ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г.

СОДЕРЖАНИЕ

Этот номер журнала подготовлен при участии и финансовой поддержке платежной системы "КиберПлат"

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

УДК 004.41

В.А. Васенин, д-р физ.-мат. наук, проф., зав. отделом,
Институт проблем информационной безопасности МГУ им. М.В. Ломоносова

E-mail:vasenin@msu.ru

О задачах и основных тематических направлениях журнала "Программная инженерия"

Обсуждается круг задач и основные тематические направления журнала "Программная инженерия".

Ключевые слова: *журнал "Программная инженерия", задачи, тематические направления*

Процесс производства программ за более чем полувековую историю существования электронно-вычислительных машин существенно изменился. От программирования, как сугубо интеллектуальной деятельности, которая сродни искусству [1–4], к настоящему времени этот процесс в большей степени приобретает черты массового производства – индустрии создания программного продукта, которая требует подлежащей регламентации ее нормативной базы и стандартизации. Активное внедрение средств и систем автоматизации различных сфер деятельности человека влечет за собой многообразие поддерживающих их программных продуктов (программных средств, комплексов). Различны подходы к проектированию и изготовлению, к сопровождению, модификации и снятию с эксплуатации таких продуктов. При создании уникальных, как правило, коротко живущих программ узкого (специального), например научного, назначения не возникает необходимости решать вопросы, связанные с их эффективным отчуждением и удобствами эксплуатации, с оптимизацией кода и его модифицируемостью, с масштабностью программ, их надежностью и защитой от деструктивных воздействий. Характеристики качества для таких программ не имеют принципиального значения. Для программных комплексов, поддерживающих средства автоматизации процессов, протекающих в объектах критически важных для государства инфраструктур [5], многие из перечисленных характеристик, наоборот, приобретают решающее значение. Столь же важны они и для программных продуктов широкого спектра применения, которые производятся и используются в массовых масштабах.

Следует заметить, что вопросы регламентации и стандартизации процессов и технологий, методов и средств разработки и внедрения, эксплуатации и модификации программных комплексов становятся актуальными для государства при определенном, относительно высоком уровне информатизации всех сфер его общественных отношений. К их числу относятся традиционные – материальная, социальная, духовная и политическая сферы [7, 8]. Этот уровень характеризуется появлением новой сферы складывающихся в обществе (государстве) отношений – сферы информационной. В России к настоящему времени есть все предпосылки для формирования и развития информационной сферы, соответствующей мировым тенденциям. Такое развитие будет в ближайшие годы одним из важнейших резервов и определяющим фактором перехода России на инновационно-технологический путь развития. Только эффективное использование годами накопленного в мире опыта производства и эксплуатации программных комплексов различного назначения позволит создавать со-

ответствующие мировым стандартам отечественные средства и системы автоматизации технологических процессов во всех, особенно в критически важных для страны, секторах национального хозяйственного комплекса.

К сожалению, в настоящее время приходится констатировать, что должного понимания в массах и широкого распространения на практике мировой опыт в России пока не находит. И это, несмотря на то что определенные работы на этом направлении велись ранее и ведутся в настоящее время [10], на появление в последние годы ряда изданий учебного плана, например работ [9–17].

Причиной этому является и объективно сложившееся отставание России от технологически развитых стран в области микросхемотехники, аппаратной базы, средств телекоммуникаций и, как следствие, их программного обеспечения. Вместе с тем относительно высокие темпы информатизации общества, развития телекоммуникационной инфраструктуры, появление в стране современных средств вычислительной техники и высокопроизводительных систем дают основания для надежды на устранение отмеченных диспропорций. Основания для оптимизма – в позиции, которую в последние годы активно озвучивают представители высших органов государственной власти в отношении стратегии перехода страны к информационному обществу [18]. Важным на этом направлении является решение комиссии при Президенте РФ "О развитии компьютерных технологий и программного обеспечения", а также целый ряд следующих за ним документально подкрепленных практических шагов государства.

