяющие спецификациям требований, отслеживать и исправлять ошибки, возникающие при разработке программ сложных систем, для чего необходимо детальное знание языков программирования, сетевых технологий и проектирования баз данных;
- системные интеграторы сложных проблемно-ориентированных ПС должны работать над проектами в значительной степени отличными от программистов методами, на других языках проектирования, используя другие средства автоматизации и должны иметь на выходе результаты в виде крупных компонентов и комплексов программ;
- тестировщики должны обеспечивать проверку реализации функциональных спецификаций требований, систем обеспечения производительности, пользовательских интерфейсов, а также разрабатывать стратегию, планы и выполнять тестирование для каждой фазы и компонента проекта, они должны быть административно независимыми от программистов и спецификаторов;
- управляющие сопровождением и конфигурацией должны отвечать за реализацию и снижение затрат на модификацию и сопровождение программного продукта, за обеспечение максимально эффективной деятельности разработчиков по взаимодействию компонентов и реализации версий ПС, принимать участие в обсуждениях пользовательского интерфейса и архитектуры продукта;
- документаторы процессов и объектов жизненных циклов сложных ПС должны обеспечивать подготовку и издание сводных технологических и эксплуатационных документов в соответствии с требованиями стандартов.

Успех и качество при разработке сложных программных комплексов в значительной степени зависят от слаженности работы, квалификации и профессионализма коллектива, состоящего из перечисленных категорий специалистов, на всех этапах и уровнях создания крупных проектов. При выборе заказчиком надежного поставщика – разработчика проекта необходима оценка тематической и технологической квалификации предполагаемого коллектива специалистов, а также его способности реализовать про-

ект с заданными требованиями и показателями качества. Тематическую квалификацию специалистов в области создания ПС определенного функционального назначения в первом приближении можно характеризовать средней продолжительностью работы основной части команды исполнителей в данной проблемной области, непосредственно участвующей в разработке алгоритмов, спецификаций требований, программ и баз данных. Технологическая квалификация коллектива характеризуется опытом и длительностью работы с регламентированными технологиями, инструментальными комплексами автоматизации программной инженерии, языками проектирования, программирования и тестирования ПС. Важнейшую роль при этом играет квалификация руководителей – лидеров разработки и системных аналитиков функциональных компонентов и в меньшей степени непосредственных разработчиков программных компонентов в конкретной прикладной области. Особенно важны не столько индивидуальные характеристики каждого специалиста, сколько интегральный показатель квалификации команды, реализующей некоторую, достаточно крупную функциональную задачу или весь проект.

По мере расширения сферы применения сложных программных продуктов стало ясно, что интегральные затраты на их сопровождение и создание новых версий могут значительно превосходить затраты на разработку их первой версии. Это приводит к тому, что по мере накопления эксплуатируемых ПС и их компонентов все большее число специалистов переходит из области непосредственного написания новых программ в область системного проектирования, управления конфигурацией и создания новых версий программного продукта на базе повторно используемых компонентов. При организации сопровождения и модернизации крупных ПС следует учитывать важные психологические факторы, усложняющие отбор, обучение и деятельность менеджеров и квалифицированных специалистов в этой области, а именно:

- эта деятельность требует высокой квалификации, больших творческих и умственных затрат, связанных, прежде всего, с необходимостью одновременного широкого охвата и анализа множества компонентов ПС и их взаимосвязей, находящихся в различных состояниях завершенности модификаций или устранения дефектов;
- корректируемые компоненты зачастую разрабатывались в разное время, различными специалистами, в различном стиле и с неодинаковой полнотой документирования, что усложняет освоение их содержания при внесении изменений и устранении дефектов;
- творческая сторона работ при сопровождении ПС осложняется тем, что приходится овладевать и анализировать программы, разработанные ранее другими специалистами, которые, как правило, проще не корректировать, а разработать заново;
- комплексы программ, прошедшие всесторонние испытания и эксплуатацию у заказчиков, гарантируют достигнутое качество результатов функционирования, и любые изменения в них имеют высокий риск внесения дополнительных ошибок и ухудшения этого качества, что ограничивает возможности существенных модификаций;
- выполняемые работы требуют аккуратных и точных корректировок, четкого регламентированного взаимодействия специалистов, которые различаются квалификацией и уровнем ответственности;
- процессы и результаты сопровождения не отличаются наглядностью и внешним эффектом, которые адекватно представляли бы их размеры и сложность, вследствие чего они должным образом не оцениваются ни рядовыми программистами, ни руководителями проектов.

Зачастую менеджеры и разработчики нового ПС не предусматривают перечисленные факторы, их влияние на другие этапы его жизненного цикла, что значительно снижает эффективность последующего совершенствования и применения созданного программного продукта. Вместе с тем по некоторым оценкам непосредственно программированием новых компонентов в мире занято только около 15...20 % специалистов, участвующих в создании программных продуктов.

Задачи отбора и подготовки специалистов в программной инженерии. Перечисленные выше специализации и квалификация персонала, участвующего в выполнении крупных программных проектов, требуют соответствующего отбора и обучения кадров. Организация работы на этом направлении является самостоятельной, важной задачей программной инженерии. Обучение представляет собой сложный процесс и требует высокого уровня его организации и сопровождения. Должны быть подготовлены и документированы планы, зафиксированы требования и цели обучения, а также разработаны учебники и учебные пособия. Персонал, ответственный за выполнение конкретных задач, если это необходимо, должен быть аттестован на основе соответствующих требований к их подготовке и/или к опыту практической работы. Может быть необходима подготовка и ознакомление со специфической (проблемно-ориентированной) областью, в которой будет применяться программный продукт, и повышение квалификации в этой области.

Крупные программные продукты являются одними из наиболее сложных объектов, создаваемых человеком. В процессе их разработки творчество специалистов – поиск новых методов, альтернативных решений и способов реализации заданных требований, а также формирование и декомпозиция этих требований составляют значительную часть всех трудозатрат. Как в любой инженерной отрасли, квалифицированный программист должен развивать умения, позволяющие построить набор моделей и оценивать эти модели, управляя выбором компромиссов. В программной инженерии неуклонно повышаются размеры и сложность создаваемых ПС, что вызывает возрастание затрат творческого труда на единицу размера (объема кода) новых программ. В перспективе, несмотря на автоматизацию и повышение инструментальной осна-

щенности технологий разработки программ, доля творческого труда при создании полностью новых крупных программных продуктов возрастает. Даже при сокращении суммарных затрат на разработку программных компонентов за счет автоматизации нетворческого труда, все более определяющей для технико-экономических показателей создания ПС становится доля затрат на труд творческий. Как следствие, возрастают требования к творческим способностям при отборе и обучении специалистов.

По мере повышения квалификации коллектива и автоматизации творческой составляющей труда следует ожидать приближения проектов к предельным значениям относительных экономических характеристик полностью новых разработок. Эти значения определяются интеллектуальными возможностями человека по интенсивности принятия творческих решений. Им соответствуют наличие предельных значений производительности труда и длительности разработки сложных комплексов программ. Для их оценки необходимо изучение и экстраполяция прецедентов и экспериментальных данных реальных разработок ПС с наилучшими экономическими характеристиками с учетом возрастания квалификации специалистов и уровня автоматизации разработок. Вряд ли следует ожидать в ближайшие годы радикального повышения производительности труда при создании полностью новых, крупных программных продуктов. Еще более консервативна продолжительность реализации таких разработок.

Задачи обучения специалистов по программной инженерии. Специалисты по программной инженерии должны быть хорошо обучены и знать системотехнику вычислительных систем, поскольку в них программный продукт играет определяющую роль. Технологии программной инженерии являются одним из критических факторов при разработке сложных вычислительных систем. Высокие темпы роста основных доступных ресурсов аппаратных средств (приблизительно на порядок каждые пять лет), а также сохраняющаяся потребность в расширении сфер их применения со стороны отдельных категорий пользователей и факторов экономики приводят к необходимости адекватного совершенствования технологий создания программных комплексов и баз данных.

К сожалению, большинство вузов страны технологиям разработки ПС на основе положений программной инженерии пока не учит. Как правило, обучение студентов ограничивается элементами программирования только небольших по объему кода и простейших по логике программ. Они не готовят остро необходимых для современной программной индустрии системных аналитиков, архитекторов и менеджеров проектов ПС, специалистов по комплексированию, испытаниям и обеспечению качества крупных комплексов программ реального времени. Выпускники вузов, как правило, не знают современных промышленных методов, технологий и международных стандартов программной инженерии, поддерживающих и регламентирующих жизненный цикл ПС. Они не владеют инструментальными системами обеспечения качества, верификации, тестирования и сертификации сложных программных продуктов. Перечисленные недостатки определяют низкую системотехническую квалификацию специалистов и ряда отечественных предприятий, которые, тем не менее, берутся за создание крупных программных систем. В результате многие проекты сложных ПС оказываются неконкурентоспособными, недостаточного качества и требуют длительной доработки для устранения системных и технических дефектов и ошибок.

Задачи обучения специалистов методам программной инженерии и обеспечения современного качества комплексов программ при их практической деятельности оставляют широкое поле для произвола при оценивании качества программных продуктов и к появлению в них многочисленных дефектов и ошибок. Возрастание сложности и ответственности современных задач, которые решаются с использованием комплексов программ реального времени, а также возможного ущерба от неудовлетворительного качества получаемых с их помощью результатов, значительно повысило актуальность освоения учащимися методов полного, стандартизированного описания требований к характеристикам качества на различных этапах жизненного цикла ПС. Необходимо освоение студентами и специалистами понятий, определений и способов оценивания реальных характеристик качества программных продуктов. Особенно остро в последние годы выявилась необходимость систематизации реальных характеристик качества ПС, а также обучения студентов применению стандартов, выбора из них и адаптации необходимого набора характеристик и диапазонов их значений, которые необходимы для конкретных проектов комплексов программ.

Российским студентам, аспирантам и специалистам необходимо освоить современный комплекс задач, методов и стандартов промышленного создания и развития сложных, тиражируемых программных средств и баз данных с высокими требованиями к их качеству. Обучение должно быть ориентировано на коллективную, групповую работу команд специалистов над средними и крупными программными проектами. Следует акцентировать внимание на комплексе индустриальных методов и международных стандартов программной инженерии, которые непосредственно обеспечивают эффективный жизненный цикл сложных высококачественных программных продуктов и баз данных. Необходимо обучение специалистов современной программистской культуре промышленного создания высококачественных проектов ПС – умению формализовать требования и достигать конкретные значения характеристик функционирования и применения сложных комплексов программ с учетом тех ресурсов, которые выделяются для обеспечения и совершенствования качества таких комплексов.

Задачи подготовки учебных планов и курсов по программной инженерии. В последние годы в некоторых

отечественных вузах проявился интерес к программной инженерии и делаются попытки ввести такой курс в планы обучения студентов. В качестве основы для разработки учебных планов рассматриваются: международный стандарт – свод знаний по программной инженерии ISO 19759:2005 – ТО. – SWEBOK [7], а также Рекомендации по преподаванию программной инженерии и информатики в университетах SE2004 [8]. Документы ориентированы на создание стандартов обучения студентов, а также на разработку учебных планов по программной инженерии в рамках вузовских программ подготовки кадров в области прикладной математики, информатики, информационных технологий и систем. Издан ряд переводов на русский язык учебников по программной инженерии. На сайте Института системного программирования РАН (www.ispras.ru) в разделе *Публикации* аннотированы 5 учебников и 12 монографий автора настоящей статьи по ряду проблем программной инженерии. С полными текстами этих книг можно ознакомиться по адресу www.iqlib.ru.

Необходимо **разрабатывать и включать в учебные программы вузов по соответствующим специальностям учебный курс "Программная инженерия"** и поддерживающие его отдельные (специальные) курсы, которые могут быть разработаны на основе имеющейся отечественной и зарубежной литературы. К их числу могут относиться следующие:

- Экономика производства сложных программных продуктов [9];
- Требования к профессиональной квалификации руководителей и специалистов по программной инженерии [10];
- Методы обеспечения качества сложных программных средств [11];
- Тестирование модулей, компонентов и комплексов программ на соответствие требованиям [12];
- Функциональная безопасность программных средств [13];
- Документирование сложных программных средств [14];
- Сертификация программных продуктов [15].

В подготовке представленного материала активное участие принял доктор физ.-мат. наук, профессор Валерий Александрович Васенин, которому автор глубоко благодарен.

СПИСОК ЛИТЕРАТУРЫ

1. **Липаев В.В.** Программная инженерия. Методологические основы. М.: Изд-во гос. ун-та Высшая школа экономики ТЕИС. 2006. 608 с.

2. **ГОСТ** Р ИСО/МЭК 12207–99. Государственный стандарт Российской Федерации. Информационная технология. Процессы жизненного цикла программных средств (Information technology. Software life cycle processes). М.: Изд-во стандартов, 1999.

3. **Липаев В.В.** Отечественная программная инженерия: фрагменты истории и проблемы. М.: СИНТЕГ, 2007. 312 с.

4. **Оценка** и аттестация зрелости процессов создания и сопровождения программных средств и информационных систем (ISO/IEC TR 15504-CMM). М.: Книга и бизнес, 2001. 224 с.

5. **Андреев О.О. и др.** Критически важные объекты и кибертерроризм. Часть 1. Системный подход к организации противодействия / под ред. В.А. Васенина. М.: МЦНМ, 2008. 398 с.

6. **Андреев О.О. и др.** Критически важные объекты и кибертерроризм. Часть 2. Аспекты программной реализации средств противодействия / под ред. В.А. Васенина. М.: МЦНМ, 2008. 607 с.

7. **Guide** to the Software Engineering Body of Knowledge (SWEBOK). URL: http://www.computer.org/portal/web/swebok.

8. **Рекомендации** по преподаванию программной инженерии и информатики в университетах / Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. Computing Curricula 2001: Computer Science: пер. с англ. / пер. SE2004. Н.И. Бойко; пер. CC 2001. М.Е. Зверинцева. М.: Интернет-Университет Информационных Технологий, 2007. 462 с.

9. **Липаев В.В.** Экономика производства сложных программных продуктов. М.: СИНТЕГ, 2008. 432 с.

10. **Липаев В.В.** Человеческие факторы в программной инженерии: Рекомендации и требования к профессиональной квалификации специалистов. М.: СИНТЕГ, 2009. 348 с.

11. **Липаев В.В.** Методы обеспечения качества крупномасштабных программных средств. М.: РФФИ. СИНТЕГ, 2003. 520 с.

12. **Липаев В.В.** Тестирование крупных комплексов программ на соответствие требованиям. М.: СИНТЕГ, 2007. 300 с.

13. **Липаев В.В.** Функциональная безопасность программных средств. М.: СИНТЕГ, 2004. 348 с.

14. **Липаев В.В.** Документирование сложных программных средств. М.: СИНТЕГ, 2005. 216 с.

15. **Липаев В.В.** Сертификация программных средств. М.: СИНТЕГ, 2010. 344 с.

УДК 519.681

С.Д. Махортов, канд. физ.-мат. наук, доцент, зав. каф., Воронежский государственный университет

E-mail: sd@expert.vrn.ru

LP-структуры для обоснования и автоматизации рефакторинга в объектно-ориентированном программировании

В индустрии программного обеспечения важную составляющую образует направление, связанное с разработкой формальных моделей автоматизируемых объектов. Такие модели создают основу для эффективной верификации и оптимизации программного кода. В настоящей работе рассматривается класс основанных на решетках алгебраических структур, описывающих семантику иерархии типов в объектно-ориентированной программной системе. Исследуются свойства таких структур, включая замкнутость, эквивалентность преобразований, существование логической редукции. Методология предназначена для верификации и модернизации иерархий типов, важным направлением которой является автоматизированное устранение избыточности кода. В результате обобщения теоретической модели получен и формализован новый метод рефакторинга – совмещение атрибутов.

Кратко обсуждаются вопросы компьютерной реализации.

Ключевые слова: *иерархия типов, рефакторинг, алгебраическая модель, совмещение атрибутов, компьютерная реализация*

Введение

Алгебраические системы предоставляют основу формального построения и исследования компьютерных программ, основанных на различных парадигмах и технологиях [1–3]. Это в полной мере относится и к логическому программированию, включая широко распространенные на практике системы продукционного типа [4].

В ряде работ ([5] и библиография к ней) автором предложена методология алгебраизации продукционно-логических систем на основе решеток и бинарных отношений. Получены теоретические результаты для обоснования эквивалентных преобразований, верификации и оптимизации таких систем. Решетка с заданным на ней дополнительным продукционным отношением названа LP-структурой (*lattice production structure*).

Исследования показали, что данная технология применима в различных областях теории программирования. В частности, иерархии типов в объектно-ориентированных системах образуют разновидность решеток [6]. В работе [7] автор впервые установил, что отношения обобщения и агрегации типов обладают свойствами продукционно-логического вывода. В результате был построен класс LP-структур для моделирования иерархий типов в целях верификации, а также рефакторинга – модернизации кода. В этой модели в LP-структуре из решеточных операций в качестве основной использовалась лишь операция объединения. Данное обстоятельство ограничило возможности теории формализацией единственного метода рефакторинга – поднятия общих атрибутов (обзор известных методов представлен в работах [8, 9]).

Решение родственных задач алгебраическими методами анализа формальных понятий (FCA) представлено в работе [10], где элементам определенного множества классов предлагается в некотором смысле оптимально назначить наборы атрибутов – элементов другого независимого множества. В соответствии с выбранными назначениями формируется иерархия

классов. В постановке, которая рассматривается в настоящей работе, в отличие от работы [10], атрибуты сами относятся к исследуемой иерархии классов (типов), что усложняет задачу и не оставляет возможности непосредственного применения методов FCA.

Эффективный аппарат формализации иерархий типов представляют описанные в работах [11, 12] многоуровневые упорядоченно-сортные алгебры. Однако, как справедливо замечено в работе [13], разработка этих алгебр еще не завершена.

Настоящая статья развивает теорию LP-структур для иерархий типов. Теперь при определении LP-структуры в качестве базовой используется решеточная операция пересечения. В результате естественного обобщения алгебраической модели [7] (применения двойственности операций) получен и формализован *новый метод рефакторинга*. Суть данного метода, прежде не упоминавшегося в классическом перечне рефакторингов [8], состоит в замене нескольких атрибутов класса их общим потомком (*совмещение атрибутов*). Для нового вида LP-структур изучен стандартный круг основных вопросов: существование замыкания, архитектура, эквивалентные преобразования. Представлена теорема о существовании логической редукции бинарного отношения и способе ее построения. Указанные результаты расширяют возможности автоматизированных исследований и модернизации иерархий типов.

В разделе 1 данной работы приводятся необходимые базовые сведения из теории отношений и решеток. Раздел 2 содержит предварительное обсуждение рассматриваемых задач на конкретных примерах, а также их приложений. Данное обсуждение служит иллюстрацией и отправной точкой для построения LP-структуры в разделе 3, где также формулируются основные теоретические результаты. В связи с ограниченным объемом статьи доказательства сформулированных в ней утверждений не приводятся. Тем не менее, ввиду двойственности построенной модели по отношению к рассмотренной в работе [7], общее представление о методике доказательств может быть получено в последней.

В разделе 4 обсуждаются вопросы компьютерной реализации LP-структур, включая доказательства оценок вычислительной сложности решаемых задач.

1. Основные понятия и обозначения

Бинарное отношение R на множестве F называется:

- рефлексивным, если для всех $a \in F$ справедливо $(a, a) \in R$;
- транзитивным, если для любых $a, b, c \in F$ из (a,b), $(b, c) \in R$ следует $(a, c) \in R$.

Существует замыкание R^* произвольного отношения R относительно свойств рефлексивности и транзитивности – рефлексивно-транзитивное замыкание (РТЗ). Пара элементов $a, c \in F$ называется транзитивной в R, если $(a, c) \in R_1^*$, где R_1^* – РТЗ отношения $R_1 = R \setminus \{(a, c)\}$.

Обратная задача – нахождение транзитивной редукции: по заданному R ищется минимальное отношение R' такое, что его РТЗ совпадает с РТЗ для R [14]. Как обычно, для частично упорядоченных множеств различаются понятия минимального элемента (для него нет меньшего элемента) и наименьшего элемента (он меньше всех). В работе [14] представлен алгоритм построения транзитивной редукции конечного отношения; показано, что эта задача вычислительно эквивалентна построению РТЗ.