Следует, однако, иметь в виду, что от решений даже на самом высоком государственном уровне до эффективных практических действий, а тем более результатов – долгий и тяжелый, кропотливый путь большого числа специалистов, которые задействованы в сфере разработки и эксплуатации программных продуктов. Развитие компьютерных технологий и программного обеспечения невозможно без использования мирового и отечественного опыта проектирования и разработки, внедрения и сопровождения программных комплексов, основанного на регламентированных действиях и стандартах. Вместе с тем должного количества квалифицированных специалистов, обладающих таким опытом и навыками в России на сегодня объективно нет. К их массовой подготовке в необходимом объеме российская высшая школа сегодня не готова. Необходимы некоторые стимулы, которые позволили бы активизировать работы, направленные на устранение этих недостатков. Следует правильно выбрать первоочередные задачи, требующие индустриального подхода к разработке и использованию программ на приоритетных для государства направлениях инновационно-технологического развития, сосредоточиться на контроле их безусловного решения, постепенно расширяя и перечень направлений, и список задач. На мой взгляд, журнал "Программная инженерия" должен стать местом обсуждения актуальности таких направлений и перечня задач, подходов к их решению и практических результатов. Главные задачи журнала: обсуждение и аккумулирование идей и предложений на этом направлении, их анализ и выработка стратегических целей и тактики их реализации; обмен опытом и демонстрация результатов научных исследований и прикладных работ, направленных на создание программного обеспечения на всех этапах его жизненного цикла; вопросы подготовки кадров, обладающих необходимым комплексом знаний и навыков индустриального подхода к созданию программ.

Журнал только начинает свою жизнь. Пока в окончательном виде не сформирована его редакционная коллегия и тематические направления. На стадии обсуждения вопросы организации эффективного взаимодействия с подписчиками и авторами. Да и сам термин "Программная инженерия" в силу отмеченных выше причин в России пока еще не имеет однозначного определения. Появляются разные его толкования, иногда отличные от принятых за рубежом [19]. Одним словом, обсуждению подлежит весь комплекс вопросов, которые характерны для любого вновь организованного журнала, затрагивающего новую, неапробированную область научно-технической деятельности. В этой связи представляется вполне уместным предоставление его страниц различным идеям, суждениям и предложениям. Думается, что такой диалог с заинтересованными в судьбе журнала авторами не только не нанесет ущерб его репутации, а наоборот, расширит его читательскую аудиторию, привлечет внимание потенциальных (будущих) авторов и позволит бо-

лее эффективно аккумулировать их идеи и предложения.

Настоящая публикация – призыв к такому обсуждению. Она отражает частное мнение автора на трактовку понятия "программная инженерия" как области научно-технической деятельности и отдельной учебной дисциплины со своими объектами исследования, методами ее изучения и развития, использования полученных в ее рамках знаний для обучения кадров, применяющих эти знания. Далее в конспективной форме (кратко) постараюсь изложить свои соображения по перечисленным вопросам.

Под программной инженерией понимается область научно-технической деятельности, объединяющей совокупность взаимосвязанных между собой математических моделей, разноплановых методов, реализующих их инструментальных средств и систем, основным назначением которой является развитие современной индустрии программного обеспечения. Такое развитие обеспечивается соблюдением международного опыта принятых методологией программной инженерии положений, включающих административно-управленческие, технические и технологические нормативы, стандарты и рекомендации, решения теоретического и прикладного характера, инструментальные средства и системы, которые обобщают результаты многолетних фундаментальных исследований и прикладных работ ученых и инженерно-технических работников в этой области и регламентируют деятельность на этапах проектирования, производства, внедрения и эксплуатации сложных программных продуктов.

Представляется целесообразным освещать в журнале состояние и тенденции основных направлений теоретических исследований и прикладных работ в области индустрии программного обеспечения, связанных с проектированием, архитектурой, обеспечением качества, сопровождением жизненного цикла программных комплексов. Следует, на мой взгляд, рассматривать в журнале достижения в области создания и эксплуатации автоматизированных информационных систем во всех областях научной и хозяйственной деятельности, а также вопросы подготовки кадров по следующим основным тематическим направлениям. С учетом изложенного к основным тематическим направлениям, которые нужно освещать в журнале, можно отнести перечисленные далее.