Необходимые для чтения статьи сведения о решетках содержатся в работе [15]. Решеткой называется частично упорядоченное множество **F**, в котором наряду с отношением ≤ ("не больше", "содержится") определены также две двуместные операции $\wedge$ ("пересечение") и $\vee$ ("объединение"), вычисляющие соответственно точную нижнюю и верхнюю грани любой пары $a, b \in \mathbf{F}$.

Решетка называется ограниченной, если она содержит общие верхнюю и нижнюю грани – такие два элемента O, I, что $O \le a \le I$ для любого $a \in \mathbf{F}$.

2. Обсуждение задач и приложений

Рассмотрим иерархию типов **F** в объектно-ориентированной программной системе. Между парами типов могут существовать как минимум два вида связей – наследование (тип наследует атрибуты типа-предка) и агрегация (тип содержит в качестве атрибута представителя другого типа) [8].

Отношение наследования порождает на **F** частичный порядок: если тип b является потомком a (соответственно a – предком b), то $b \le a$. Требуется, чтобы для любых $a, b \in \mathbf{F}$ были определены две решеточные операции: пересечение $a \wedge b$ – наибольший общий потомок; объединение $a \vee b$ – наименьший общий предок a, b (первая операция актуальна в прикладных системах с множественным наследованием). Для ограниченности решетки добавим к **F** два специальных элемента: I – универсальный тип (общий предок, имеется в ряде современных систем программирования) и O – фиктивный потомок всех типов.

На решетке **F** рассмотрим второе, соответствующее агрегации, отношение R: если экземпляр типа a в качестве атрибута содержится в определении типа b, то $(b, a) \in R$. Оба отношения (≤ и R) имеют общую семантику: в каждом случае $b \le a$ или $(b, a) \in R$, тип b получает возможности типа a в виде доступа к его атрибутам. Семантически понятно, что это общее отношение "обладания набором возможностей" (обозначим его $\xleftarrow{R}$) обязано быть рефлексивным и транзитивным. Обсудим другие свойства введенных отношений. Пусть для элементов $a, b_1, b_2 \in \mathbf{F}$ справедливо $b_1 \le a$, $b_2 \le a$. Тогда по определению решетки имеем $b_1 \vee b_2 \le a$. Это естественное для отношения ≤ свойство (согласно [5]) называется $\vee$-дистрибутивностью. Посмотрим, что будет означать обладание этим же свойством для отношения $\xleftarrow{R}$. Пусть $b_1 \xleftarrow{R} a$ и $b_2 \xleftarrow{R} a$, т.е. каждый тип b_1 и b_2 обладает возможностями типа a. Тогда в силу предполагаемой $\vee$-дистрибутивности имеем $b_1 \vee b_2 \xleftarrow{R} a$. Последнее означает, что тип $b_1 \vee b_2$ так-

же обладает возможностями типа a. С точки зрения проектирования типов это необязательно. Однако если более одного типа-наследника (в данном случае – b_1 и b_2) содержат одинаковые атрибуты, то согласно принципу рефакторинга [8] целесообразно "поднять" общие атрибуты, т.е. поместить один такой атрибут в общий тип-предок $b_1 \vee b_2$, после чего каждый b_1 и b_2 получит возможности a в порядке наследования. В рассмотренной ситуации $\vee$-дистрибутивность отношения $\xleftarrow{R}$ содержит решение важной задачи – устранение дублирования кода.

Последнее из отмеченных свойств формализовано и исследовано в работе [7]. В ней, в частности, показано, что отношение $\xleftarrow{R}$, обладая свойством транзитивности вне зависимости от контекста, не может во всех ситуациях удовлетворять свойству $\vee$-дистрибутивности, так как это приведет к некорректным результатам. Продемонстрируем указанное обстоятельство на примерах.

Пример 1. Для иллюстрации рассмотрим пример: $b \xleftarrow{R} a$ и $a \xleftarrow{R} a$. При $\vee$-дистрибутивности $\xleftarrow{R}$ выполнялось бы $b \vee a \xleftarrow{R} a$. Согласно принципам объектно-ориентированного программирования, тип $b \vee a$ не имеет права что-либо знать о своих наследниках. По этой причине в данной ситуации $b \vee a$, являясь общим предком типов a и b, может обладать возможностями типа a лишь в случае, если он совпадает с a (т.е. $b \le a$). В остальных вариантах соотношение $b \vee a \xleftarrow{R} a$ окажется некорректным.

Пример 2. Существуют ситуации, когда выполнение $\vee$-дистрибутивности отношения $\xleftarrow{R}$ теоретически возможно, однако нецелесообразно с точки зрения качества кода. Пусть при $b_1 \xleftarrow{R} a$, $b_2 \xleftarrow{R} a$ элементы $b_1 \vee b_2$ и a имеют непустое пресечение: $(b_1 \vee b_2) \wedge a = d \neq O$, причем $d < b_1 \vee b_2$ и $d < a$. Если в данном случае допустить $(b_1 \vee b_2, a) \in R$, то окажется, что тип d обладает возможностями типа a одновременно по двум линиям, а именно – как его потомок и как потомок типа $b_1 \vee b_2$, также имеющего возможности a. Недостаток такого кода состоит в его избыточности – разрыв связи $d < a$ (при $a \neq I$) не приведет к потере функциональности системы типов.

Пример 3. Другая подобная ситуация – наличие конфликта. Пусть имеет место (b_1, a), (b_2, a), $(b_3, a) \in R$, причем элементы b_1, b_2, b_3, $b_1 \vee b_2$, $b_2 \vee b_3$ попарно различны и $(b_1 \vee b_2) \wedge (b_2 \vee b_3) = b_2$. Тогда пары (b_1, a), (b_2, a) "конфликтуют" с парами (b_2, a), (b_3, a): если в обоих случаях "поднять" атрибуты, то тип b_2 унаследует атрибут типа a одновременно от двух различных предков – $b_1 \vee b_2$ и $b_2 \vee b_3$, что также ухудшит код.

В работе [7] формально описаны возможные ситуации подобных коллизий и построена соответствующая LP-структура с ограниченным свойством $\vee$-дистрибутивности. Принятая стратегия предполагает отказ от "поднятия" общих атрибутов при наличии описанных ситуаций (невыполнение $\vee$-дистрибутивности). Теоретически возможны и другие подходы, более тонко учитывающие особенности конкретных систем.

Как известно, например, из алгебраической логики [16], "решеточные" операции объединения и пересечения порождают в алгебраических системах двойственные свойства (например законы де Моргана). Подобная закономерность имеет место и в разработанных автором стандартных LP-структурах [5], где наряду с $\vee$-дистрибутивностью бинарных отношений рассмотрено симметричное свойство – $\wedge$-дистрибутивность. Однако применительно к модели иерархии типов [7] такое свойство до сих пор рассмотрено не было. Выясним теперь, что означает свойство $\wedge$-дистрибутивности применительно к указанному выше отношению $\xleftarrow{R}$.

Предположим, что для элементов a_1, a_2, $b \in \mathbf{F}$ выполнено $b \le a_1$, $b \le a_2$. Тогда по определению решетки справедливо $b \le a_1 \wedge a_2$. Такое свойство частичного порядка $\le$ на решетке называется $\wedge$-дистрибутивностью [5]. Распространим его на отношение $\xleftarrow{R}$. Пусть $b \xleftarrow{R} a_1$ и $b \xleftarrow{R} a_2$, т.е. тип b обладает возможностями типов a_1 и a_2. В силу предполагаемой $\wedge$-дистрибутивности получим $b \xleftarrow{R} a_1 \wedge a_2$. Таким образом, тип b также обладает возможностями типа $a_1 \wedge a_2$. Как и выше, с точки зрения проектирования типов последнее соотношение обязательным не является. Его семантика такова: если тип имеет два или более различных атрибута, то эти атрибуты можно заменить единственным атрибутом, относящимся к ближайшему общему потомку типов исходных атрибутов. Нетрудно заметить, что такая реорганизация может сделать определение типа-контейнера более компактным с сохранением его функциональности. Итак, на основе формального рассмотрения двойственных свойств LP-структур получен новый метод рефакторинга (по крайней мере, он отсутствует в известном перечне [8]). Назовем его "совмещением атрибутов". Следует отметить, что данный метод актуален лишь для иерархий типов с разрешенным множественным наследованием. Тем не менее, его существование можно оправдать наличием одного лишь языка C++, который, по-видимому, будет жить до тех пор, пока нужны эффективные компьютерные программы.

Выясним далее вопрос о том, насколько в данной алгебраической модели типов свойство $\wedge$-дистрибутивности отношения $\xleftarrow{R}$ универсально, и не окажутся ли его ограничения двойственными по отношению к описанным выше ограничениям $\vee$-дистрибутивности.

Пример 4. Для иллюстрации рассмотрим случай двойственности по отношению к примеру 1: $b \xleftarrow{R} a$ и $b \xleftarrow{R} b$. При $\wedge$-дистрибутивности отношения $\xleftarrow{R}$ должно быть выполнено $b \xleftarrow{R} b \wedge a$. Аналогично исходному примеру 1 тип b не имеет права что-либо знать о своем типе-наследнике $b \wedge a$. По этой причине тип b, являясь предком типа $b \wedge a$, может обладать его возможностями лишь в случае $b \le a$, иначе соотношение $b \xleftarrow{R} b \wedge a$ окажется некорректным.

Пример 5. Пусть при $b \xleftarrow{R} a_1$, $b \xleftarrow{R} a_2$ элементы b и $a_1 \wedge a_2$ имеют объединение $b \vee (a_1 \wedge a_2) = d \neq I$, причем $b < d$ и $a_1 \wedge a_2 < d$. Если в данном случае допустить $(b, a_1 \wedge a_2) \in R$, то окажется, что тип b обладает возможностями другого типа d одновременно по двум линиям, а именно и как его потомок, и как контейнер типа $a_1 \wedge a_2$, также имеющего возможности d в порядке наследования. Недостатки такого кода (подобно примеру 2) заключаются в его избыточности – разрыв связи $a_1 \wedge a_2 < d$ не приведет к потере функциональности системы типов.

Пример 6. Рассмотрим конфликтную ситуацию, двойственную по отношению к примеру 3. Пусть $(b, a_1), (b, a_2), (b, a_3) \in R$, причем элементы a_1, a_2, a_3, $a_1 \wedge a_2$, $a_2 \wedge a_3$ попарно различны и $(a_1 \wedge a_2) \vee (a_2 \wedge a_3) = a_2$. Тогда пары (b, a_1), (b, a_2) "конфликтуют" с парами (b, a_2), (b, a_3). Этот факт означает, что если в обоих случаях совместить атрибуты (применив свойство $\wedge$-дистрибутивности), то тип b получит возможности типа a_2 одновременно посредством двух атрибутов – $a_1 \wedge a_2$ и $a_2 \wedge a_3$, что также ухудшит код.

Анализируя примеры 4–6 в сравнении с соответствующими примерами 1–3, приходим к выводу о двойственном характере ограничений свойств $\wedge$-дистрибутивности и $\vee$-дистрибутивности отношения $\xleftarrow{R}$, которые необходимы для адекватного моделирования иерархий типов. Отмеченный факт служит дополнительным подтверждением естественности разрабатываемых алгебраических моделей, и, в частности, вводимых ограничений дистрибутивности.

Заметим, что с помощью LP-структур рассматривается обобщенная постановка задач "распределения возможностей" между типами. Варианты, когда типы по каким-либо практическим соображениям содержат в виде атрибутов представителей одних и тех же типов под различными идентификаторами, в расчет не принимаются. На практике тип может содержать много атрибутов, но не все они одинаково существенны при построении иерархии. Кроме того, подчеркнем, что алгебраические модели в большей степени предназначены для автоматизированного рефакторинга, чем для автоматического, т.е. окончательное решение о конкретных преобразованиях типов остается за программистом.

На основе представленных выше соображений в следующем разделе определяется понятие логического бинарного отношения на ограниченной решетке. Оно отражает свойство "обладания набором возможностей" в иерархии типов. Логическое замыкание произвольного отношения на решетке предоставляет все такие пары (b, a), что в типе b доступны возможности типа a. Решив задачу построения логического замыкания, можно автоматизировать верификацию системы типов. К такого сорта задачам, в частности, относится исследование LP-структуры на наличие циклов. Исходя из предметной области, можно также формулировать правила, которым должна удовлетворять система типов, и контролировать их выполнение. Например, возможна проверка системы на наличие необходимых или отсутствие запрещенных логических связей ("автомобиль обладает возможностями руля", "руль не имеет двигателя" и другие подобные им). Логическая редукция предоставляет иерархию с минимальным эквивалентным набором связей в рамках рассматриваемой модели и, соответственно, снижает избыточность кода.

Таким образом, представленный в работе формализм позволяет проводить автоматизированные исследования иерархий типов, включая эквивалентные преобразования, верификацию и оптимизацию. Он может служить основой для практической реализации (или модернизации) типов.

В предыдущих работах автора [5] были изучены бинарные отношения, обладающие без ограничений как свойством транзитивности, так и дистрибутивности (в обоих ее смыслах). Область их применения – продукционные системы с монотонным выводом. В частности, понятие $\wedge$-дистрибутивности отношения $\xleftarrow{R}$ задает семантику монотонного вывода в том смысле, что из $b \xleftarrow{R} a$, $b \xleftarrow{R} b$ следует $a \xleftarrow{R} a \wedge b$. В данной работе (как и в работе [7]) в силу проведенного выше обсуждения свойство дистрибутивности зависит от контекста. Этот факт, в частности, порождает немонотонный характер логического вывода на рассматриваемых LP-структурах.

3. LP-структуры и их основные свойства

В разделе 2 изложены общие идеи формализованного представления иерархии типов на основе LP-структуры. Далее будет дано формальное определение соответствующей алгебраической системы.

Вначале сформулируем некоторые понятия и условия, связанные с ограничением свойства $\wedge$-дистрибутивности продукционно-логических отношений. Как отмечалось в п. 2, это свойство описывает совмещение атрибутов в определении типа. В данном случае трудности формализации обусловлены попыткой ввести статическое формальное условие для описания динамического процесса. Более точно – условие $\wedge$-дистрибутивности отношения R на любых подходящих парах (b, a_1), $(b, a_2) \in R$ должно быть выполнено как до совмещения атрибутов, относящихся к типам a_1, a_2, так и после него. В дальнейшем при ссылках на примеры для краткости будут использоваться фразы "совмещение атрибутов a_1, a_2", или просто "совмещение a_1, a_2", если это не вызовет противоречий. Такое тем более возможно потому, что формальные определения данного раздела связаны не с типами, а с элементами абстрактной решетки.

Определение 1. Пусть R – бинарное отношение на ограниченной решетке $\mathbf{F}$. Две пары вида (b, a_1), $(b, a_2) \in R$ называются $\wedge$-совместимыми в R, если существуют такие $c_1, c_2 \in \mathbf{F}$, что $c_1 \wedge c_2 \leq a_1 \wedge a_2$, причем $(c_1 \wedge c_2) \vee b = I$, а пары (b, c_1), (b, c_2) нетранзитивны в $R \cup \leq$. При этом набор элементов $T = (b, c_1, c_2)$ будем называть $\wedge$-дистрибутивной тройкой, а $C = (b, a_1, a_2, c_1, c_2)$ – $\wedge$-дистрибутивным кортежем (в R).

Поясним определение 1 на примере. Предположим, что тип b непосредственно содержит атрибуты a_1 и a_2, причем имеет место $(a_1 \wedge a_2) \vee b = I$. Тогда, в соответствии с обсуждением п. 2, атрибуты a_1 и a_2 могут быть совмещены, т.е. заменены одним атрибутом $a_1 \wedge a_2$. До этого действия имеем $c_1 = a_1$, $c_2 = a_2$. После совмещения атрибутов условие определения 1 остается выполненным, но с другими промежуточными элементами: $c_1 = c_2 = a_1 \wedge a_2$.

Сформулированное в определении 1 условие нетранзитивности пар (b, c_1), (b, c_2) обеспечивает совмещение атрибутов, относящихся непосредственно к типу b, а не косвенных, связанных с b цепочкой наследований и агрегаций. Конечно, атрибуты неявно совмещаются вместе со своими "возможностями", но именно непосредственные атрибуты должны удовлетворять условиям определения 1.

Следующее понятие формально описывает возможные варианты конфликтов между тройками элементов, претендующими на свойство $\wedge$-дистрибутивности (см. также о конфликте пар в примере 6).

Определение 2. Рассмотрим $\wedge$-дистрибутивную тройку $T = (b, c_1, c_2)$, а также тройку $T' = (b', c_1', c_2')$, проверяемую на аналогичное свойство. Тройка T называется нейтрализующей для T' (обозначим $T' \prec T$), если выполнено одно из условий:

1) $b' = b$, $c_1 \wedge c_2 \neq c_1' \wedge c_2'$ и справедливо хотя бы одно из неравенств $c_1 \wedge c_2 < c_i'$;

2) $b' < b$, $c_1 \wedge c_2 \neq c_1' \wedge c_2'$ и выполнено хотя бы одно из неравенств $c_1 \wedge c_2 \leq c_i'$;

3) $b' < b$, $c_1 \wedge c_2 = c_1' \wedge c_2'$.

С неформальных позиций данное определение описывает ситуации, когда наличие нейтрализующей тройки T "угрожает" свойству $\wedge$-дистрибутивности тройки T'. Рассмотрим с этой точки зрения условия 1–3 определения 2.

Пусть имеет место условие 1. Тогда, после возможного совмещения атрибутов c_1 и c_2 (т.е. их замены атрибутом $c_1 \wedge c_2$), в силу соотношений $b' = b$, $(b, c_1 \wedge c_2) \in R$, $c_1 \wedge c_2 < c_i'$ пара (b', c_i') окажется транзитивной в $R \cup \leq$, что сделает невозможной $\wedge$-дистрибутивность тройки T'. Условие 1, в частности, содержит конфликт, который иллюстрируется в примере 6 предыдущего раздела работы. Для него справедливы оба соотношения $T' \prec T$ и $T \prec T'$.

Если для T, T' выполнен вариант 2, то после совмещения атрибутов c_1 и c_2 получим соотношения $b' < b$, $(b, c_1 \wedge c_2) \in R$, $c_1 \wedge c_2 < c_i'$, что вновь означает транзитивность в $R \cup \leq$ пары (b', c_i') и, соответственно, аннулирует свойство $\wedge$-дистрибутивности тройки T'.

В случае выполнения условия 3 после совмещения атрибутов c_1 и c_2 придем к ситуации, когда обе пары (b', c_i'), $i = 1, 2$ окажутся транзитивными в $R \cup \leq$: $b' < b$, $(b, c_1 \wedge c_2) \in R$, $c_1 \wedge c_2 = c_1' \wedge c_2' < c_i'$.

Тройку $T' = (b', c_1', c_2')$ будем называть неконфликтной, если для нее не существует ни одной нейтрализующей $\wedge$-дистрибутивной тройки.

Понятия, введенные в определении 2 (и далее) для троек вида $T = (b, c_1, c_2)$ и $T' = (b', c_1', c_2')$, автоматически распространяются на соответствующие им кортежи $C = (b, a_1, a_2, c_1, c_2)$ и $C' = (b', a_1', a_2', c_1', c_2')$.

Две $\wedge$-совместимые пары (b, a_1), (b, a_2) называются неконфликтно $\wedge$-совместимыми, если для них существует неконфликтный $\wedge$-дистрибутивный кортеж (b, a_1, a_2, c_1, c_2).

Отношение R на решетке **F** называется ограниченно $\wedge$-дистрибутивным, если для любых неконфликтно $\wedge$-совместимых в R пар (b, a_1), (b, a_2) справедливо $(b, a_1 \wedge a_2) \in R$.

Определение 3. Отношение называется логическим с ограничением пересечений (в данной работе – просто логическим), если оно содержит отношение $\leq$, транзитивно и ограниченно $\wedge$-дистрибутивно. Логическим замыканием отношения R называется наименьшее логическое отношение, содержащее R и его множество неконфликтно $\wedge$-совместимых пар.

Два отношения R_1 и R_2, определенные на общей решетке, называются эквивалентными ($R_1 \sim R_2$), если их логические замыкания совпадают. Логической редукцией отношения R называется эквивалентное ему минимальное отношение R_0. Заметим, что при этом не требуется вложения $R_0 \subseteq R$. В принципе можно говорить о некотором отношении R_0 как логической редукции вообще, не относя этот факт к какому-либо другому отношению R. Это означает, что при изъятии любой пары меньшее отношение не будет эквивалентно R_0.

Для выяснения вопроса о существовании логического замыкания и логической редукции введем следующее понятие логической связи.

Определение 4. Пусть задано произвольное бинарное отношение R на решетке **F**. Будем констатировать, что упорядоченная пара $b, a \in \mathbf{F}$ логически связана отношением $R(b \xleftarrow{R} a)$, если выполнено одно из следующих условий:

1) $(b, a) \in R$;

2) $b \leq a$;

3) существуют такие $a_1, a_2 \in \mathbf{F}$, что $a = a_1 \wedge a_2$, причем $b \xleftarrow{R} a_1$, $b \xleftarrow{R} a_2$ и пары (b, a_1), (b, a_2) неконфликтно $\wedge$-совместимы;

4) существует элемент $c \in \mathbf{F}$ такой, что $b \xleftarrow{R} c$ и $c \xleftarrow{R} a$.

Условия 1–4 определения 4 будем также называть правилами (вывода). Справедлива следующая теорема.

Теорема 1. Для произвольного отношения R на решетке **F** логическое замыкание существует и совпадает с множеством $\xleftarrow{R}$ всех упорядоченных пар, логически связанных отношением R.

Далее обсудим вопросы, связанные с эквивалентными преобразованиями рассматриваемых логических структур. Пусть дано отношение R на решетке **F**. Его эквивалентным преобразованием называется такая замена множества упорядоченных пар R, что полученное в результате новое отношение P эквивалентно R.

В [5] для логических отношений с монотонным выводом показано, что локально-эквивалентное преобразование части исходного множества R приводит к логически эквивалентному общему отношению P. Для рассматриваемых в настоящей статье отношений это утверждение в общем неверно: если при $T' \prec T$ преобразовать отдельно пары из T', общий результат окажется неверным. Однако имеет место эквивалентность ряда преобразований.

Теорема 2. Пусть R – отношение на решетке **F**. Тогда каждая из следующих операций на R приводит к эквивалентному отношению:

- добавление или исключение пары (b, a), если $b \le a$;
- добавление пары $(b, c_1 \wedge c_2)$, если тройка $T = (b, c_1, c_2)$ $\wedge$-дистрибутивна и неконфликтна в R;
- добавление или исключение пары (b, a) при наличии пар (b, c), $(c, a) \in (R \cup \le)$, где a, b, c попарно различны.

В работе [5] и других работах автора для структур с монотонным выводом показано, что логическое замыкание отношения R совпадает с РТЗ другого отношения $\widetilde{R} \supseteq R$, построенного в виде некоторого "дистрибутивного многообразия" над R. Этот факт позволяет свести некоторые вопросы, касающиеся логических отношений, к соответствующим проблемам транзитивных отношений. В частности, построение логического замыкания или редукции можно осуществить с помощью быстрых алгоритмов (типа Уоршолла) [14]. В рамках модели, основанной на ограниченной $\wedge$-дистрибутивности, также удается разделить процесс построения логического замыкания на этапы дистрибутивного и рефлексивно-транзитивного замыканий.

Для отношения R на решетке **F** рассмотрим отношение $\widetilde{R}$, построенное последовательным выполнением следующих двух шагов:

- для каждой неконфликтной $\wedge$-дистрибутивной тройки (b, c_1, c_2) $(c_1 \ne c_2)$ добавить к исходному отношению пару $(b, c_1 \wedge c_2)$;
- к полученному отношению добавить отношение $\le$.

Заметим, что по теореме 2 отношение $\widetilde{R}$ эквивалентно R.

Теорема 3. Логическое замыкание отношения R совпадает с рефлексивно-транзитивным замыканием $\widetilde{R}^*$ соответствующего отношения $\widetilde{R}$.

Выясним также вопрос о существовании и построении логической редукции рассматриваемых LP-структур. Справедлива следующая теорема.

Теорема 4. Пусть для отношения R построено соответствующее отношение $\widetilde{R}$. Тогда, если для $\widetilde{R}$ существует транзитивная редукция R^0, то отношение $\widetilde{R^0}$, полученное исключением из R^0 всех пар вида $b \le a$, представляет собой логическую редукцию исходного отношения R.

4. Алгоритмические вопросы

В данном разделе кратко обсуждаются некоторые вопросы практической реализации рассмотренных выше LP-структур и, соответственно, вычислительной сложности построения их логического замыкания и редукции. Настоящая работа в основном посвящена теоретическому обоснованию решения этих задач, а создание готовых к реализации оптимальных алгоритмов может стать предметом отдельной статьи. Однако краткое обсуждение алгоритмических вопросов поможет составить общее представление о практической применимости теории LP-структур на решетках типов.

Заметим сразу, что разработка алгоритмов на абстрактных решетках и получение оценок их сложности в общем случае представляется "неблагодарным делом". Например, часто встречающийся вид решетки булеан при числе атомов n состоит из 2^n различных элементов со всеми вытекающими "алгоритмическими последствиями".

Для общих решеток разработаны методы их представления деревьями и битовыми векторами [17]. В частности, булеан может быть представлен множеством битовых векторов размерности n. Каждому атому решетки соответствует вектор с одной единицей в соответствующей позиции и остальными нулями. Вычисление решеточных операций сводится к вычислению побитовых логических операций сложения и умножения над компонентами векторов. Нельзя сказать, что операция над двумя векторами будет выполняться за константное время, поскольку разрядность компьютера (например, 32 или 64) может оказаться недостаточной для представления битового вектора единственным словом памяти. Однако значение сложности операции $n/32$ или $n/64$ вполне приемлемо при общем числе элементов решетки 2^n.

Решетка типов обычно не является дистрибутивной и число ее элементов не так велико. По этой причине *в данном случае* все же допустимо выбрать в качестве основы анализа сложности величину N – общее число элементов решетки, а также хранить саму решетку и бинарные отношения на ней в виде матриц смежности размера $N \times N$. Таким образом, в рамках настоящей работы можно абстрагироваться от низкоуровневых проблем представления решеток.

При получении оценок будем обращаться к быстрым алгоритмам построения транзитивного замыкания и транзитивной редукции бинарных отношений. Классический алгоритм Уоршолла для графа с N вершинами строит замыкание за $O(N^3)$ шагов. Существуют улучшения данного алгоритма (например, $O(N^{\log_2 7})$), их обзор имеется в работе [14], согласно которой транзитивная редукция конечного графа вычисляется со сложностью нахождения транзитивного замыкания.

Итак, пусть имеется решетка **F**, представленная матрицей смежности $N \times N$. Построим для нее матрицу достижимости, что составит не более $O(N^3)$ шагов (алгоритм Уоршолла). С помощью этой матрицы операция $a \le b$ $(a, b \in \mathbf{F})$ вычисляется за время $O(1)$. Операции $a \wedge b$ и $a \vee b$ сводятся к нахождению соответственно максимума и минимума при однократном просмотре решетки, т.е. занимают $O(N)$ шагов каждая. Вспомнив, что результаты этих операций также принадлежат решетке, построим еще две матрицы $N \times N$, в

первой из которых для каждой пары a, b (число пар составит $O(N^2)$) будет храниться индекс элемента $a \wedge b$, во второй – $a \vee b$. Построение каждой такой матрицы обойдется в $O(N^3)$ шагов, и в дальнейшем с их помощью любую "решеточную" операцию можно будет выполнить за время $O(1)$.

Пусть также задано отношение R на решетке **F**. Как обсуждалось в п. 3, задачи нахождения логического замыкания и редукции данной LP-структуры связаны с рассмотрением $\wedge$-дистрибутивных троек вида $T = (b, c_1, c_2)$, где $(c_1 \wedge c_2) \vee b = I$, а пары (b, c_1), (b, c_2) нетранзитивны в $R \cup \leq$. Для избавления от влияния транзитивных пар сразу построим транзитивную редукцию отношения $R \cup \leq$, затем исключив также из полученной матрицы R все подчиненные пары (т.е. пары вида $a \leq b$). Этот процесс займет не более $O(N^3)$ шагов.

На следующем этапе сформируем все $\wedge$-дистрибутивные тройки отношения R. С указанной целью будем просматривать решетку (N шагов) и для каждого элемента b по соответствующему столбцу матрицы R находить всевозможные подходящие элементы c_1, c_2 ($O(N^2)$ действий). Таким образом, полный набор $\wedge$-дистрибутивных троек **T** может быть найден за время $O(N^3)$.

Очередной этап алгоритма (построение логического замыкания или редукции) состоит в том, чтобы в полученном множестве **T** оставить лишь неконфликтные тройки. Для этого достаточно попарно их сопоставить. Согласно определению 2, сопоставление двух троек осуществляется за время $O(1)$, а общее число сопоставляемых пар квадратично зависит от числа троек $|\mathbf{T}|$. Общее число троек, составленных из произвольных элементов решетки, равно $C_N^3 = O(N^3)$ (число сочетаний из N по 3). Казалось бы, эту оценку можно попробовать улучшить, используя свойства $\wedge$-дистрибутивной тройки $T = (b, c_1, c_2)$. К сожалению, следующий ниже пример опровергает такие надежды, и в результате сложность построения множества неконфликтных $\wedge$-дистрибутивных троек $\mathbf{T}_0$ займет время $O(N^6)$. Полученные здесь оценки также оправдывают хранение множества троек в виде булевского трехмерного массива $N \times N \times N$, который обеспечит, в частности, любую операцию доступа к множеству за время $O(1)$.

Приведем частный пример отношения R на решетке **F**, при котором число $\wedge$-дистрибутивных троек составит $O(N^3)$. С указанной целью вначале разобьем решетку на две (почти) равночисленные группы элементов $\{b_i, i = 1, ..., N/2\}$ и $\{c_j, j = 1, ..., N/2\}$. Далее сформируем отношение $R = \{(b_i, c_j),\ i = 1, ..., N/2;\ j = 1, ..., N/2\}$. Оно является ациклическим и состоит лишь из нетранзитивных пар. В этой ситуации для каждого элемента b_i можно составить $\wedge$-дистрибутивные тройки вида $T = (b_i, c_{j_1}, c_{j_2})$ $(j_1 \neq j_2)$ в количестве $O(C_{N/2}^2)$. Здесь также предполагается, что в рассматриваемой решетке число различных элементов вида $c_{j_1} \wedge c_{j_2}$ (они не участвуют в построении троек) является константным. Таким образом, получим $|\mathbf{T}| = \frac{N}{2} O(C_{N/2}^2) = O(N^3)$.

Также вполне возможным является случай, когда и $|\mathbf{T}| = O(N^3)$.

Далее по схеме п. 3 требуется построить дистрибутивное замыкание $\tilde{R}$ исходного отношения R. Перебрав с этой целью все множество $\mathbf{T}_0$, потратим соответственно времени $O(N^3)$. Если в качестве исходной стояла задача нахождения логического замыкания, то по теореме 3 остается вычислить рефлексивно-транзитивное замыкание отношения $\tilde{R}$, что займет не более $O(N^3)$ операций. Если же изначально решалась задача нахождения логической редукции, то, согласно теореме 4, построим для отношения $\tilde{R}$ транзитивную редукцию R^0 (число действий – $O(N^3)$), откуда с использованием матрицы **F** за время $O(N^2)$ найдем требуемый результат.

Таким образом, объединяя представленные выше оценки, можно сформулировать следующую теорему.

Теорема 5. Задачи нахождения логического замыкания и логической редукции LP-структуры на решетке типов решаются за полиномиальное время, не превышающее $O(N^6)$, где N – общее число элементов решетки.

СПИСОК ЛИТЕРАТУРЫ

1. **Подловченко Р. И.** Иерархия моделей программ // Программирование. 1981. № 2. С. 3–14.

2. **Нигиян С.А., Аветисян С.А.** О семантике бестиповых функциональных программ // Программирование. 2002. № 3. С. 5–14.

3. **Замулин А.В.** Алгебраическая семантика императивного языка программирования // Программирование. 2003. № 6. С. 51–64.

4. **Davis R., King J.** An overview of production systems // Mahine Intelligence. Chichester : Ellis Horwood Limited. 1977. V. 8. P. 300–332.

5. **Махортов С.Д., Подвальный С.Л.** Алгебраический подход к исследованию и оптимизации баз знаний продукционного типа // Информационные технологии. 2008. № 8. С. 55–60.

6. **Тейз А., Грибомон П. и др.** Логический подход к искусственному интеллекту: от классической логики к логическому программированию: пер. с франц. М.: Мир, 1990. 432 с.

7. **Махортов С.Д.** LP-структуры на решетках типов и некоторые задачи рефакторинга // Программирование. 2009. Т. 35. № 4. С. 5–14.

8. **Фаулер М.** Рефакторинг: улучшение существующего кода: пер. с англ. СПб.: Символ-Плюс, 2004. 432 с.

9. **Mens T., Tourw'e T.** A Survey of Software Refactoring // IEEE Trans. on Software Engineering. Feb. 2004. V. 30(2). P. 126–139.

10. **Godin R., Valtchev P.** Formal Concept Analysis-Based Class Hierarchy Design in Object-Oriented Software Development // Formal Concept Analysis / eds. B. Ganter, G. Stumme, R. Wille // Lecture Notes In Computer Science. Springer Berlin/Heidelberg. 2005. V. 3626. P. 304–323.

11. **Erwing M.** Specifying Type Systems with Multi-Level Order-Sorted Algebra // Proc. of 3rd Int. Conf. On Algebraic Method. and Software Technol. 1993. P. 179–188.

12. **Erwing M., Guting R.** Explicit Graphs in a Functional Model for Spatial databases. Report 110, Fern University Hagen. 1991. Revised Version. 1993.

13. **Вагин В.Н., Головина Е.Ю., Загорянская А.А., Фомина М.В.** Достоверный и правдоподобный вывод в интеллектуальных системах / Под ред. В.Н. Вагина, Д.А. Поспелова. М.: Физматлит, 2004. 704 с.

14. **Aho A.V., Garey M.R., Ulman J.D.** The transitive reduction of a directed graph. SIAM J. Computing 1:2. 1972. P. 131–137.

15. **Биркгоф Г.** Теория решеток: пер. с англ. М.: Наука, 1984. 568 с.

16. **Расёва Е., Сикорский Р.** Математика метаматематики: пер. с англ. М.: Наука, 1972. 591 с.

17. **Halib M., Nourine L.** Bit-vector encoding for partially ordered sets // Lect. Notes Comput. Sci. 1994. V. 831. P. 1–12.

УДК 004.4; 004.451

Е.Ю. Шутилин, магистрант, **Л.Б. Атымтаева,** канд. физ.-мат. наук, проф., Казахстанско-Британский Технический университет, г. Алматы, Казахстан

E-mail: atymtayeva@gmail.com

Оптимизация разработки программного обеспечения на основе методики "Канбан"

Дано описание новой, так называемой "гибкой", методики управления процессом командной разработки программного обеспечения "Канбан", которая изначально использовалась в производственной системе Toyota Production System. Показаны варианты адаптации методики к различным проектам. В работе используются аналитические данные, полученные при применении методики в течение года в реальном проекте по разработке программного обеспечения.

Ключевые слова: *гибкая методология разработки, программная инженерия, "Канбан", разработка программного обеспечения*

Введение

Термин Software – программное обеспечение (ПО), впервые появился в 1958 году. Международный стандарт, дающий единое представление о процессах разработки ПО, разрабатывался 5 лет с 1990 года. Отрасль создания программного обеспечения находится в процессе зарождения и сегодня принято говорить, что процесс создания ПО является более ремеслом, нежели чем промышленной индустрией, поскольку в программировании до сих пор нет системы знаний о закономерностях создания программ.

В данной статье описана методология "Канбан" для управления проектами разработки программного обеспечения, которая имеет все основания стать фундаментом в отрасли создания ПО.

1. Основные положения и применимость методологии "Канбан"

Термин "Канбан" пришел в область создания ПО из производственной системы Toyota (TPS), где им обозначалась система сигналов, используемая в концепции "точно вовремя" (*Just In Time*, JIT) [1].

"Канбан" является "легковесным" представителем методологии разработки ПО из группы Agile [2]. Также в эту группу входят известные методологии Scrum и XP (*eXtreme Programming*). Плюсами легковесности являются меньшее число формальных процессов, связанных с управлением проектом, и упрощенные стадии анализа и проектирования. Основной акцент в таких методологиях сделан на разработку функциональности, совмещение ролей и неформальные коммуникации, меньшее количество документации. Соответственно, минусы таких методологий – сильная зависимость эффективности работы от индивидуальных способностей членов команды, проект требует более квалифицированной, универсальной и стабильной работы; объем и сложность выполняемой работы ограничены, риски срыва проекта высоки. Таким образом, "Канбан" подходит для проектов малой и средней сложности с числом участников от 2 до 20 и, возможно, больше.

"Канбан" позволяет зафиксировать определенный технологический процесс и получать основные аналитические данные о ходе выполнения задач в рамках этого процесса. "Канбан" ориентирован на операционную деятельность – продолжающийся и повторяемый во времени процесс, в котором задачи ставятся на конвейер и последовательно выполняются командой разработчиков, дизайнеров, архитекторов, бизнес-аналитиков. Специфика проекта может быть обозначена как развитие и поддержка, но не создание нового проекта, поскольку такая деятельность по определению является проектной, а не операционной [5].

Теория методологии "Канбан" уже описана достаточно подробно и в Интернете есть достаточное количество вариаций методологии [3–6], поэтому здесь бу-

дет дано описание и анализ конкретной реализации на примере проекта из трех разработчиков, использующих "Канбан" в течение года.

Для применения методологии "Канбан" в проекте использовалась доска 57×45 см на клейкой основе (рис. 1, см. третью сторону обложки) и разноцветные карточки – листки для заметок размером 9×9 см. Наш вариант доски имеет три рабочих области: очередь для типовых задач, очередь для нетиповых задач и, собственно, основное рабочее пространство "Канбан" – так называемая производственная линия, которая делится на шесть этапов и визуализирует технологический процесс разработки одной типовой задачи. Здесь заметим, что определение *задачи* в контексте статьи – это одна или совокупность нескольких функций в информационной системе, выполняющая определенные действия и реализуемая на языке программирования.

Очередь для типовых задач (оранжевые листки), подлежащих реализации. Типовые задачи – те задачи, которые должны быть решены посредством нашего технологического процесса, с участием всех присутствующих в команде ролей – бизнес-аналитика, архитектора, дизайнера, кодера, тестера и писателя технической документации. Пока задачи находятся в этой области, можно менять их приоритет или вообще убрать из очереди – по решению менеджера проекта.

В **очереди для нетиповых задач** – синие листки – и так называемых **"багов"** (ошибок, сбоев программы) – розовые листки – находятся нестандартные задачи, которые так или иначе возникают в любом проекте, требуют какую-то часть рабочего времени и решаются одним из специалистов. "Баги" имеют наибольший приоритет и решаются в первую очередь соответствующим разработчиком. "Синие" задачи решаются в зависимости от нагрузки команды и по решению менеджера.

На области производственной линии размечены следующие этапы: разработка бизнес-логики, разработка интерфейса и архитектура СУБД, кодирование, тестирование, документирование и, наконец, внедрение задачи в рабочую среду. В ходе выполнения задачи карточка движется поэтапно слева направо. На каждом этапе ответственный сотрудник или группа сотрудников реализуют определенный объем работы. На первом этапе бизнес-аналитик рисует блок-схемы, диаграммы состояний, пишет техническое задание. Если задача небольшая, он вербально объясняет детали архитектору, дизайнеру и программисту. Здесь всей команде поясняют цели задачи, определяются информационные сущности, связи между ними, проясняются возможные нюансы в реализации, которые необходимо будет протестировать. На следующем этапе дизайнер создает пользовательский интерфейс и текстовки, а архитектор уточняет и оптимизирует сущности, атрибуты, связи и операции в бизнес-модели задачи и переносит модель в БД. Далее задача непосредственно реализуется программистом. Затем задача тестируется – обязательно другим разработчиком, не кодером. Как правило, тестируются функциональные точки со сложной логикой, нюансы, обозначенные на этапе "Бизнес-логика", и проводятся стресс тесты. На следующем этапе реализованная задача документируется: юрист делает корректировки в необходимые документы; менеджер по работе с клиентами обновляет справочный центр, готовит рассылку и соответствующую новость. На последнем этапе разработчики внедряют задачу в производственную среду, менеджер обновляет материалы на сайте, делает рассылку.

Карточка может двигаться и в обратном направлении, например, если тестер нашел серьезный архитектурный недочет в коде, задача может вернуться обратно к архитектору на доработку.

Цифры в нижней части этапа согласно источникам [4, 7] призваны ограничить число выполняемых задач одновременно. Цифра означает число задач, которое должно выполняться на этом этапе. Если число задач растет, то эффективность команды падает. Число подбирается или экспериментально [3], или устанавливается менеджером, который хорошо знает разработчиков.