Теоретические вопросы программной инженерии. Здесь речь может идти о следующем круге вопросов:

– процессы и модели жизненного цикла программных средств;

– модели, требования и стандарты управления качеством программных продуктов;

– модели оценивания зрелости продуктов и процессов разработки программных средств;

– стандарты функциональной безопасности программируемых электронных систем;

– модели и стандарты обеспечения информационной безопасности ресурсов сложных программных комплексов;

– модели верификации и валидации (подтверждения достоверности) программных средств;

– вопросы экономики, психологии и права при разработке сложных комплексов программ;

– вопросы подготовки и переподготовки кадров в области программной инженерии.

Методы и инструментальные средства программной инженерии. К числу данных методов и средств можно отнести:

– методы и средства разработки и тестирования программных средств;

– методы и средства обеспечения информационной безопасности;

– методы и средства управления надежностью программных средств и рисками неблагоприятных событий на всех этапах их жизненного цикла;

– системы управления качеством сложных программных продуктов;

– методы и средства верификации и валидации функциональных свойств и качества программных продуктов;

– документирование процессов разработки, сопровождения и модификации программного обеспечения.

Управление программными проектами. Данное управление должно обеспечивать:

– организационные модели управления программными проектами;

– механизмы и методы обеспечения совместной работы участников программных проектов;

– методы и средства управления программными конфигурациями (контроль версий);

– методы и средства планирования программных проектов;

– интегрированные среды разработки, управления, тестирования, установки и ввода в действие программного обеспечения;

– организация инфраструктуры разработки и ввода в эксплуатацию.

Нормативно-правовые и организационные вопросы программной инженерии. К подобному кругу вопросов относятся:

– лицензирование программного обеспечения;

– защита интеллектуальной собственности (защита программ от несанкционированного тиражирования и использования);

– использование информационных технологий в государственном секторе, вопросы стандартизации;

– программное обеспечение с открытым кодом (open source) и др.

Технологии разработки и анализа программ. В качестве таковых можно рассматривать:

– извлечение, анализ и моделирование требований;

– парадигмы моделирования вычислительных систем;

– статический и статико-динамический анализ программ;

– динамическую верификацию;

– метрики тестового покрытия;

– моделирование, измерение и тестирование производительности и потребления ресурсов;

– интеграцию различных методов верификации;

– вопросы обучения технологиям разработки и анализа программ.

Языки и системы программирования, семантика программ. Здесь могут обсуждаться:

– парадигмы, языки и системы программирования;

– формальные модели и программные механизмы представления синтаксиса и семантики языков и программ;

– средства и системы компиляции программ;

– опыт использования современных языковых средств и систем программирования в различных предметных областях.

Методы, средства и системы управления базами данных и знаний. Предметом рассмотрения данного направлении могут быть:

– модели, методы и средства формального описания знаний, хранения и управления ими;

– модели, методы и средства управления распределенными данными, в том числе из источников с разной структурой их организации;

– системы и механизмы управления базами данных;

– опыт использования программного обеспечения баз данных и знаний в различных прикладных областях;

– подготовка кадров в области программного обеспечения современных систем управления данными и знаниями.

Человеко-машинные интерфейсы, средства визуализации, обработки изображений, системы виртуальной реальности и мультимедийного общения. На этом направлении могут рассматриваться:

– математические модели человеко-машинного общения и программные механизмы их реализации;

– модели, методы и программные механизмы извлечения данных, их визуализации и обработки изображений;

– программные системы виртуальной реальности;

– современные программные средства и системы мультимедийного общения и опыт их эксплуатации в составе предметно-ориентированных систем.

Модели, методы и инструментальные средства создания сложных программных продуктов для параллельной и распределенной обработки данных. Предмет рассмотрения на данном направлении могут составлять:

– парадигмы, языки и системы параллельного программирования;

– формальные модели описания языков и систем параллельного программирования;

– модели, методы и средства высокопроизводительной обработки данных в составе распределенных автоматизированных систем;

– программное обеспечение технологий и средств повышения производительности суперкомпьютерных установок;

– подготовка кадров в области программного обеспечения суперкомпьютерных систем и высокопроизводительной обработки распределенных данных.

Создание и эксплуатация программного обеспечения прикладных автоматизированных систем. Данное направление может включать:

– опыт создания и эксплуатации программного обеспечения автоматизированных систем

управления технологическими процессами в различных предметных областях;

– методы и средства автоматизации процессов управления проектами разработки сложных программных систем;

– подготовка кадров для эксплуатации программного обеспечения прикладных автоматизированных систем.