Карточка также имеет три рабочие области – две для временного периода и одна текстовая – информационная. Карточки (рис. 2, см. третью сторону обложки) делятся на цвета согласно характеру задачи: типовая (рис. 2, *а*), нетиповая (рис. 2, *б*) и ошибка ("баг") (рис. 2, *в*).

Типовая задача – это та задача, которая попадает на производственную линию и выполняется всей командой последовательно. Для отслеживания времени выполнения задачи на карточке присутствует дата постановки задачи из очереди на производственную линию и дата внедрения задачи в производство. Ответственные за этап сотрудники сами определяют, когда задача завершена, и ее можно перемещать дальше. Следует отметить, что целью технологии "Канбан" является снижение времени выполнения одной типовой задачи. На карте можно было бы добавить еще несколько функциональных областей, например для определения скорости работы каждого этапа, и фиксировать даты выполнения карандашом (чтобы править, если задача возвращается назад), но, как оказалось, в этом необходимости нет – на данный момент визуального контроля достаточно.

Нетиповая задача обычно связана с инфраструктурой всего проекта, начиная с определения политики безопасности и заканчивая установкой антивирусов на рабочие станции.

Как уже было сказано выше, "баги" (ошибки) имеют наивысший приоритет, требуют устранения в первую очередь. Над "багом" работает тот разработчик, который непосредственно реализовывал задачу.

2. Возможности и преимущества методологии "Канбан"

Говоря о возможностях методологии "Канбан" следует отметить следующее.

Во-первых, "Канбан" позволяет вычислять среднее время работы команды над одной задачей, используя минимум формальных инструментов. Этими данными можно оперировать при планировании. Например, скорость выполнения работы за первый квартал 2010 года составила 14 типовых задач, 14 "багов" и нетиповых задач; за первое полугодие 2010 года – 14 типовых, 18 "багов" и нетиповых задач (до недавнего времени "баги" и нетиповые задачи не разделялись).

Здесь стоит отметить, что исследователи Университета Нового Южного Уэльса Майкл Лоуренс (*Michael Lawrence*) и Росс Джеффри (*Ross Jeffery*) в обзоре 1985 года привели оценки производительности различных проектов. Как это не парадоксально, но лучшими по производительности оказались те проекты, в которых шеф не оказывал на команду никакого временного давления [8].

Во-вторых, применение технологии "Канбан" позволяет сгруппировать задачи по приоритетам и упорядочить порядок их выполнения. Это актуально, поскольку зачастую работа в команде выполняется "как есть", без учета приоритетов, с частыми прерываниями на сторонние задачи. В итоге результатом деятельности такой команды становится ПО низкого качества.

Как преимущество можно отметить то, что использование технологии поэтапной реализации каждой задачи позволило тщательней продумывать детали реализации каждой задачи и значительно снизить число возникающих ошибок.

Заключение

В итоге отметим, что "Канбан" в разработке ПО – это методология анализа и контроля выполнения типовых задач, реализуемых по заранее определенному технологическому процессу. "Канбан"–Agile методология, которая требует высокой квалификации каждого участника команды. Многие решения принимаются локально. Методика может применяться для поддержки и развития информационных систем и подсистем с числом персонала от 2 до 20 человек. К сожалению, "Канбан" в примере не решает ряд управленческих задач, таких как управление сроками и рисками, оценка трудозатрат, определение критериев приема задач.

СПИСОК ЛИТЕРАТУРЫ

1. **Лайкер Дж.** Дао Toyota: 14 принципов менеджмента ведущей компании мира: пер. с англ. 2-е изд. М.: Альпина Бизнес Букс, 2006. 400 с.

2. **Архипенков С.** Лекции по управлению программными проектами. Москва, 2009. 128 с.

3. **Kanban**, Flow and Cadence. URL: http://availagility.co.uk/2008/10/28/kanban-flow-and-cadence/

4. **"Канбан"** в IT (Kanban Development) URL: http://bishop-it.ru/2009/07/kanban-development/

5. **Шутилин Е.** Организация командной работы разработчиков посредством методики "Канбан" // Сборник трудов международного конкурса студенческих проектов по информационным технологиям. Алматы, 2010 г. 14 апреля. С. 169–173.

6. **Kanban** kick-start example. URL: http://blog.crisp.se/henrikkniberg/2009/11/16/1258359420000.html.

7. **Kanban**. URL: // http://www.crisp.se/kanban

8. **Демарко Т., Листер Т.** Человеческий фактор: успешные проекты и команды: пер. с англ. 2-е изд. М.: Символ плюс, 2008. 256 с.

При публикации состава редакционной коллегии в первом номере журнала была допущена опечатка в фамилии Дзегелёнок И.И. Приносим извинения за допущенную опечатку.

Редакция журнала

УДК 004.41

В.А. Васенин, д-р физ.-мат. наук, проф., зав. отделом, Институт проблем информационной безопасности МГУ им. М.В. Ломоносова, **А.И. Гирча,** канд. физ.-мат. наук, стар. науч. сотр., Институт механики МГУ им. М.В. Ломоносова

E-mail: vasenin@msu.ru

Программный комплекс для проведения наукоемкого вычислительного эксперимента на основе вихревых методов численного моделирования процессов нестационарной гидродинамики

Рассмотрены вопросы построения с использованием элементов программной инженерии эффективного комплекса для проведения вычислительного эксперимента на основе вихревых методов численного моделирования процессов нестационарной гидродинамики. Последовательно изложены результаты основных этапов его разработки, включая формализацию и систематизацию требований к целевому программному комплексу, описание его общей архитектуры, техническое проектирование, разработку эффективных алгоритмов, их программную реализацию и тестирование. Результаты тестовых испытаний подтверждают соответствие свойств комплекса предъявляемым к нему требованиям.

Ключевые слова: *программный комплекс, программная инженерия, качество программного обеспечения, метрики оценки сложности программного обеспечения, вычислительная гидродинамика*

Введение

Создание программ и их комплексов для проведения наукоемкого вычислительного эксперимента представляет собой отдельную, обладающую определенной спецификой область прикладной математики и информатики. Характерной ее особенностью является наличие большого числа объективно возникающих и постоянно изменяющихся требований к подобным программам. Такого сорта изменения возникают в силу условий, в которых программа используется. Вычислительный эксперимент обладает циклической структурой, в основе которой лежит триада, сформулированная академиком А.А. Самарским, "модель – алгоритм – программа" [8, 9]. В процессе расчетов математическая модель исследуемого явления и вычислительный алгоритм подвержены постоянным модификациям [9]. Они, в свою очередь, приводят к необходимости вносить соответствующие изменения. Причины изменений могут также носить чисто технический характер, например, в случае разработки более эффективных способов представления данных или при необходимости организовать взаимодействие с внешним устройством. Следует заметить, что в условиях вычислительного эксперимента программа выступает в роли, которую выполняет перманентно модифицируемый стенд при эксперименте физическом. С учетом изложенного, первая версия программы рассматривается лишь как отправная точка в последовательности дальнейших многочисленных ее изменений.

Построение эффективных программных решений, предназначенных для проведения вычислительного эксперимента, представляет собой очень сложную задачу. В подтверждение этого факта отметим слова,

сказанные в 1977 г. академиком Н.Н. Яненко в работе [11], посвященной проблемам математического моделирования.

"Вместе с ростом числа ЭВМ и задач трудности технологические и организационные все больше стали преобладать над трудностями "чисто научными". Сейчас масштаб и объем этих трудностей настолько вырос, что можно говорить, что задача их преодоления сама стала задачей науки и представляет собой проблему фундаментального значения".

Отметим, что в настоящее время, спустя более 30 лет после выхода данной работы, организационно-технологические вопросы проведения вычислительных экспериментов приобрели еще более острый характер. При этом Н.Н. Яненко считал, что к задачам, связанным с ЭВМ и программированием, которые традиционно считались чисто технологическими, необходим более серьезный, математический подход.

В 1990 г. вышла работа [7], в которой академик О.М. Белоцерковский и профессор В.В. Щенников следующим образом охарактеризовали проблемы, связанные с технологическими аспектами проведения вычислительных экспериментов.

"Бурное развитие вычислительной техники, которое особенно ярко проявилось в последние 10–15 лет, с особой остротой поставило проблему создания принципиально новой технологии решения задач на ЭВМ.

...Исторически сложилось так, что проблемы численного моделирования (в это понятие мы включаем собственно математическое моделирование, сопряженное с численным экспериментом), будучи заметно продвинуты еще в "домашинный" период и развиваясь опережающими темпами в последующие периоды, оказались наиболее консервативной компонентой современной математической технологии решения задач на ЭВМ. Прибегая, может быть к излишней с точки зрения математиков образности изложения, можно охарактеризовать сложившуюся ситуацию двумя устойчивыми тенденциями:

- увеличение сложности математических моделей;
- построение очень изощренных математических методов.

Обе отмеченные тенденции неизбежно приводят к технологическому тупику, поскольку создают большие сложности в решении задачи создания программно-аппаратных средств поддержки функционирования всей технологической цепочки... Не претендуя на глубину и значимость аналогии, мы отваживаемся утверждать, что ситуация, складывающаяся в современном численном моделировании, схожа с ситуацией, наблюдавшейся в механике перед появлением основных идей и концепций квантовой механики".

В настоящей работе рассматриваются вопросы разработки и реализации теоретически обоснованных подходов к созданию эффективного комплекса программ для проведения вычислительных экспериментов на основе вихревых методов численного моделирования нестационарных течений вязкой несжимаемой жидкости. Представленные выше мнения свидетельствуют о сложности и практической значимости рассматриваемых задач. В качестве базового вихревого метода в контексте настоящей работы рассматривается численный метод вязких вихревых доменов [2, 5, 6]. Он был разработан в МГУ имени М.В. Ломоносова несколько лет назад. В отличие от других методов этого класса, таких как метод Чорина [17], метод диффузионной скорости [18] и метод перераспределения завихренности [20], метод вязких вихревых доменов является строго обоснованным. Он не содержит подгоночных параметров и позволяет корректно учитывать влияние вязкости среды. С помощью данного метода удается эффективно исследовать различные двумерные сопряженные задачи аэрогидродинамики о нестационарном взаимодействии твердых тел с течениями несжимаемой жидкости. Подходы на его основе с успехом применяются в вычислительных экспериментах по исследованию широкого класса задач, в частности, по изучению гидродинамических механизмов машущего полета, моделированию ветроэнергетических установок роторного и волнового типа. Об актуальности темы, рассматриваемой в настоящей работе, свидетельствует отсутствие подходов (моделей и механизмов) к построению программных реализаций для рассматриваемого вычислительного эксперимента, которые удовлетворяют современным требованиям к его проведению. В частности, в программных реализациях, созданных с использованием традиционных подходов, отсутствуют механизмы учета упоминавшихся ранее, объективно возникающих и перманентно изменяющихся требований к эксперименту, среде окружения и к вычислительным алгоритмам.

В настоящее время известны два подхода к организации программного обеспечения для вычислительного эксперимента на основе метода вязких вихревых доменов. Первый подход применяют "предметники" – специалисты по вихревым методам, в меньшей степени знакомые с технологиями программирования. Для этого подхода характерна множественность программных реализаций, каждая из которых предназначена для решения определенных, "фиксированных" задач аэрогидродинамики. Основным недостатком этого подхода является отсутствие явно сформулированных инвариантов, свойств таких программ, а также четко (формально) определенных взаимосвязей между ними. Как следствие, с течением времени происходит неконтролируемое нарастание сложности программных реализаций, понижается эффективность проведения вычислительного эксперимента, увеличиваются временные затраты на подготовку и проведение массовых расчетов. Передача таких программ другим исследователям представляет большие сложности. Второй подход основан на использовании программных комплексов с монолитным вычислительным ядром. Примером программной реализации такого типа является программный комплекс "Ротор" [4]. В ее ядре реализован алгоритм, универсальный для некоторого класса задач. В рамках данного подхода можно до-

биться ряда полезных свойств программной реализации, например, отсутствия дублирования кода. Такие программные комплексы, как правило, должным образом документированы, возможна их передача потенциальным пользователям. Однако для программных комплексов этого типа также характерен ряд недостатков, основной причиной которых является подход к созданию таких комплексов на основе принципа универсальности используемого при реализации вычислительного ядра. По мере расширения ядра на новые классы задач происходит значительное усложнение языка задания расчетов. Более того, начиная с некоторого момента времени добавить новую функциональность в вычислительное ядро становится крайне сложно в силу трудностей, связанных с разработкой универсального алгоритма для довольно широкого класса задач. В результате снижается эффективность подготовки и проведения численных расчетов, увеличиваются временные затраты.

Следует отметить, что на некоторые из перечисленных выше недостатков традиционно используемых подходов указывал в своих работах академик А.А. Самарский [9]. Представленные соображения подтверждают не только практическую значимость, но и фундаментальный характер исследований, результаты которых приведены далее.

1. Цели и методы их достижения

Работа, результаты которой представлены далее, не преследует цели изложения и разрешения всех технологических проблем моделирования. Рассматриваемый программный комплекс ориентирован на решение более узкого класса задач на преодоление некоторых трудностей, которые возникают при создании и сопровождении программного обеспечения в рамках вычислительного эксперимента на основе метода вязких вихревых доменов. В качестве тестовой задачи для проверки свойств программного комплекса рассматривается задача о течении вязкой жидкости в вихревой ячейке. Вихревыми ячейками называют поперечные профилированные вырезы, расположенные на поверхности обтекаемого тела [10]. Задачи такого рода рассматриваются в рамках международного проекта *VortexCell2050* [21].

Конечной целью работы является создание эффективного, с использованием элементов программной инженерии, программного комплекса, позволяющего минимизировать временные затраты, необходимые для подготовки и проведения вычислительных экспериментов на основе метода вязких вихревых доменов.

При создании программного комплекса используются современные методы и технологии разработки программных систем. Такие методы и технологии основаны на формальных моделях описания эксперимента и элементах программной инженерии, позволяющих верифицировать код на основе данных вычислительного и натурного экспериментов. Данные методы претерпели значительные изменения за последние несколько десятилетий. Полное и точное описание таких изменений представляет собой сложную задачу. Среди важных понятий современного подхода к разработке программных систем, используемых в настоящей работе, необходимо отметить следующие [14, 19]: жизненный цикл программной системы; модель жизненного цикла; программная архитектура; паттерн проектирования. В работе используются элементы компонентного подхода.

Другой способ снижения временных затрат на проведение рассматриваемого вычислительного эксперимента заключается в разработке и использовании более эффективных алгоритмов вычисления различных характеристик выбранного численного метода. В работе предлагаются адаптированные к особенностям проводимого эксперимента алгоритмы, разработанные и достаточно широко известные за рубежом. К таким алгоритмам относятся, в частности, алгоритмы решения задачи N тел – метод Барнеса–Гута [13] и *Fast Multipole Method,* FMM [12].

2. Программный комплекс

В соответствии с целевыми установками рассматриваемой работы и методами, принятыми для их достижения, необходимо было решить следующие задачи:

- формирование требований, отражающих особенности вычислительного эксперимента на основе метода вязких вихревых доменов;
- разработка архитектуры программного комплекса, анализ его свойств;
- разработка эффективных алгоритмов вычисления различных характеристик выбранного численного метода;
- реализация отдельных компонентов программного комплекса для выбранного класса задач обтекания;
- тестирование разработанных компонентов в составе единого комплекса, проведение демонстрационных расчетов.

Представленный список задач определяет последовательность дальнейшего изложения результатов, полученных в ходе выполнения работ.

2.1. Формирование требований

Согласно стандарту ISO 42010:2007 (IEEE 1471) "Рекомендации к описанию архитектуры сложных систем" требования к программной системе определяются средой окружения, в которой должна разрабатываться и функционировать создаваемая система.

Основные характеристики среды окружения целевого программного комплекса определяются особенностями вычислительного эксперимента на основе метода вязких вихревых доменов. Описание рассматриваемого вычислительного эксперимента проведено с использованием известной схемы-триады, предложенной академиком А.А. Самарским, "модель – алгоритм – программа". В качестве уравнений математической модели, описывающих движение сплошной среды, выбраны уравнения Навье–Стокса и неразрывности:

$$\frac{\partial \vec{V}}{\partial t} - \vec{V} \times \vec{\Omega} = -\nabla\left(\frac{p}{\rho} + \frac{\vec{V}^2}{2}\right) + \nu\Delta\vec{V}, \qquad (1)$$
$$\operatorname{div}\vec{V} = 0, \ \vec{\Omega} = \operatorname{rot}\vec{V}, \ \rho = \text{const}, \ \nu = \text{const},$$

в которых используются стандартные обозначения скорости $\vec{V}$, давления p, плотности ρ, завихренности $\vec{\Omega}$ и вязкости ν. Движение твердых тел (при их наличии) описывается классическими уравнениями динамики. Данная модель в ходе эксперимента остается неизменной. Второму элементу триады соответствует численный метод решения уравнений (1), описывающий нестационарные движения вязкой несжимаемой жидкости постоянной плотности в рамках лагранжева подхода [2, 6], – метод вязких вихревых доменов, который выбран в качестве базового.

Метод вязких вихревых доменов допускает ряд модификаций для моделирования различных типов течений вязкой среды (далее также – классов задач обтекания). Выделим следующие основные типы течений.

1. Течения в бесконечном пространстве, заполненном неподвижной на бесконечности средой, с набором погруженных в нее движущихся областей, которые соответствуют профилям обтекаемых тел. Число степеней свободы любого тела может быть равно 0, 1, 2 или 3. Предполагается, что в процессе движения тела не сталкиваются и не выходят за пределы области течения. На границах областей, соответствующих профилям обтекаемых тел, могут быть заданы различные условия (условия непротекания, деформируемости, подвижности стенок). Между областями также могут быть заданы различные дополнительные условия (условия соединения посредством пружины, шарнира). На границах обтекаемых тел, а также в области течения может быть задан набор точек (с законом их движения), соответствующих областям отбора (вдува) среды. Допускаются внутренние поверхности разрыва. Предусматривается возможность внешнего силового воздействия на среду.

2. Внутренние течения в неподвижной замкнутой, ограниченной области произвольной геометрической формы. Такие течения можно рассматривать в качестве моделей течений, получаемых, например, в экспериментах с использованием аэродинамической трубы. Кроме условий прилипания и скольжения на непроницаемых частях границы задаются специальные условия на протекаемых частях: сочетание вихреисточников и вихрестоков. Дополнительно в область течения может быть добавлено множество областей, соответствующих профилям обтекаемых тел. Предположения о характере движения областей, об условиях, заданных между ними и на границах, аналогичны изложенным выше.

Как уже упоминалось выше, традиционно используемое программное обеспечение для рассматриваемого вычислительного эксперимента условно можно разделить на два класса. К первому относятся программы, разрабатываемые "предметниками" – специалистами по вихревым методам. Второй класс программных реализаций представляют комплексы с монолитным вычислительным ядром, которые, как правило, разрабатываются с участием квалифицированных программистов. Анализ программных средств обоих классов показывает несоответствие их характеристик условиям, в которых они используются. Существенным недостатком первого класса программных реализаций является отсутствие явно сформулированных принципов и положений программной инженерии, которые используются при их создании. Основной недостаток второго подхода заключается в том, что он предполагает использование принципа универсальности.

С учетом результатов такого анализа поставлена задача построения программного комплекса на основе нового подхода, лишенного основных недостатков традиционных подходов. Основные требования к программной реализации выглядят следующим образом.

С позиций функциональных требований программный комплекс должен реализовать перечень заданных модификаций метода вязких вихревых доменов для описанного выше класса задач обтекания. Требования к качеству программного комплекса представляют собой уточнение целей настоящей работы. Список требований такого типа включает следующие элементы: правильность (атрибут, относящийся к обеспечению соответствия получаемых результатов и эффектов апробированным результатам); производительность (атрибут, относящийся к времени выполнения функций комплекса на вычислительной машине); изменяемость (атрибут, характеризующий простоту внесения изменений в программный комплекс); удобство эксплуатации (характеристика усилий, которые должен прикладывать пользователь в ходе эксплуатации комплекса).