В заключение автор считает необходимым еще раз подчеркнуть, что не считает перечисленные направления единственно возможными, каноническими для освещения в журнале "Программная инженерия" и всегда готов ознакомиться с другими мнениями и предложениями по рассматриваемым вопросам.

СПИСОК ЛИТЕРАТУРЫ

1. **Кнут Д.Э.** Искусство программирования. Том 1. Основные алгоритмы. 3-е издание. М.: Вильямс, 2009. 720 с.

2. **Кнут Д.Э.** Искусство программирования. Том 2. Получисленные алгоритмы. 3-е издание. М.: Вильямс, 2010. 832 с.

3. **Кнут Д.Э.** Искусство программирования. Том 3. Сортировка и поиск. 2-е издание. М.: Вильямс, 2008. 824 с.

4. **Кнут Д.Э.**. Искусство программирования. Том 4. Генерация всех кортежей и перестановок. 2-е издание. М.: Вильямс, 2007. 160 с.

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

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

7. **Безопасность** России. Правовые, социально-экономические и научно-технические аспекты. Информационная безопасность. М.: МГФ "Знание", ГЭИТИ, 2005. 512 с.

8. **Стрельцов А.А.** Правовое обеспечение информационной безопасности России: теоретические и методологические основы. Минск, 2005. 304 с.

9. **Липаев В.В.** Программная инженерия. Методологические основы: Учеб. Гос. ун-т – Высшая школа экономики. М.: ТЕИС, 2006. 608 с.

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

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

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

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

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

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

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

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

18. **Стратегия** развития информационного общества в Российской Федерации от 7 февраля 2008 г. № ПР-212. // Российская газета. 2008. 16 февраля.

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

УДК 004.41

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

E-mail: alexlip@mail.ru

Проблемы программной инженерии: качество, безопасность, риски, экономика

В настоящей статье, изложение которой предполагается в двух частях, рассмотрены основные научные, методологические и технологические проблемы программной инженерии, которые возникают на различных этапах жизненного цикла современных сложных комплексов программ. В первой части представлены проблемы обеспечения качества программных продуктов, безопасности их применения, сокращения рисков неблагоприятных событий и экономики в процессе создания, эксплуатации и модификации программных систем. Для разрешения этих проблем в них выделены и сформулированы около двадцати наиболее актуальных частных задач, для некоторых из них представлены методы и пути решения.

Ключевые слова: *программная инженерия, жизненный цикл, сложный комплекс программ, программный продукт, методология*

Введение

На протяжении нескольких десятилетий двадцатого века происходило практически независимое от зарубежных стран оригинальное развитие отечественной вычислительной техники и программирования, научной и прикладной составляющей этой области. Созданные нашими учеными и инженерами сложные вычислительные системы и комплексы программ для различных сфер применения многие годы находились на уровне мировых достижений и не содержали зарубежных компонентов. Этот путь самостоятельного развития отечественных ЭВМ и программирования [3], в том числе в закрытых областях, практически неизвестен современному поколению специалистов в области информатики. В эти годы необходимость в технологических программных средствах (ПС) и первых операционных систем стимулировалась преимущественно появлением и освоением конкретных типов машин. Такие средства разрабатывались научными организациями и даже вузами, и обычно не имели определенных, формальных заказчиков.

Важнейшее направление составляли крупные комплексы программ реального времени для сложных систем управления и обработки информации, которые создавались большими коллективами специалистов преимущественно для оборонных систем и оформлялись в виде программных продуктов с гарантированным качеством. Такие комплексы программ являлись компонентами систем, которые обычно автоматизировали их основные функции и содержали предпосылки для последующего развития и изменений. Методологии управления проектами, создания и внедрения ПС зависели от многих факторов, в том числе от персонала, технических, организационных, договорных требований и сложности выполняемых программами функций. Организованная, контролируемая и слаженная коллективная разработка при строгом учете и контроле каждого изменения являлась основой эффективного, поступательного развития каждой крупной системы методами программной инженерии. Однако методологиче-

ские и технологические достижения в этой области передовых предприятий оборонной промышленности оставались секретными, не отражались в открытых публикациях и были неизвестны специалистам даже близких по функциям и задачам предприятий.