Сформулируем планируемые способы измерения свойств программной реализации и проверки их соответствия сформулированным требованиям. Функциональные возможности, а также правильность реализации функций будут проверены путем тестирования. Оценка качественных свойств программного комплекса будет осуществлена посредством их сравнения со свойствами программных реализаций, которые получаются в рамках традиционных подходов. В частности, для измерения изменяемости используются следующие метрики: μ_1 – метрика, характеризующая объем повторяющегося исходного кода программного обеспечения в рамках того или иного подхода; метрика Чепина оценки сложности программного обеспечения (*Chapin's software complexity metric*) [16]; метрика Карда и Агрести (*Card and Agresti's metric*) [15]; μ_2 – метрика, характеризующая объем исходного кода, который можно использовать повторно при построении новых программных реализаций. Оценка программного обеспечения с использованием первых трех метрик производится на этапе проектирования, оценка на основе метрики

μ_2 производится после этапа кодирования. Следует отметить, что формулирование требований к качеству и описание методов их оценки проведены с учетом действующих на территории России стандартов качества программного обеспечения ГОСТ 28195 и ГОСТ Р ИСО/МЭК 9126.

2.2. Общая организация программного комплекса

В основу общей организации программного комплекса положены элементы компонентного подхода (*component-based development*) [19]. Такой подход, как правило, применяется для разработки линеек (далее также – семейств) программ в некоторой предметной области с целью уменьшения времени на их создание. Основная идея достижения цели компонентного подхода заключается в выделении компонентов семейства программ и в повторном их использовании. Исследования показали [19, 22], что применение данного подхода к разработке семейств программных систем позволяет снизить временные затраты более чем в два раза. Используются общие характеристики следующих элементов компонентного подхода: библиотеки компонентов повторного использования; архитектуры; модели жизненного цикла и этапов процесса анализа и моделирования предметной области (*domain engineering*).

Представим основные характеристики целевого программного комплекса, разработанного с использованием выбранного подхода. Последовательность программ, которые получаются на каждой итерации цикла вычислительного эксперимента, рассматривается в виде линейки (семейства) программных продуктов. Архитектура семейства программ для вычислительного эксперимента на основе метода вязких вихревых доменов, которая учитывает результаты анализа рассматриваемой предметной области, обладает трехуровневой структурой. На нижнем уровне находятся типы данных, основные характеристики которых определяются набором используемых сущностей в методе вязких вихревых доменов. К таким сущностям относятся вектор, вихрь, обтекаемое тело и др. Средний уровень состоит из модулей. Под модулем понимается программный элемент, реализующий параметризованную функцию, соответствующую обобщенному на некоторый класс задач обтекания подшагу основного шага моделирования по времени. На верхнем уровне расположен контроллер, который осуществляет управление вычислениями (созданием/удалением модулей и типов данных и последовательностью обращения к ним). В основу сценария использования программного комплекса положена модель жизненного цикла разработки программного обеспечения, которая применяется в рамках компонентного подхода.

Архитектура влияет на свойство изменяемости программного комплекса. Ее анализ показал, что верны следующие соотношения:

- $\mu_1^A \gtrsim \mu_1^B \gtrsim \mu_1^C$, где μ_1 – метрика, характеризующая суммарный объем повторяющегося исходного кода для программных реализаций в рамках того или иного подхода к построению программного обеспечения для рассматриваемого вычислительного эксперимента;
- $Q^A \lesssim Q^B \lesssim Q^C$, где Q – метрика Чепина;
- $Z^A \lesssim Z^B \lesssim Z^C$, где Z – метрика Карда и Агрести.

Здесь A обозначает подход "предметников", B – предложенный в настоящей работе подход, C – подход с использованием монолитных вычислительных ядер к организации программного обеспечения для вычислительных экспериментов.

Таким образом, в программном комплексе с представленной архитектурой недостатки традиционных подходов к построению программного обеспечения для вычислительного эксперимента на основе метода вязких вихревых доменов выражены в меньшей степени. Более того, в отличие от традиционных подходов, программный комплекс обладает механизмом управления параметрами, которые соответствуют выбранным метрикам, за счет выбора объемов функций, реализованных в отдельных модулях. Следовательно, предложенная архитектурная организация программного комплекса является более эффективной.

2.3. Детальный дизайн компонентов. Программная реализация комплекса

Согласно стандарту ГОСТ Р ИСО/МЭК 12207–99 детальный дизайн является вторым этапом проектирования программной системы. На этом этапе более точно определяют особенности компонентов программной системы, которые были выделены на этапе архитектурного проектирования. При описании общей архитектуры целевого программного комплекса в предыдущем разделе были выделены типы данных и модули. Далее основное внимание уделено модулям программного комплекса.

Основным побудительным мотивом при проектировании модулей является требование высокой производительности программного комплекса. Согласно методу вязких вихревых доменов расчет течений осуществляется с помощью дискретизации по времени и по пространству. Обтекаемые тела и среда представляются в виде набора дискретных элементов (доменов) – присоединенных и свободных вихрей. Скорость свободных вихрей складывается из конвективной и диффузионной составляющих. Нахождение этих скоростей является наиболее сложной с вычислительной точки зрения частью численной схемы метода.

При подсчете конвективной скорости вихрей возникает задача, которая называется задачей N тел. Ее суть в том, что имеется N частиц, некоторым образом распределенных в двумерном пространстве, и задан закон их попарного взаимодействия. Требуется определить мгновенную скорость каждой частицы. Задача N тел возникает в различных научных областях. В контексте данной работы в качестве закона взаимодействия выступает функция, являющаяся ядром в законе Био–Савара.

Прямой метод решения задачи N тел обладает сложностью $O(N^2)$. Известны более эффективные ме-

тоды решения данной задачи. К таким относятся метод Барнеса–Гута (*Barnes–Hut*) [13], сложность которого O ($N \ln N$), и *Fast Multipole Method* (FMM) [12], имеющий асимптотическую сложность $O(N)$.

Разработана модификация [3] известных алгоритмов, которая позволяет вычислить конвективные скорости свободных вихрей. Оценки сложности алгоритма и оптимальных значений параметров дополнительных структур данных (в частности, дерева для хранения иерархической информации о группах вихрей) получены для модельной задачи. Согласно ее условиям центры свободных вихрей равномерным образом распределены в квадрате $D = [0, 1] \times [0, 1]$.

Разработанный алгоритм обладает следующими свойствами:

- вычислительная сложность алгоритма равна $O(N \ln N)$;
- оптимальная глубина дерева равна $n = [\log_4 ((9\ln 4)/24) N]$.

Для вычисления диффузионной скорости свободных вихрей разработан другой эффективный алгоритм, обладающий меньшей вычислительной сложностью по сравнению с прямым методом расчета. Данный алгоритм основан на использовании свойств закона диффузионного взаимодействия вихрей, в частности, экспоненциального убывания попарного влияния вихрей по мере удаленности друг от друга. Сложность прямого метода вычисления диффузионной скорости составляет $O(N^2 + MN)$. Здесь N – число свободных вихрей в области течения, M – число точек разбиения границы области течения. Сложность разработанного алгоритма вычисления диффузионной скорости составляет $O(N + M)$. Данная оценка также получена для модельной задачи, начальные условия которой описаны выше.

В настоящее время обсуждаемый программный комплекс состоит из 14 модулей. Эти модули позволяют моделировать внутренние течения в замкнутой ограниченной области произвольной геометрической формы. В область течения может быть помещен набор дискретных объектов (вихрей или источников/стоков) для задания потока среды нужной конфигурации, а также для моделирования механизмов отбора/вдува среды. Допускаются внутренние поверхности разрыва, которые являются моделями проницаемых экранов. Общий объем исходного кода разработанных компонентов программного комплекса составляет порядка 10 000 строк на языке C++. Проведена оценка значения метрики μ_2 – процента исходного кода, который можно использовать повторно при разработке программных решений для новых задач обтекания. Показано, что как минимум 26 %, а в среднем 70–80 % исходного кода можно использовать повторно, что подтверждает эффективность представленной организации программного комплекса.

Для всех компонентов программного комплекса разработан специальный код, осуществляющий их привязку к скриптовому языку *Python*. Использование скриптового языка делает эксплуатацию программного комплекса более удобной по сравнению с программным обеспечением, которое традиционно применяется для проведения вычислительного эксперимента.

2.4. Тестирование программного комплекса

С помощью тестирования проверялись: производительность; функциональные возможности; правильность реализации функций программного комплекса. Тестовые испытания, направленные на оценку производительности, включали в себя проверку на специально подобранных задачах влияния языка *Python* на скорость вычислений, а также эффективность алгоритмов вычисления конвективной и диффузионной скорости свободных вихрей, которые были представлены в предыдущем разделе. Тестирование функциональных возможностей и правильности реализации функций программного комплекса проводилось на задаче о течении вязкой несжимаемой жидкости в вихревой ячейке, т.е. в поперечном профилированном вырезе, расположенном на поверхности обтекаемого тела. Рассматривались вихревые ячейки круговой формы на одной из стенок двумерного канала. Задачи такого рода активно исследуются в рамках международного проекта *VortexCell2050* [21]. В рамках тестирования функций программного комплекса оценивались возможности моделирования механизмов отбора/вдува среды и проницаемых экранов с его помощью. Необходимо отметить, что подготовка, проведение тестовых испытаний и интерпретация их результатов выполнялись коллективом исследователей, в состав которого кроме авторов входили сотрудники лаборатории 107 Института механики МГУ им. М.В. Ломоносова, включая заведующего лабораторией С.В. Гувернюка, Г.Я. Дынникову, П.Р. Андронова.

Результаты проведенных испытаний показали, что использование скриптового языка *Python* незначительно сказывается на производительности вычислений. Сравнительная характеристика времени, затрачиваемого на вычисления в ходе соответствующих запусков программного комплекса с интерфейсом на скриптовом языке и без него, представлена в табл. 1. В качестве модельной задачи выбрана задача о течении вязкой жидкости в прямоугольном канале.

Таблица 1

Число шагов по времени	Время работы, с	
	C++	C++/Python
50	197	205
100	500	503

Проведено тестирование [3] представленных в предыдущем разделе алгоритмов вычисления конвективной и диффузионной скорости свободных вихрей. Оно проводилось на модельной задаче, условия которой также изложены выше. Для тестирования модуля вычисления конвективной скорости произведено две

серии запусков. Первая серия была предназначена для проверки критерия оптимальности выбора глубины дерева, вторая – для сравнения быстрого и прямого методов расчета. Проверка критерия оптимальности производилась для $N = 75\,000$ вихрей. Далее представлено время работы данного алгоритма при различной глубине дерева.

Глубина дерева	5	6	7	8	9
Время счета, с	36	10	4	10	41

Как из этого следует, оптимальное значение глубины дерева равно 7. Согласно следствию из утверждения о вычислительной сложности алгоритма нахождения конвективных скоростей свободных вихрей, доказанного в предыдущей главе, оптимальное значение глубины дерева равно $n = [\log_4 ((9\ln 4)/24)75000] = 7$, что подтверждает правильность результатов, полученных аналитически.

В табл. 2 отражены результаты сравнения разработанного быстрого алгоритма (с оптимальным значением глубины дерева) с прямым методом.

Таблица 2

Время работы метода, с	Число вихрей, *N*, тыс			
	10	20	40	80
Прямого	13	51	213	875
Быстрого	1	1	2	5

В табл. 3 даны результаты сравнения алгоритма вычисления диффузионной скорости свободных вихрей с прямым методом.

Таблица 3

Время работы метода, с	Число вихрей, *N*, тыс			
	5	10	20	40
Прямого	14	53	221	864
Быстрого	<1	1	2	3

Из представленных таблиц следует, что предложенные и реализованные в программном комплексе быстрые алгоритмы вычисления конвективной и диффузионной скорости свободных вихрей значительно превосходят по эффективности прямые методы вычисления этих характеристик.

Проведено тестирование функциональных возможностей программного комплекса. Результаты расчетов, полученные с использованием программного комплекса и представленные в работе [1], подтверждают его возможности моделировать механизмы отбора/вдува среды, а также возможности моделировать течения через проницаемые поверхности. Во втором случае рассматривались конфигурации течений, в которых вихревая ячейка накрыта крышкой с различными коэффициентами проницаемости.

Сравнение полученных ранее, в том числе аналитическим путем, характеристик течения вязкой несжимаемой жидкости в вихревой ячейке с результатами тестовых экспериментов подтвердили выполнение требований, которые предъявляются к правильности реализации функций программного комплекса. Результаты такого сравнения подробно представлены в работе [1].

Заключение

В настоящей работе рассмотрены вопросы разработки и реализации теоретически обоснованных подходов к созданию эффективного комплекса программ для проведения вычислительных экспериментов на основе вихревых методов численного моделирования нестационарных течений вязкой несжимаемой жидкости. К основным результатам работы относятся следующие.

- На основе методологии программной инженерии и формальных моделей описания реализованы механизмы компонентного подхода к созданию комплекса программ для подготовки и проведения ресурсоемкого и сложно организованного вычислительного эксперимента на основе вихревых методов.
- Предложены новые архитектурные решения, положенные в основу комплекса программ, предназначенного для проведения вычислительных экспериментов на основе метода вязких вихревых доменов. Показано преимущество предложенных решений перед традиционными подходами к построению программного обеспечения для вычислительного эксперимента на основе вихревых методов.
- Разработаны оригинальные, эффективные алгоритмы расчетов различных характеристик моделируемых процессов, которые определяются рассматриваемым численным методом. Эффективность предложенных алгоритмов подтверждена путем оценки их вычислительной сложности.
- Проведено тестирование программного комплекса. Результаты тестовых испытаний продемонстрировали соответствие характеристик программного комплекса предъявляемым к нему требованиям.

СПИСОК ЛИТЕРАТУРЫ

1. **Андронов П.Р., Гирча А.И., Гувернюк С.В.** О стабилизации уловленного вихря в каверне с помощью проницаемой крышки // Современные проблемы математики и механики. Выпуск, посвященный 70-летию со дня рождения В.А. Садовничего. Под редакцией академика Г.Г. Черного и профессора В.П. Карликова. 2009. № 2.1. С. 33–41.

2. **Андронов П.Р., Гувернюк С.В., Дынникова Г.Я.** Вихревые методы расчета нестационарных гидродинамических нагрузок. М.: Изд-во Моск. ун-та, 2006. 184 с.

3. **Гирча А.И.** Быстрый алгоритм решения "Задачи *N* тел" в рамках программной реализации численного метода вязких вихревых доменов // Системы управления и информационные технологии. 2007. № 4.2(30). С. 239–242.

4. **Григоренко Д.А.** Комплекс программ для реализации семейства вихревых методов и его применение // Автореферат диссертации на соискание ученой степени кандидата физико-математических наук. Москва. 2008.

5. **Гувернюк С.В., Дынникова Г.Я.** Моделирование обтекания колеблющегося профиля методом вязких вихревых доменов // Изв. РАН. МЖГ. 2007. № 1. С. 3–14.

6. **Дынникова Г.Я.** Лагранжев подход к решению нестационарных уравнений Навье–Стокса // ДАН. 2004. Т. 399. № 1. С. 42–46.

7. **Рациональное** численное моделирование в нелинейной механике / Под ред. О.М. Белоцерковского. М.: Наука, 1990.

8. **Самарский А.А.** Математическое моделирование и вычислительный эксперимент // Вестник АН СССР. 1979. № 5. С. 38–49.

9. **Самарский А.А.** Компьютеры, модели, вычислительный эксперимент. Введение в информатику с позиций математического моделирования. М.: Наука, 1988. 176 с.

10. **Баранов П.А., Гувернюк С.В., Ермишин А.В. и др.** Управление обтеканием тел с вихревыми ячейками в приложении к летательным аппаратам интегральной компоновки (численное и физическое моделирование). М.: Изд-во Моск. Ун-та, 2003.

11. **Яненко Н.Н., Карначук В.И., Коновалов А.И.** Проблемы математической технологии // Численные методы механики сплошной среды. 1977. № 8(3).

12. **Ambrosiano J., Greengard L., Rokhlin V.** The fast multipole method for gridless particle simulation // Computer Physics Communications. 1988. № 48(1). P. 117–125.

13. **Barnes J., Hut P.** A hierarchical o (n log n) force-calculation algorithm // Nature. 1986. № 324(4).

14. **Buschmann F., et al.** Pattern-Oriented Software Architecture: A System of Patterns. John Wiley and Sons, 1996.

15. **Card D.N., Agresti W.W.** Measuring software design complexity // Journal of Systems and Software. 1988. № 8. P. 185–197.

16. **Chapin N.** A measure of software complexity // Proc. of the National Computer Conference. 1979. P. 995–1002.

17. **Chorin A.J.** Numerical study of slightly viscous flow // Journal of Fluid Mechanics. 1973. Vol. 4. № 57. P. 785–796.

18. **Ogami Y., Akamatsu T.** Viscous flow simulation using the discrete vortex model – the diffusion velocity method // Computers & Fluids. 1991. Vol. 19. № 3/4. P. 433–441.

19. **Pressman R.** Software Engineering: a practitioner's approach, 5th edition. McGraw-Hill, 2000. 840 p.

20. **Shankar S., van Dommelen L.** A new diffusion procedure for vortex methods // Comp. Phys. 1996. № 127. P. 88–109.

21. **VortexCell2050** project homesite.
http: // www.vortexcell2050.org.

22. **Yourdon E.** Software reuse // Application Development Strategies. 1994. № 6(12). P. 1–16.

Продолжается подписка на журнал "Программная инженерия" на первое полугодие 2011 г.

Оформить подписку можно через подписные Агентства или непосредственно в редакции журнала.

Подписные индексы по каталогам:

"*Роспечать*" – **22765**; "*Пресса России*" – **39795**

Адрес редакции: 107076, Москва, Стромынский пер., д. 4,
редакция журнала **"Программная инженерия"**

Тел.: (499)269-53-97. Факс: (499)269-55-10. E-mail: prin@novtex.ru.

УДК 681.306

В.В. Подбельский, д-р техн. наук, проф., Государственный университет – Высшая школа экономики

E-mail:vpodbelskiy@hse.ru

Эволюционный подход в преподавании программирования

Современная интегрированная среда разработки программ обеспечивает программистов гораздо большими возможностями, нежели были у их предшественников. Наличие этих возможностей позволяет даже начинающим программистам применять в процессе создания программ передовые методики разработки программных средств. Тому, как использовать эти возможности на первых этапах изучения программирования, посвящена эта работа.

Ключевые слова: *обучение программированию, эволюционный подход, модели обучения, методики программирования.*

Центральной проблемой базовой подготовки специалистов по программной инженерии является обучение программированию с применением современных средств автоматизированной разработки программных продуктов.

В настоящее время в практике преподавания программирования в основном применяются следующие два подхода.

В первом из них будущий специалист в учебных курсах проходит путь исторического развития вычислительной техники и программирования. Изложение начинается с архитектуры ЭВМ и языка ассемблера, затем изучаются один или несколько языков высокого уровня (процедурное и объектно-ориентированное программирование), после чего рассматриваются вопросы программирования для современных операционных систем.

При втором подходе вначале изучаются вопросы алгоритмизации с практикумом на наиболее удобном для учебного процесса языке (обычно это Паскаль), затем следуют современные языки и поддерживаемые ими технологии (процедурная, объектно-ориентированная, обобщенная), и только после этого рассматриваются особенности программирования для современных операционных систем с применением средств автоматизации труда программистов.

Независимо от применяемого подхода на первом этапе обучения программированию существует весьма заметное затруднение, связанное с существенно разной подготовленностью студентов. Различия в подготовленности связаны с тем, что до поступления в вуз кто-то закончил среднее учебное заведение со специализацией по информатике и программированию, занимался в кружках при кафедрах и т. д. Другие серьезной подготовки по программированию не получили. Здесь ситуация сходна с положением в изучении иностранных языков – там также возникает проблема необходимости "выравнивания" знаний учащихся.

Если нет возможности разделить учащихся на подгруппы в соответствии с уровнем первичных знаний, то изложение материала приходится начинать "с нуля". Неизбежное при этом изучение азов программирования (или иностранного языка) для "выравнивания" знаний контингента требует временных затрат и "расслабляет" подготовленную часть аудитории. Опытные преподаватели проводят дополнительные занятия для одних и дают более сложные задания для других, но потери времени при этом остаются. Обсуждение существующих подходов к решению этой проблемы выходит за рамки нашей работы. Предполагается, что эта проблема решена и учащиеся имеют приблизительно одинаковую начальную подготовку, т.е. каждый усвоил основы программирования и алгоритмизации (в объеме школьного курса информатики).

В соответствии с учебным планом и программой дисциплины ее преподавание и изучение предусматривают следующие виды работ:

1) лекционный курс по программированию;

2) практические и семинарские занятия, цель которых – освоение средств конкретного языка программирования и механизмов среды разработки программ;

3) аудиторные контрольные работы и тестирование;

4) выполнение контрольных домашних заданий с оформлением отчетной документации;

5) курсовые работы с оформлением отчетной документации и защитой результатов.

Работа преподавателя и работа студента при выполнении предписаний учебного плана имеют разный характер в зависимости от перечисленных выше пунктов. Дидактически пункты 1, 2 укладываются в схему традиционного (репродуктивного) обучения, а пункты 4, 5 (в особенности 5) должны иметь продуктивный (поисковый, частично исследовательский) характер.

Следуя литературе, например [1], используем термин "модель обучения" как обозначение плана осуществления учебного процесса.

Для *репродуктивной (технологической) модели* обучения характерна ориентация на "достижение практически всеми учащимися заданных эталонных результатов на уровне гарантированного минимума на основе организации предъявления стандартизованного контроля и коррекции текущих учебных результатов" [1]. Предусмотренный этой моделью контроль выполняется в пункте 3.

Поисковая (продуктивная) модель обучения (точнее, ее частный случай – исследовательская модель, которую мы будем применять) предусматривает "формирование у учащихся опыта самостоятельного поиска новых знаний, их применения в новых условиях, формирования опыта творческой деятельности" [1].

Не следует считать, что роль обучаемого в репродуктивной модели пассивна. Для прочного усвоения лекционного материала необходимо дополнительно изучать литературные источники, которых в настоящее время в области программирования достаточно (даже слишком) много. Существуют электронные и печатные пособия и справочники, как по интегрированным средам разработки, так и по языкам программирования. Доступны пособия по методологиям программирования и анализа предметных областей и по дисциплинам проектирования программных продуктов. При работе на семинарах и практических занятиях студент изучает ремесло программирования. Обычно ему предоставляются образцы, шаблоны и эталоны программ, на основе которых студент разрабатывает собственные приложения. Таким образом, первый этап обучения ведется на примерах, эталонах, образцах и шаблонах. Код таких эталонных приложений обычно 50...100 операторов на языке программирования высокого уровня. Самостоятельная работа предусматривается в основном в виде домашних заданий, выполняемых по разобранным в дисплейном классе образцам. Такой подготовки студенту достаточно для самостоятельного программирования уже готовых алгоритмов и участия в качестве кодировщика в разработке больших проектов.

Однако при переходе к пунктам 4 и 5 изменяется модель обучения, появляются нестандартные задачи. Для их решения нужен не только лекционный материал, шаблоны и эталоны, не только кодирование, но и самостоятельно выполненный студентом этап проектирования. Вместе с тем учебные дисциплины "Технологии программирования", "Объектно-ориентированный анализ" и подобные, в которых изучаются методы, приемы, средства проектирования программных продуктов, преподаются не на первом курсе, а позже. Поэтому, приступая к самостоятельной разработке программного продукта в рамках курсового проекта или даже контрольного домашнего задания, студент-первокурсник сталкивается со следующими затруднениями:

- недостаточно хорошие знания среды разработки и ее возможностей;
- непрочные знания нужного языка программирования;
- отсутствие навыков проектирования программных продуктов.

В связи с перечисленными затруднениями преподаватель должен помочь студенту: обеспечить его методикой, позволяющей студенту (начинающему программисту) не только преодолеть названные трудности, но в процессе выполнения работы получить новые знания и навыки. Другими словами, преподаватель должен помочь студенту занять в учебном процессе активную позицию, при которой студент должен понимать возникающие в работе трудности, уметь отыскивать способы их преодоления, анализировать их эффективность и принимать проектные решения на основе выполненного анализа.

Таким образом, преподаватель должен предложить наиболее рациональную методологию, которую без особой специальной подготовки могли бы применять студенты, переходя от простых учебных программ к небольшому самостоятельному проекту, связанному с разработкой завершенного программного продукта.

В сформулированном требовании использованы три прилагательных: "небольшой", "самостоятельный" и "завершенный". Поясним их смысл.

Самостоятельность предусматривает выполнение индивидуального задания, в рамках требований которого необходимо самостоятельно выбрать архитектурные (проектные) решения и особенности реализации.

Термин "небольшой" предполагает, что проект должен быть завершен в два-три месяца и трудоемкость разработки (в часах) не должна превышать запланированной по учебному плану.

Завершенность проекта предполагает, что созданный в рамках проекта программный продукт должен быть работоспособным, и эта работоспособность должна быть подтверждена проведением достаточно полного тестирования. (Тот факт, что проект должен быть снабжен программной документацией, важен, но в нашей работе не затрагивается.)

Понимая, что нужно студенту, обратим внимание на существующие методологии программирования. Как отметил Э. Гамма [4], "у всех методик программирования одна цель – создание программного продукта с заданной функциональностью к заданному сроку". Достижение этой цели может быть выполнено разными способами. Созданию и совершенствованию методик программирования посвящены многочисленные работы.

Первые методологии программирования ведут свои родословные от методологий, созданных в таких предметных областях, как строительство и машиностроение. Их особенность – тщательное проектирование, завершающееся выпуском проектной документации, и только потом – производство (в нашем случае кодирование). В этих методологиях предполагается, что после завершения проектной стадии реализацию могут выполнять люди более низкой квалификации. Но для наших целей это никак не подходит – и архитектор (проектировщик), и исполнитель (кодировщик) в учебном проекте – одно лицо. И, самое важное, обе эти роли необходимо играть человеку, не имеющему достаточной квалификации ни в проектировании, ни в кодировании. Кроме того, громоздкость и малая эффективность таких методологий в программировании уже неоднократно доказаны, например в работе [2].

Современные методологии программирования не требуют строгого разграничения этапов проектирования и реализации. В них всегда предусматривается итеративность, проект развивается поэтапно, и решения, принятые на очередном этапе, могут существенно влиять на последующие этапы проектирования. На первых этапах создаются версии программного продукта, обладающие минимальной функциональностью, и эта функциональность нарастает при дальнейшей разработке. Такая итеративность процесса и постепенное увеличение функциональности проектируемого продукта вполне подходят для целей учебного процесса, построенного на основе продукционной модели. Однако в полной мере промышленные методологии программирования применимы в тех случаях, когда ими пользуются достаточно квалифицированные проектировщики и программисты. А такую квалификацию нельзя требовать от начинающего программиста, которым является студент первого курса. Для учебного процесса нужна упрощенная методология, не требующая специального длительного изучения.

Возможность предложить такую методологию и применить ее в учебных целях основана на потенциале современных интегрированных сред разработки программных продуктов. Но даже при использовании в обучении современных инструментальных средств разработки нет необходимости изобретать методологию с нуля. В наибольшей степени для использования в учебном процессе по нашему мнению пригодна методология, получившая название "экстремальное программирование" [4, 5]. Она относится к облегченным или гибким технологиям, основная цель которых – "достичь необходимого компромисса между перегруженным процессом проектирования и полным его отсутствием" [2]. Экстремальное программирование предлагает строить процесс создания программного продукта как итерационный и эволюционный, в котором проектирование неразрывно связано с реализацией и тестированием. На каждой итерации принимается небольшое число проектных решений, которые полностью реализуются в виде кода. Завершает каждую итерацию процедура тестирования, результаты которой служат основой для принятия проектных решений следующей итерации. Новая итерация расширяет функциональные возможности создаваемого программного продукта. При этом в ряде случаев может потребоваться реорганизация уже отлаженного кода предыдущей итерации. Дело в том, что принятые ранее проектные решения могут влиять на последующие шаги проектирования, и иногда это влияние препятствует дальнейшему развитию проекта. Самое главное – каждая итерация завершается работающим вариантом создаваемого продукта, и оценки его особенностей являются фундаментом для принятия проектных решений на следующей итерации.

Однако непосредственное применение методологии экстремального программирования (да и любой другой промышленной методологии) на первом этапе обучения программированию практически невозможно. Не все подходит начинающему программисту, который зачастую испытывает затруднения в правильном выборе или применении возможностей стандартной библиотеки, предлагаемой ему средой разработки. Более того, на первых порах затруднения могут возникать в локализации ошибок кодирования и в понимании диагностических сообщений, получаемых на этапе компиляции и/или выполнения программы. В последних случаях может потребоваться изучение технической документации и применения справочных средств интегрированной среды разработки. Еще большие затруднения возникают в тех случаях, когда программист (он одновременно выступает и в роли архитектора программного продукта), приступая к разработке, не знает, какие алгоритмические решения потребуются для реализации функциональных требований, сформулированных в техническом задании на программный продукт. Выбор математических методов и алгоритмов может зависеть от общей архитектуры программного продукта, которая не определена до начала разработки. Также возможна и обратная зависимость.

При переносе методологии экстремального программирования в учебный процесс важным вопросом является длительность отдельной итерации и ее смысловое содержание. Если в промышленном применении основной целью следующей итерации является добавление функциональности, то в учебном применении некоторые итерации служат "познавательным целям". Например, студент не умеет организовать обмены между экранными формами одного приложения или не знает, как определить размер коллекции, пред-

ставляемой индексатором класса. А указанные вопросы ему необходимо решить для того, чтобы на очередной итерации добавить новую функциональность к уже имеющемуся варианту программного продукта. В этом случае назначение промежуточной итерации – получить и проверить на практике новые знания. Таким образом, традиционные для экстремального программирования итерации в учебном процессе должны перемежаться познавательными (исследовательскими) итерациями, направленными на изучение программных и инструментальных средств, которыми пользуется студент при выполнении курсового проекта. Кстати говоря, современные интегрированные среды разработки программных продуктов, например [6], включают средства для синтаксического контроля правильности программного кода и тем самым частично автоматизируют процесс выполнения познавательных итераций.

Следуя прогрессивным методикам разработки программных средств и учитывая рассмотренные требования учебного процесса, сформулируем рекомендации по выполнению разработки программного продукта.

1. Разработка должна выполняться по этапам (шагам), каждый из которых включает: проектирование (в том числе проектирование элементов визуального интерфейса); кодирование (включая синтаксическую и семантическую отладку кода) и тестирование.

2. Этапы разработки неформально делятся на два вида: исследовательские этапы (познавательные) и этапы разработки, добавляющие создаваемому продукту новую функциональность в соответствии с техническим заданием (с условием задачи).

3. Каждый (даже первый) этап независимо от его вида предусматривает создание работающего программного продукта. (На первом этапе создается программа с минимальной функциональностью.)

4. В конце каждого этапа разработки выполняется анализ существующего (работоспособного) варианта программного продукта. Цель анализа – оценка возможности добавления средств (конструкций) для реализации новых, дополнительных требований, предусмотренных техническим заданием.

5. На основе результатов анализа либо выполняется реорганизация существующего варианта, либо непосредственное принятие проектных решений, кодирование и тестирование (в соответствии с пунктом 1).

6. После завершения каждого этапа разработки продукт, оставаясь работоспособным, приобретает новую (зачастую дополнительную) функциональность либо изменяет уже имеющуюся функциональность, приближаясь к требованиям технического задания.

Таким образом, продукт постоянно находится в работоспособном состоянии и в него постоянно вносятся изменения. Другими словами, программный продукт от шага к шагу, независимо от вида шагов (этап разработки или познавательный), эволюционирует и его функциональность растет от минимальной до желаемой (указанной в техническом задании).

1	0	1	1	0	0	0	1	0	1	0	0
1	1	1	0	1	1	0	0	1	0	1	1
1	0	1	0	0	0	0	1	0	0	1	0
0	1	0	1	0	1	1	0	1	0	1	0
0	1	1	0	0	1	1	0	1	0	0	1
1	0	1	1	1	1	1	0	0	1	0	0
1	0	1	1	1	0	1	1	1	0	1	0
1	0	0	0	0	0	1	1	1	0	0	0

Рис. 1. Кластер ячеек с нулевыми значениями (иллюстрация задания)

Иллюстрированный пример: Покажем, как могут быть реализованы предложенные рекомендации на конкретном примере.

Задача. Определите матрицу с размерами 12 на 12, элементы которой принимают случайные значения 0 или 1. Визуализируйте матрицу. Выделяя (например, мышью) конкретный элемент на изображении матрицы, "закрасьте" все смежные элементы, имеющие в матрице те же значения, что и выделенный (начальный). Смежными считать все элементы, имеющие на изображении общую сторону и примыкающие к нему непосредственно и опосредовано (через промежуточные элементы). Пример требуемого результата выполнения программы приведен на рис. 1. Будем называть совокупность выделенных элементов кластером. Таким образом, кластер – это совокупность элементов матрицы, имеющих общие стороны и одинаковые значения.

Анализируя постановку задачи, отмечаем, что для ее решения необходимо сконструировать средства диалога (интерфейс программы) и разработать алгоритмы, обеспечивающие нужную функциональность программы.

Шаг 1. Проектирование формы

Следуя предложенной схеме эволюционного построения, начнем с шага, предполагающего создание программы с минимальной функциональностью. Как было оговорено ранее, каждый шаг должен включать три действия: проектирование, кодирование и тестирование. Проектирование на первом шаге решения нашей задачи сводится к выбору элементов интерфейса. В качестве основы выбираем стандартное окно Windows-приложения (форму с заголовком "Выделение кластеров"). Для размещения приглашения пользователю введем элемент **Lablel**, разместив на нем текст "Выделите ячейку:". Для изображения матрицы используем элемент **DataGridView**. В качестве еще одного средства диалога с пользователем введем кнопку (элемент **Button**) с надписью "Построить кластер". Отметим, что на первом шаге не предполагается обработки события "нажатие на кнопку", поэтому добавление элемента **Button** можно было бы отложить на второй шаг разработки. Однако в нашей несложной задаче интерфейс очень прост и визуальное конструирование

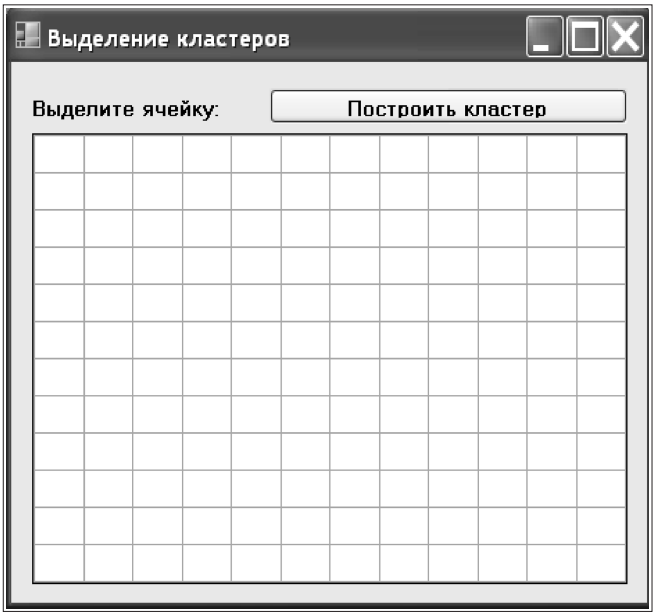

Рис. 2. Результат выполнения программы на первом шаге

формы может быть выполнено уже сейчас. Итак, минимальной функциональностью создаваемого программного продукта может быть форма с тремя элементами: **Lablel, DataGridView, Button.**

Поместив на форму указанные элементы, необходимо явно задать значения некоторых из свойств.

Form1: Text = Выделение кластеров
Font.Bold = True
StartPosition = CenterScreen
label1 Text = Выделите ячейку:
Font.Bold = True
button1: Text = Построить кластер
Font.Bold = True
DataGridView1:
ColumnHeadersVisible = False – убрать заголовки столбцов;
RowHeadersVisible = False – убрать названия строк;
AutoSizeColumnMode = Fill – "растянуть" строки по ширине элемента.

Кодирование: для достижения минимальной функциональности достаточно поместить в код класса Form1.cs следующий обработчик события Form1_Load.

```
using System.Windows.Forms;
namespace Clasters {
public partial class Form1 : Form {
   int M=12, N=12;  // размеры матрицы:
//M - строки, N - столбцы.
     public Form1() {
     InitializeComponent( );
     }
private void Form1_Load(object sender,
EventArgs e) {
   dataGridView1.ColumnCount = N;
   dataGridView1.RowCount = M;
   dataGridView1.Rows[0].Cells[0].Selected =
= false; // снять выделение
      }
   }
}
```

Тестирование на первом шаге сводится к трансляции и выполнению программы. Результат выполнения программы в конце первого шага (форма на экране) показан на рис 2. По результатам автор может, возвращаясь к визуальному проектированию формы, подравнять размеры ячеек, изменить взаимное расположение элементов и т.д. Кроме того, следует проверить возможность выделения мышью элементов формы.

Шаг 2. Добавление класса матриц со случайными (0 или 1) значениями элементов

Присвоим классу имя **RandomMatrix.**

```
namespace Clasters {
class RandomMatrix {
   int mm, nn; // размеры матрицы
public sbyte[,] matrix;    // ссылка на
//массив элементов матрицы
public RandomMatrix(int m, int n) {
// конструктор
   mm=m; nn=n;
   matrix = new sbyte[m, n];
   Random gen = new Random();
   for (int i = 0; i < m; i++)
      for(int j = 0; j < n; j++)
         matrix[i,j] = (sbyte)gen.Next(2);
  }  // конструктор
  }
  }
```

В **Form1** добавить ссылку на объект:

RandomMatrix `matr; // матрица со случайными элементами 0, 1.`

В конструктор формы **Form1()** добавить:

`matr` = new RandomMatrix`(M, N);`

В обработчик событий **Form1_Load()** добавить:

```
// в таблицу значения элементов матрицы:
for (int i = 0; i < M; i++)
 for (int j = 0; j < N; j++);
  dataGridView1[j, i].Value =   // первый
// индекс - столбец!
matr.matrix[i,j] == 0 ? "0" : "1";
```

Результат выполнения программы на шаге 2 показан на рис. 3.

Шаг 3. Добавление класса Position, объект которого представляет координаты элемента матрицы и ячейки сетки

```
namespace Clasters {
  class Position {
public int x, y;
    public  Position(int  xi,  int  yi)
{x=xi; y = yi;}
    public  Position(  Position  p)  {x  =
p.x; y = p.y;}
  }
 }
```

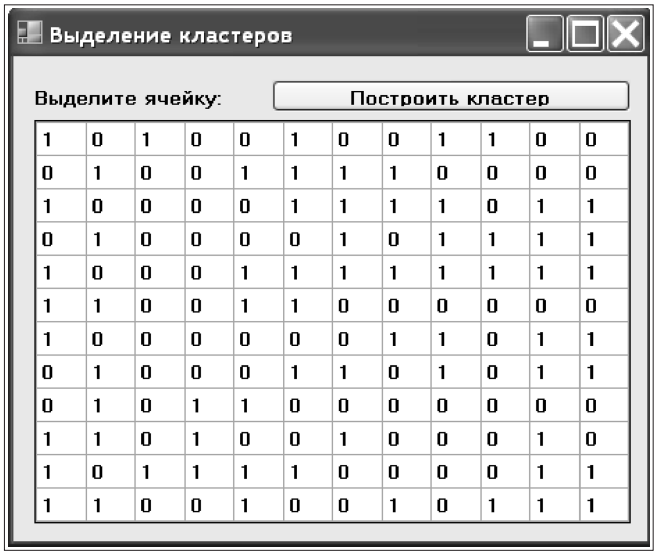

Рис. 3. Результат выполнения программы на шаге 2

В код формы добавить:

```
  Position pos;  // координаты выделенной
//ячейки
```

Обработчик события **button1_Click():**

```
private void button1_Click(object sender,
EventArgs e) {
//.. Ищем выделенную ячейку (K - коли-
// чество выделенных):
  int K=0;
  for (int i = 0; i < M; i++)
  for (int j = 0; j < N; j++)
    if (dataGridView1.Rows[i].Cells[j].
Selected == true) {
    pos = new Position(j,i); K++;
    }
  if (K == 0) { MessageBox.Show("Нет вы-
деленной ячейки!");
  return;
  }
if (K > 1) { MessageBox.Show("Выделите
одну ячейку!");
    for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
    dataGridView1.Rows[i].Cells[j].Selected
= false;
    return;
  }
else
    MessageBox.Show("x="+pos.x+"
y="+pos.y);
}  // button1_Click()
```

При выполнении программы возможны три варианта результата (рис. 4, см. четвертую сторону обложки).

Шаг 4. Добавление в класс RandomMatrix метода region() для формирования массива смежных ячеек

```
public Position[] region(Position p) {
// ближайшие точки
  Position [] res = null;
  if (p.x > 0 & p.x < nn-1 & p.y > 0 & p.y
< mm-1)
  { res = new Position[5];
  res[0] = new Position(p);
  res[1] = new Position(p.x+1, p.y);
  res[2] = new Position(p.x, p.y-1);
  res[3] = new Position(p.x-1, p.y);
  res[4] = new Position(p.x, p.y+1);
  }  //***************
  if (p.x == 0 & p.y ==0) {res = new
Position[3]
    {new Position(p), new Position(p.x+1,
p.y), new Position(p.x, p.y+1)};
  }
  if (p.x == nn-1 & p.y ==0) {res = new
Position[3]
    {new Position(p), new Position(p.x-1,
p.y), new Position(p.x, p.y+1)};
  }
  if (p.x == nn-1 & p.y == mm-1) {res =
new Position[3]
  {new  Position(p),  new  Position(p.x-1,
p.y), new Position(p.x, p.y-1)};
  }
  if (p.x == 0 & p.y == mm-1) {res = new
Position[3]
    {new Position(p), new Position(p.x+1,
p.y), new Position(p.x, p.y-1)};
  }  //*****************
  if (p.x == 0 & p.y != 0 & p.y != mm-1)
{res = new Position[4]
  {new  Position(p),  new  Position(p.x+1,
p.y),
  new    Position(p.x,    p.y+1),    new
Position(p.x, p.y-1)};
  }
  if (p.x == nn - 1 & p.y != 0 & p.y !=
mm-1) {res = new Position[4]
  {new  Position(p),  new  Position(p.x-1,
p.y),
  new    Position(p.x,    p.y+1),    new
Position(p.x, p.y-1)};
  }
  if (p.y == 0 & p.x != nn-1 & p.x != 0)
{res = new Position[4]
  {new  Position(p),  new  Position(p.x+1,
p.y),
  new    Position(p.x-1,    p.y),    new
Position(p.x, p.y+1)};
  }
  if (p.y == mm-1 & p.x != nn-1 & p.x !=
0) {res = new Position[4]
```

```
  {new Position(p), new Position(p.x+1,
p.y),
  new    Position(p.x-1,    p.y),    new
Position(p.x, p.y-1)};
  }
  return res;// Здесь будут добавления
  }  // region()
```

В обработчик события **button1_Click()** вносим дополнение:

```
  else {
  MessageBox.Show("x="+pos.x+"
y="+pos.y);
  foreach      (Position      s      in
matr.region(pos))
  dataGridView1.Rows[s.y].Cells[s.x].
Selected = true;
  }
```

Результаты выполнения программы на шаге 4 представлены на рис. 5 (см. четвертую сторону обложки).

Шаг 5. Дополнения метода region() операторами для удаления элементов, указывающих на элементы матрицы, значения которых отличны от помеченного элемента

Вместо оператора **return res**; помещаем:

```
// Проверка найденных позиций:
  int r=0;
  for(int k=0; k < res.Length; k++) {
    if (matrix[res[0].y, res[0].x] ==
                  matrix[res[k].y,
res[k].x]) r++;}
  Position [] temp = new Position[r];
  for(int k=0, j=0; k < res.Length; k++)
    if  (matrix[res[0].y,  res[0].x]  ==
matrix[res[k].y, res[k].x])
      temp[j++] = res[k];
  return temp;
}  // region()
```

Результаты выполнения программы на шаге 5 показаны на рис. 6. (см. четвертую сторону обложки).

Шаг 6. Дополнение программы средствами для выделения кластера, т.е. построения и окрашивания всей совокупности одинаковых смежных ячеек, прилегающих прямо и косвенно (через соседей) к начальной (выделенной)

На этом шаге придется заниматься разработкой алгоритма построения кластера. Для этого необходимо проанализировать соседей каждого из соседей начального элемента. Так как у каждого из анализируемых элементов в свою очередь могут оказаться подходящие смежные элементы и т.д., то размер кластера заранее предсказать нельзя. Выбор элементов, принадлежащих кластеру, это типичная комбинаторная задача, в общем виде сводящаяся к требованию: "выделите все такие, и только такие элементы множества, которые удовлетворяют заданному условию".

Применим для решения нашей задачи один из вариантов основного метода комбинаторных вычислений – метода ветвей и границ, который еще называют методом перебора с возвратом. Основных подходов к реализации метода ветвей и границ два: применение стека, сохраняющего выбранные, но еще не включенные в результат элементы, и применение рекурсивного алгоритма. Остановимся на рекурсивном подходе.

Пусть P – позиция, определяющая элемент матрицы, для которого выделяется множество соседей, удовлетворяющих требованию принадлежности кластеру.

Пусть R – множество позиций, уже включенных в кластер.

Пусть T – множество соседей элемента P, удовлетворяющих требованию принадлежности кластеру, но еще не включенных в R и в предшествующие множества T.

Пусть $F(P, R)$ – рекурсивная процедура построения кластера R из позиции P.

Псевдокод процедуры можно представить так:

$F(P, R)$ {
Поместить позицию P в кластер R
Определить $T(P)$ – множество соседей элемента P
Пометить P и все P_i из T как уже выбранные
Цикл по всем P_i из T
    Вызов $F(P_i, R)$
Выход из $F()$
}

До обращения к процедуре P – задано, R – пусто. После выхода из процедуры $F(P, R)$ в R – множество позиций, включенных в кластер. Процедура имеет одну особенность – в ней должна быть предусмотрена защита от повторного включения в очередное множество соседей уже выбранных ранее элементов. Поэтому при определении $T(P)$ нужно выбирать во множество соседей элементы, соответствующие кластеру, но еще не включенные в рассмотрение (не помеченные ранее). Для выбора множества соседей, пригодных для включения в кластер, можно было бы применить разработанный нами метод **RandomMatrix.region()**. Однако в нем не предусмотрена оценка "помеченности" элементов. Помечать уже использованные элементы можно разными способами. С целью максимального использования без каких-либо изменений уже закодированных фрагментов нашей программы помечать выбранные элементы будем, изменяя значения элементов матрицы, на которой решается наша задача. Тем самым появляется возможность применить для получения множества $T(P)$ метод **RandomMatrix.region()**. По условию задачи элементы матрицы имеют значения 0 и 1, метод **region()** выбирает соседние элементы, имеющие то же значение, что и начальный. Если изменить значения уже использованных элементов матрицы, то они не будут считаться подходящими для кластера, и метод **region()** "проигнорирует" их присутствие по соседству с начальным элементом.

Техническое решение: будем помечать рассмотренный элемент, увеличивая соответствующее значение элемента матрицы на 2. Не останавливаясь на других деталях программной реализации алгоритма, включим в класс **RandomMatrix** следующий рекурсивный метод для построения кластера:

```
public void spot(Position p, ref
Position[] res) {
  int size = res.Length;
  Position [] temp = res;
  res = new Position [size+1];
  temp.CopyTo(res,0);
  res[size] = p;
  Position [] path = region(p);
  foreach (Position g in path)
    matrix[g.y, g.x] += 2;  // Помечаем
//выделенные элементы
  for (int i=1; i<path.Length; i++)
      {
    matrix[path[i].y, path[i].x] -= 2;
    spot(path[i], ref res);
  }
  return;
}  // spot()
```

Как сказано, в методе уже рассмотренные элементы специальным образом помечаются. Для этого соответствующие элементы матрицы увеличиваются на 2. После этого они становятся "не похожими" на вновь подключаемые к кластеру элементы. Тем самым исключается дублирование и возможное зацикливание. Однако для продолжения поиска новый начальный элемент приводится в исходное состояние (элемент матрицы уменьшается на 2).

Принудительное "окрашивание" элементов, уже включенных в кластер, требует их восстановления для возможности повторных построений того же кластера (возможно из другого начального элемента). Это приходится явно выполнять в коде:

В обработчик события **button1_Click()** вносим следующие изменения:

```
else  {  MessageBox.Show("x="+pos.x+"
y="+pos.y);
  Position [] cla = new Position[0];
  matr.spot(pos, ref cla);
  foreach (Position s in cla) {
    dataGridView1.Rows[s.y].Cells[s.x].
Selected = true;
    matr.matrix[s.y, s.x] -=2;
    }  // foreach
  }  // else
```

Этот код должен заменить следующий, исключаемый из метода, фрагмент:

```
  else {
    MessageBox.Show("x="+pos.x+"
y="+pos.y);
    foreach (Position s in
matr.region(pos))
      dataGridView1.Rows[s.y].Cells[s.x].
Selected = true;
    }
```

Самое главное в коде – замена обращения к методу **matr.region(pos)** на вызов рекурсивного метода **matr.spot(pos, ref cla)**. Различия между методами и пометка рассмотренных элементов определили особенности нового кода.

Результаты выполнения программы на шаге 6 представлены на рис. 7 (см. четвертую сторону обложки).

Заключение

В работе предложена модель обучения программированию на основе выполняемой учащимся в интегрированной среде разработки самостоятельной проектно-исследовательской работы. Тактическая цель такого подхода – поставить учащегося в позицию исследователя, которая требует не только активного применения репродуктивно полученных знаний, но и самостоятельного изучения специфических инструментов и средств, необходимых для решения именно той конкретной проблемы, которая поставлена перед ним в качестве задачи.

СПИСОК ЛИТЕРАТУРЫ

1. **Кларин М.В.** Инновации в обучении: метафоры и модели. Анализ зарубежного опыта. М.: Наука, 1997. 223 с.

2. **Фаулер М.** Рефакторинг: улучшение существующего кода = Refactoring: Improving the Design of Existing Code (2000). СПб: Символ-Плюс, 2004. 430 с.

3. **Вирт Н.** Алгоритмы + структуры данных = программы. Пер. с англ. М.: Мир, 1985. 406 с.

4. **Бек К.** Экстремальное программирование. Пер. с англ. СПб.: Питер, 2002. 224 с.

5. **Бек К.** Экстремальное программирование: разработка через тестирование. Пер. с англ. СПб.: Питер, 2003. 224 с.

6. **Пауэлс Л., Снелл М.** Microsoft Visual Studio 2008. Пер. с англ. СПб.: БХВ-Петербург, 2009. 1200 с.

УДК 004.4′22(07)

А.В. Жарковский, канд. техн. наук, стар. науч. сотр., **А.А. Лямкин,** канд. техн. наук, доц., **Т.Ф. Тревгода,** канд. техн. наук, стар. науч. сотр., Санкт-Петербургский государственный электротехнический университет "ЛЭТИ"

E-mail: tat.trevgoda@yandex.ru

Автоматизация тестирования функционального программного обеспечения комплексов управления сложными техническими системами

Предлагается структура комплекса автоматизированного тестирования и приводятся алгоритмы комплексной отладки и испытаний функционального программного обеспечения комплексов управления сложными техническими системами.

Ключевые слова: *техническая система, функциональное программное обеспечение, тестирование, формуляр, модель, структура, алгоритм*

Сложные технические системы (СТС) морского, наземного и воздушного базирования, иногда называемые подвижными объектами, имеют весьма широкое распространение и отличаются большим разнообразием [1]. Их эффективность зависит главным образом от качества функционального (боевого, бортового) программного обеспечения (ФПО). Самыми длительными и дорогостоящими этапами его создания являются комплексная отладка и испытания. Видимо повысить качество и снизить затраты на создание ФПО можно, как и в других областях, лишь за счет автоматизации этих процессов.

Основным методом, используемым при отладке и при испытаниях программных средств, является метод тестирования [2]. Он предполагает формирование входного тестового набора данных (ТНД), который подается на вход ФПО и некоторого эталона. Затем решение (реакция, выход) ФПО сравнивается с эталонным решением. Несовпадение решений не только говорит о наличии ошибки, но и позволяет судить даже о месте ее возникновения. Разница между процессами отладки и испытаний состоит в том, что отладка направлена на поиск и устранение ошибок, а испытания – на поиск соответствия ФПО техническому заданию (ТЗ) на его создание. Сложность и обусловленная этим трудоемкость процесса тестирования связаны не только с поиском ошибок и места их возникновения, но и с формированием входных и выходных тестов, с вычислением показателей качества ФПО.

ФПО комплексов управления

За исключением транспортных средств все ТС одной стороны (одной государственной принадлежности) предназначены для огневого или радиоэлектронного воздействия на СТС другой стороны [1]. ФПО комплекса управления СТС (рис. 1) на основе информации от различных средств технического зрения (Z) вырабатывает данные для управления различными средствами воздействия (V), например, данные об их пространственном положении для воздействия на выбранную цель, данные о начале и режимах работы средств и т.п.

При управлении СТС независимо от ее особенностей решаются, как минимум, четыре задачи, реализуемые с помощью отдельных, последовательно выполняемых программ ФПО:

- P – преобразование информации и вычисление характеристик движения объектов внешней среды (СТС противника);
- K – классификация объектов;
- R – ранжирование (упорядочивание) объектов по степени важности или степени опасности;

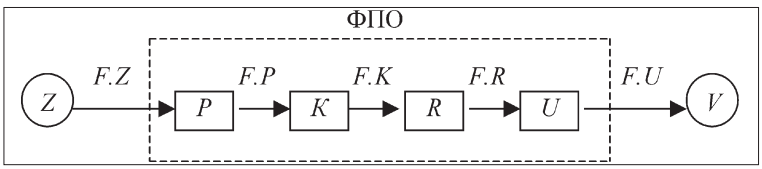

Рис. 1. Структура и связи ФПО

• U – целераспределение–целеуказание (указание положения средств сопровождения и наведения).

Внешние и внутренние (между отдельными программами) связи ФПО могут быть описаны с использованием объектно-признакового языка [3] в виде формуляров обмена ($F.$) с расширением по имени устройства или программы, из которых они выходят.

Так, формуляр средства технического зрения имеет вид:

$$F.Z\,[LC,\ N\_C,\ D,\ E,\ M,\ S],$$

где LC – число целей (объектов); N_C – порядковый номер объекта в нумерации данного средства технического зрения; D – наклонная дальность до объекта; E – курсовой угол объекта; M – угол места объекта; S – скорость объекта.

Формуляр программы преобразования имеет вид:

$$F.P\,[LC,\ N\_C,\ DG,\ E,\ TP,\ PS,\ PH],$$

где DG – горизонтальная дальность; TP – подлетное время объекта к заданной точке (области); PS – признак скорости; PH – признак высоты.

Формуляр программы классификации имеет вид:

$$F.K[LC,\ N\_C,\ TC],$$

где TC – тип цели.

Формуляр ранжирования –

$$F.R[LC,\ N\_C,\ RC],$$

где RC – ранг цели.

Формуляр программы указания может быть описан как

$$F.U\,[EZ,\ EY,\ RR,\ TN],$$

где EZ и EY – углы ориентации средства воздействия в горизонтальной и вертикальной плоскостях; RR – режим работы средства и TN – начало работы средства (начало стрельбы, начало постановки помех).

В приведенных формулярах содержатся значения лишь тех переменных, которые вычисляются указанными программами с привязкой к числу и номеру объектов и которые используются в других программах. Но это не означает, что всякая последующая программа использует только выходную информацию предыдущей программы. Она может использовать выходную информацию любых предшествующих по времени выполнения программ.

Автоматизация тестирования ФПО

Отдельные программы отлаживаются исполнителями на основе частных технических заданий на их разработку, а комплексной отладке и испытаниям подвергается ФПО в целом. Ошибки ФПО обусловлены, главным образом, несогласованностью частных ТЗ на отдельные программы и разным пониманием Заказчиком и Исполнителем функций ФПО вследствие вербальной формулировки общего ТЗ на его создание. Это выясняется в процессе испытаний. При несоответствии ФПО техническому заданию (неправильная работа, отличие реальных количественных показателей от требуемых и др.) следует доработка (доводка) ФПО, заключающаяся в устранении найденных ошибок, проверочной отладке и новом испытании. Это длительный путь и он нередко приводит к провалу всего проекта.

При испытаниях (и при отладке тоже) ФПО тестируется в соответствии с представлениями Заказчика о том, как оно должно реагировать на ту или иную складывающуюся тактическую ситуацию. При этом Заказчик обычно оперирует с тактическими ситуациями, которые описываются в неподвижной декартовой системе координат, а входной тест должен содержать данные в связанной сферической системе координат, так как именно в таком виде они поступают на вход ФПО от средств технического зрения.

Ручное формирование входных ТНД в декартовой, а тем более в сферической системе координат требует больших трудозатрат. Вместе с тем очевидно, что перевод данных об объектах противника из одной системы координат в другую может выполняться автоматически. Труднее обстоит дело с первичным вводом данных о тактической ситуации. Однако если можно ограничиться только типовыми тактическими ситуациями, а в большинстве случаев это именно так, то можно вводить лишь параметры тактической ситуации [1]. При этом число необходимых данных значительно сокращается и измеряется единицами.

Выходной тест формируется эталоном. Обычно в качестве эталона выступает человек (эксперт в данной области). Однако эксперт может судить о правильности выработанных ФПО решений лишь в самых простых случаях, которые могут быть весьма далеки от реальных ситуаций принятия решений по управлению теми или иными средствами воздействия. Здесь следует обратить внимание на то, что на стадии концептуального проектирования всегда создается модель функционирования ПО, на которой отрабатываются правила боевого использования разнообразных средств и оценивается эффективность СТС в различных тактических ситуациях. Это означает, что частью такой модели неизбежно должна выступать модель будущего реального ФПО. Именно эта модель и может быть использована в качестве эталона, как замена эксперта.

При испытаниях помимо фиксации ошибок проводится количественная оценка показателей качества функционирования ФПО. Определение расхождений в решениях реального ФПО и его модели, поиск места возникновения ошибки и расчет показателей качества можно автоматизировать. Для этих целей, равно как и для формирования входных и выходных тестов, необходимо использовать отдельную технологическую ЭВМ, так как эти функции не свойственны бортовым ЭВМ.

Комплекс автоматизированного тестирования ФПО

Структура комплекса автоматизированного тестирования ФПО показана на рис. 2, где ТЭВМ – технологическая и БЭВМ – бортовая ЭВМ; ВА – стандартная внешняя аппаратура; ОП – оператор. Ведущей является ТЭВМ, в качестве которой может использоваться достаточно мощный персональный компьютер. Физически это может быть, например, компьютер типа IBM PC/AT "Pentium 4".

Бортовая ЭВМ всегда отличается от технологической элементной базой, тактовой частотой, операционной системой, языком программирования и другими характеристиками. Для обмена информацией между операционными системами (*OS*) и программами обеих машин необходимо согласование интерфейсов. В бортовом компьютере размещается проверяемое реальное ФПО (РФПО), а в технологической ЭВМ – модель ФПО (МФПО). Реальное и модельное ФПО всегда подразделены на функционально одинаковые программы, что определяется решаемыми задачами. В технологической ЭВМ кроме модели ФПО размещаются: программный генератор входных тестовых наборов данных (Г), блок программ обработки результатов (БОР) и модули согласования интерфейсов (*SI*) обеих ЭВМ.

Генератор входных ТНД включает в себя генератор координат (*GK*) и обобщенную модель средств технического зрения. Генератор *GK* предназначен для формирования координат, скоростей и направлений движения СТС противника на основе заданной тактической ситуации в декартовой системе координат, а обобщенная модель *Z* служит для пересчета координат из декартовой в сферическую систему координат. Генератор Г работает в режимах разового и непрерывного формирования координат. Блок обработки результатов включает в себя: модули сравнения формуляров на выходе одноименных программ ФПО (*SP, SK, SR, SU*), программу отладки *O* (программу поиска и локализации ошибок) и программу испытаний *I* (программу расчета показателей качества ФПО).

Оператор вводит параметры тактической ситуации и по его сигналу формируется либо одиночный входной ТНД, либо серия ТНД, которые соответствуют развивающейся тактической ситуации. В последнем случае ТНД формируются с реальной частотой опроса средств технического зрения. Сформированный ТНД одновременно подается на вход МФПО и РФПО. По сигналу окончания работы программы *U* РФПО начинается поиск и локализация ошибок при отладке либо обработка результатов при испытаниях ФПО.

Алгоритм поиска и локализации ошибок ФПО

Отладка сводится к обнаружению ошибок в работе ФПО и определению (локализации) места их возникновения. Алгоритм поиска и локализации ошибок (алгоритм отладки) в строчной форме записи приведен на рис. 3. Алгоритм реализуется программой *O* и модулями *SP, SK, SR* и *SU* (см. рис. 2).

В блоке исходных данных алгоритма записываются выходные формуляры всех программ реального ФПО с расширением *R* и формуляры модели ФПО с расширением *M*. Здесь также указываются константы, определяющие допустимое расхождение (погрешность) в вычислениях одноименными программами РФПО и МФПО угловых (*DU*) и линейных (*DL*) величин, а также в вычислении моментов времени (*DT*). Работа начинается со сравнения формуляров *F.U*, а затем последовательно сравниваются формуляры всех предыдущих программ вплоть до *F.P*. Если разность одноименных числовых переменных не превышает допустимой погрешности или одноименные нечисловые переменные совпадают, то вырабатывается признак ошибки *PO* вычисления переменной со значением *N*, а в противном случае – со значением *Y*. Если в вычислении всех переменных формуляров *F.U* нет ошибки, значит и ошибка в работе всего РФПО отсутствует. Если есть ошибка в вычислении какой-либо переменной, то переходят к сравнению предшествующих формуляров (*F.R*). Их равенство свидетельствует о том, что ошибка в вычислении именно этой переменной произошла в программе *U*, а неравенство говорит об ошибке в одной из предшествующих программ. Выявленная ошибка устраняется путем коррекции программы *U*, и тестирование продолжается.

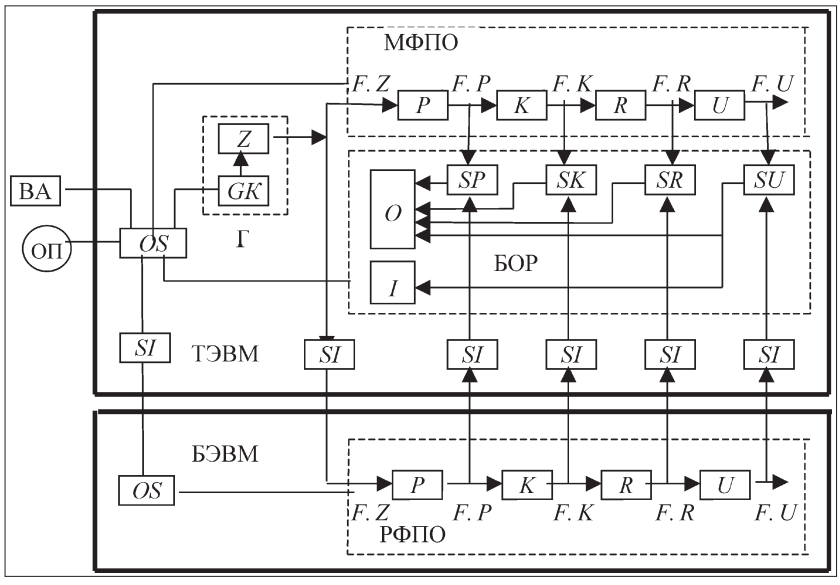

Рис. 2. Структура комплекса автоматизированного тестирования

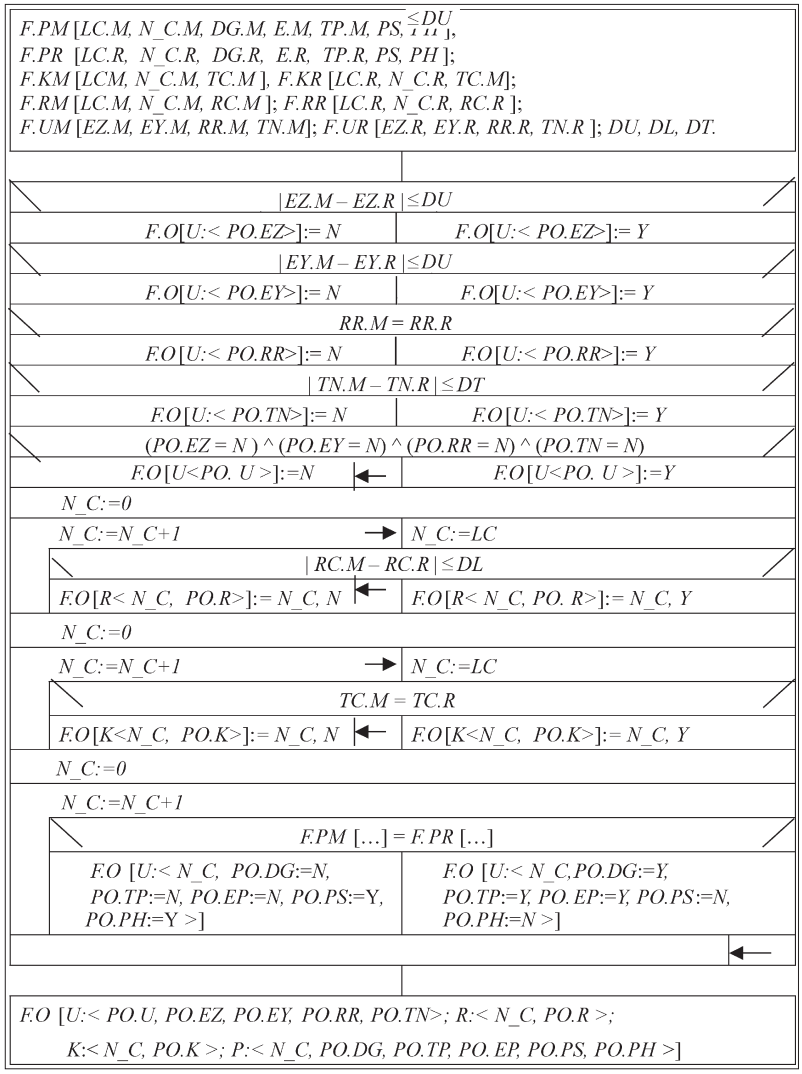

Рис. 3. Алгоритм поиска и локализации ошибок ФПО

Когда в выходном формуляре какой-либо программы присутствуют данные, относящиеся к нескольким СТС противника, то сравниваются формуляры относительно каждого из них. Это требует циклического перебора всех СТС.

Для уменьшения размерности приведенного алгоритма подробное сравнение всех переменных формуляров *F.P* обозначено как сравнение самих формуляров *F.PM* [...] = *F.PR* [...].

Выявленные ошибки в работе отдельных программ с точностью до имен переменных, в вычислении которых обнаружена ошибка, указываются в выходном формуляре алгоритма в виде признака ошибки *PO*. с расширением по имени переменной.

Функциональные показатели качества ФПО

Большинство показателей качества трудно формализовать и, следовательно, вычислить (например, эффективность, адаптивность, модернизируемость) [2]. Однако можно с уверенностью утверждать, что Заказчика в первую очередь интересуют показатели качества функционирования ФПО.

Простейшими (элементарными) свойствами ФПО [1] являются:

- *своевременность* (*SV*) – способность выработки решения за время, не превышающее заданное (*TZ*);
- *безотказность* (*BO*) – способность функционировать без отказов (отказ – это любое нарушение работы, например останов или зацикливание, требующее вмешательства оператора);
- *правильность* (*PR*) – вхождение результата работы программы в область допустимых решений (результатов).

Более сложными свойствами ФПО, выражающимися через указанные элементарные свойства, являются:

- *оперативность* (*OP*) – решение оперативно, если оно безотказно и своевременно;
- *корректность* (*KR*) – решение корректно, если оно безотказно и правильно;
- *надежность* (*ND*) – решение надежно, если оно безотказно, своевременно и правильно.

Соотношение между этими свойствами можно наглядно представить в виде отношений:

$$OP = BO \wedge SV; \quad KR = BO \wedge PR;$$
$$ND = OP \wedge KR = BO \wedge SV \wedge PR,$$

где $\wedge$ – логическая функция "И".

Эти свойства представляют собой некоторую систему, что вовсе не исключает введения в рассмотрение других свойств и соответствующих им показателей качества (например, время реакции ФПО на заданную ситуацию – *TR*).

Показатели качества определяются в процессе лабораторных испытаний ФПО по схеме тестирования с эталонной моделью. Однако оценка показателей качества ФПО отличается от его тестирования при отладке по двум причинам: 1) цель здесь – определение качества, а не поиск ошибок; 2) здесь важен не каждый отдельно взятый тест, а их совокупность.

Рассмотрим выражения, по которым можно рассчитать показатели ранее введенных качеств ФПО, а именно, показатель безотказности (*PBO*), показатель своевременности (*PSV*), показатель правильности (*PPR*), показатель оперативности (*POP*) показатель корректности (*PKR*) и показатель надежности *(PND*). *Показатель безотказности* $PBO = KB / KT$ есть отношение числа безотказных решений (*KB*) к общему числу тестов (*KT*); *показатель своевременности* $PSV = KS / KB$ – это отношение числа своевременных решений (*KS*) к числу безотказных решений (*KB*), а

показатель правильности $PPR = KP / KB$ – отношение числа правильных решений (KP) к числу безотказных решений. При вычислении PSV и PPR берется отношение к числу безотказных решений, так как при отказе нельзя судить ни о своевременности, ни о правильности решения.

Аналогичным образом можно вычислить и показатели оперативности, корректности и надежности. Показатель оперативности $POP = KO / KB$ – это отношение числа оперативных решений (KO) к числу безотказных решений. Показатель корректности $PKP = KK / KB$ – это отношение числа корректных решений (KK) к числу безотказных решений. Показатель надежности $PND = KN / KT$ – это отношение числа надежных решений (KN) к общему числу тестов.

В идеальном случае все показатели равны 1. Реально они могут выражаться в процентах от 1. Для оценки качества ФПО не требуется разработка специальных тестов, а могут использоваться ранее разработанные тесты для его отладки. Статистическая устойчивость результатов обеспечивается обычно за счет достаточно большого числа тестов (несколько десятков).

Алгоритм расчета показателей качества ФПО

Алгоритм обработки результатов при испытаниях (алгоритм испытаний), по существу, сводится к определению в каждом из множества тестов трех простейших свойств реакции ФПО на входную тактическую ситуацию (безотказность, своевременность и правильность) и к дальнейшему расчету на их основе функциональных показателей качества ФПО.

Алгоритм расчета показателей качества ФПО в строчной форме записи представлен на рис. 4. В блоке исходных данных указываются выходные формуляры программ U испытуемого ФПО и его модели, допустимое расхождение в вычислениях одноименными программами РФПО и МФПО угловых (DU) и линейных (DL) величин и моментов времени (DT), а также допустимое время на принятие решения (TZ).

В алгоритме вырабатываются признаки прохождения тестов: QB:{Y, N} – признак безотказности, QP:{Y, N} – признак правильности, QS:{Y, N} – признак своевременности.

Перед началом испытаний в блоке инициализации устанавливаются равными нулю номер теста и все показатели качества. В алгоритме подсчитывается число выполненных тестов, а при прохождении каждого очередного теста оператор следит за безотказностью работы ФПО и указывает признак $QB = N$, если произошел отказ.

При испытаниях ФПО проверяется совпадение выходных формуляров только программ U. Когда нет ошибки в реакции ФПО на заданный тест, вырабатывается признак $QP = Y$. Время реакции ФПО (TR) на заданный тест определяется счетчиком времени, который запускается входным ТНД и останавливается с окончанием вычислений в программе UR. Тестирование проводится до тех пор, пока не будет выполнена вся серия тестов, оговоренная в программе испытаний.

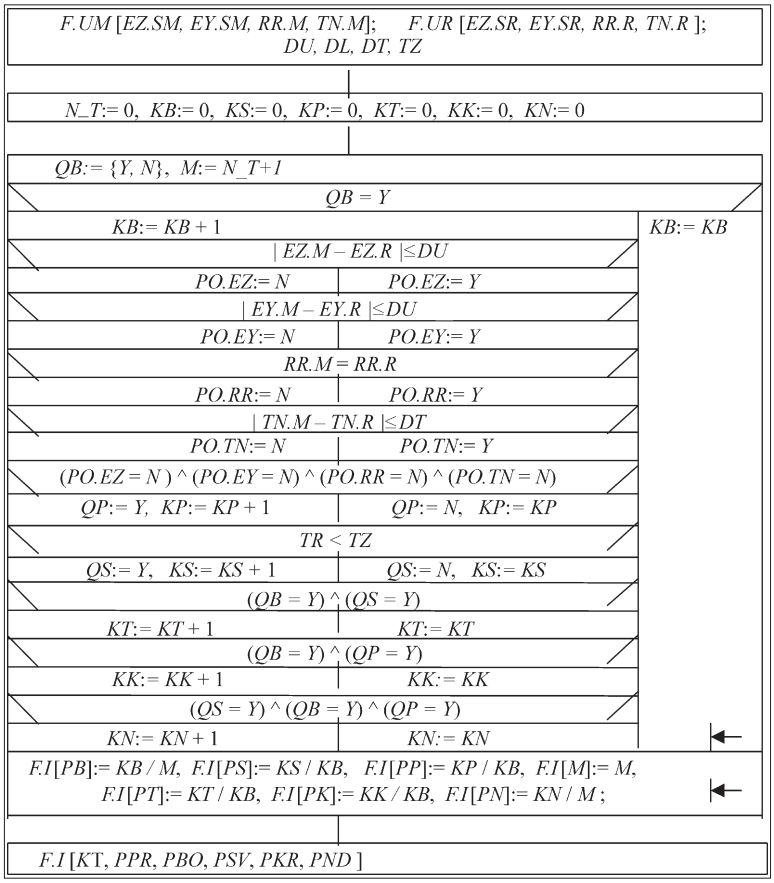

Рис. 4. Алгоритм расчета показателей качества ФПО

Приведенные материалы показывают возможность автоматизации процессов тестирования ФПО при его отладке и при испытаниях, что позволяет повысить качество и сократить сроки разработки комплексов управления подвижных объектов. Предложенный подход также может быть использован и для автоматизации тестирования программных средств других технических систем.

СПИСОК ЛИТЕРАТУРЫ

1. **Лямкин А.А.** Концептуальное проектирование средств управления подвижных объектов. СПб.: Изд-во СПбГЭТУ, 2006.

2. **Липаев В.В.** Тестирование программ. М.: Радио и связь, 1986.

3. **Лямкин А.А., Жарковский А.В., Микуленко Н.П., Тревгода Т.Ф.** Модели подвижных объектов при концептуальном проектировании // Мехатроника, автоматизация, управление. 2007. № 11. С. 2–6.

НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

ИНФОРМАЦИЯ

105679, Москва
ул. Кирпичная, д. 33
Телефон: (495)772-9590∗5151
E-mail: se.department@gmail.com
www.se.hse.ru

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Программная инженерия – область компьютерных наук и информационных технологий, которая занимается разработкой систематических моделей и надежных методов производства высококачественного программного обеспечения в индустриальных масштабах.

Факультет программной инженерии готовит высококвалифицированные кадры для бурно развивающейся отрасли современной мировой экономики – индустрии программного обеспечения.

Образован в 2006 году и первым в России начал подготовку бакалавров и магистров по новому направлению "Программная инженерия" по оригинальным учебным планам, разработанным с учетом международных образовательных и профессиональных стандартов.

В обучении:

- используется принцип интегрирования математики, информатики, психологии, управления проектами, экономики, бизнес-аналитики с инженерными подходами;
- реализован сбалансированный подход к изучению теории и использованию методологий создания программного обеспечения на практике;
- практика в ведущих компаниях – лидерах индустрии информационно-коммуникационных технологий;
- опыт научно-исследовательской и проектной работы в транснациональных компаниях.

Уникальные особенности программы:

- международная сертификация по английскому языку;
- профессиональная международная сертификация IEEE CS CSDA и CSDP выпускников на соответствие компетенциям международного профессионального стандарта ISO Guide to the Software Engineering Body of Knowledge (SWEBOK).

Трудоустройство выпускников:

- российские и зарубежные компании – производители программного обеспечения;
- научно-исследовательские центры транснациональных компаний;
- системные интеграторы;
- департаменты информационных технологий крупных российских компаний и государственных структур.

Вступительные экзамены на факультет программной инженерии:

- математика;
- информатика;
- русский язык.

Полезные ссылки:

http://se.hse.ru – факультет программной инженерии

http://si:hse.ru – студенческая школа информатики (подготовка к олимпиадам и ЕГЭ по информатике)

Уважаемые коллеги!

Приглашаем Вас к авторскому участию в журнале "Программная инженерия"

МАТЕРИАЛЫ, ПРЕДСТАВЛЯЕМЫЕ В РЕДАКЦИЮ:

Статья в двух экземплярах, оформленная в соответствии с требованиями.
Таблицы, иллюстрации и перечень подрисуночных подписей.
Фамилии и инициалы авторов, название, аннотация и ключевые слова статьи на английском языке.
Аннотация и ключевые слова на русском языке.
Сведения об авторах (фамилия, имя, отчество, ученая степень, место работы, занимаемая должность, телефоны, e-mail).
Список литературы.
Акт экспертизы.
Если статья высылается почтовой бандеролью, то необходимо вложить диск с электронной версией перечисленных материалов.

ТРЕБОВАНИЯ К СОДЕРЖАНИЮ И ОФОРМЛЕНИЮ СТАТЬИ

На первой странице статьи указывается индекс УДК (в левом верхнем углу), далее по центру даются инициалы и фамилии авторов (с указанием ученой степени, должности и полного названия места работы) и затем по центру идет название статьи, кратко и точно передающее ее содержание.

Перед текстом статьи отдельным абзацем приводится краткая аннотация, рассчитанная на широкий круг читателей и точно отражающая существо статьи (целевую направленность, новизну и область применения полученных результатов).

В следующем абзаце даются ключевые слова на русском языке.

Далее следует текст статьи, причем желательно статью структурировать, разбив ее на разделы с названиями, отражающими их содержание.

Объем статьи не должен превышать **15** ***страниц текста***, выполненного на стандартных листах с полями ***не менее 2 см, шрифтом Times New Roman размером 14 pt с полуторным межстрочным интервалом.*** В указанный объем статьи включаются приложения, список литературы, таблицы и рисунки. Страницы статьи должны быть ***пронумерованы.*** В отдельных случаях по решению редколлегии объем статьи может быть увеличен.

В формулах русские и греческие буквы следует набирать прямо; латинские буквы – курсивом; величины, обозначающие векторы и матрицы, должны быть выделены полужирным шрифтом и набраны прямо (допускается также набор всех величин, обозначенных латинскими буквами, в т.ч. матриц и векторов, светлым курсивом); знак транспонирования – буквой "т" строчной прямой; **e** в значении "экспонента" – полужирной прямой.

Номера формул располагаются справа в круглых скобках. Нумерация формул – сквозная, причем нумеруются те формулы, на которые имеются ссылки. Ссылки на литературные источники даются в квадратных скобках. Список литературы имеет сквозную нумерацию в порядке упоминания в тексте и содержит следующие данные: фамилия и инициалы автора, название работы, название журнала, сборника (если это не монография), город и полное название издательства (для монографии), год издания, номер (том) журнала и страницы. Допускаются ссылки на электронные носители.

В тексте следует придерживаться общепринятой терминологии. Термины и определения, а также единицы физических величин, употребляемые в статье, должны соответствовать действующим ГОСТам. Все используемые специальные термины, обозначения и аббревиатуры должны быть раскрыты и разъяснены.

Рисунки и таблицы должны быть выполнены качественно на лазерном принтере. Все рисунки воспроизводятся в журнале в черно-белом варианте, за исключением цветных рисунков, размещаемых по усмотрению редакции на обложке журнала. Рисунки и таблицы дублируются на отдельных листах, подрисуночные подписи даются отдельным списком. На рисунках не допускается особо мелкая надписка; буквенные и цифровые изображения на рисунках по начертанию и размеру должны соответствовать обозначениям в тексте.

На последней странице рукописи должны быть подписи авторов.

Дополнительные пояснения можно получить в редакции журнала.

Возможно также представление статей в электронном виде, требования такие же.

При подготовке рисунков в электронном виде просим соблюдать следующие требования:

1. Толщина линий должна быть не менее 0,15 мм.
2. Масштаб надписей должен быть соразмерным с самим рисунком.

АДРЕС РЕДАКЦИИ:
107076, г. Москва, Стромынский пер., 4
Издательство "Новые технологии",
Редакция журнала "Программная инженерия"
Телефоны: 8-(499) 269-55-10; 269-53-97
Факс: 8-(499) 269-55-10
E-mail: prin@novtex.ru

CONTENTS

ООО "Издательство "Новые технологии", 107076, Москва, Стромынский пер., 4

Дизайнер *Т.Н. Погорелова*. Технический редактор *Т.И. Андреева*. Корректоры *Л.И. Сажина, Л.Е. Сонюшкина*

Сдано в набор 13.10.10 г. Подписано в печать 24.11.10 г. Формат 60×88 1/8. Бумага офсетная. Печать офсетная.
Усл. печ. л. 5,88. Уч.-изд. л. 7,05. Цена свободная.

Отпечатано в ООО "Белый ветер", 115407, г. Москва, Нагатинская наб., д. 54, пом. 4.