Процессы разработки комплексов программ для оборонных систем с самого начала отличались организованностью и тесным взаимодействием с заказчиками. При этом требования заказчиков к функциям и качеству программных продуктов постоянно превышали возможности разработчиков и ресурсы доступных для использования вычислительных машин. Это стимулировало совершенствование тех и других, а также необходимость формализации технологий применявшихся тогда в программной инженерии. Одновременно повышалась производительность труда специалистов и качество программного продукта. Результатами этих усилий в разработке сложных комплексов программ для оборонных систем было то, что в период 60–90-х годов XX века основные отечественные системы вооружения с использованием вычислительной техники имели функциональные характеристики и качество, практически адекватные, а иногда и превышавшие аналогичные оборонных систем США.

В 1980-е годы государственная политика в области вычислительной техники и программирования круто изменилась в сторону копирования и массового "заимствования" программных средств, разработанных за рубежом. Это привело к деградации и отставанию на десяток лет отечественной школы программной инженерии. Ориентация на программные продукты, поступающие на российский рынок из-за рубежа, не могла удовлетворить многие требования, которые следовало предъявлять на этапах создания и применения таких продуктов в стране. Только в начале XXI века обострилась проблема, связанная с отсутствием собственных отечественных разработок оригинальных крупных программных продуктов для ряда отраслей государственного управления, народного хозяйства и оборонной техники. Для многих заказчиков и пользователей систем возникла проблема поиска и выбора квалифицированных и надежных специалистов-подрядчиков, способных создавать сложные программные средства и базы данных необходимого качества в разумные сроки, с учетом ограничений на затраты труда и другие ресурсы. В последние несколько лет в ряде передовых вузов начал проявляться интерес к решению этой важнейшей государственной проблемы.

На настоящее время органы государственного управления в области информатизации (на уровне министерств и ведомств) должным образом не занимаются координацией и управлением работами в области программной инженерии для различных сфер ее применения. Программные средства для науки и промышленности, которые являются важнейшей интеллектуальной частью всех систем информатизации, оказались вне их интересов. В результате многие современные проекты сложных ПС оказываются не конкурентоспособными, недостаточного качества и требуют длительной доработки (например, ГАС "Выборы") для устранения методологических, системных ошибок и технических дефектов и ошибок на этапах их разработки и внедрения.

Накопление в мире теоретических знаний, опыта разработки и применения огромного количества различных сложных программ для ЭВМ способствовало систематизации и обобщению методов и технологий их разработки, сокращению дефектов и неопределенностей в характеристиках и качестве поставляемых и применяемых программных продуктов. В результате в 1990-е годы в основном сформировалась современная методология и инженерная дисциплина обеспечения процессов жизненного цикла сложных программных продуктов – программная инженерия для различных областей применения таких ПС. В России отдельные положения программной инженерии нашли применение преимущественно в оборонной промышленности.

Программная инженерия – это область компьютерной науки и технологии, которая занимается построением программных систем, настолько больших и сложных, что для этого необходимо привлекать крупные "команды" разработчиков различных специальностей и квалификаций. Обычно такие системы существуют и применяются долгие годы, развиваясь от версии к версии, претерпевая на своем жизненном пути большое число изменений. Улучшаются их функции, добавляются новые или удаляются устаревшие возможности, дополняются компоненты для работы в новой среде окружения, устраняются дефекты и

ошибки. Суть методологии программной инженерии состоит в применении систематизированного, научного и предсказуемого процесса планирования, проектирования, разработки и сопровождения сложных программных продуктов.

Программная инженерия охватывает все аспекты жизненного цикла (ЖЦ) ПС от начальной стадии разработки системных требований и алгоритмов до этапа, на котором завершается использование программного продукта. На всех этапах ЖЦ специалисты выполняют научные исследования, практическую и инженерную работу. Они должны применять теоретические построения, методы и средства там, где это необходимо. Однако делать это следует выборочно, пытаясь найти практическое решение задачи даже в случае, если не существует подходящей теории или методов решения. Исполнители такого сорта проектов должны понимать, что они работают в организационных и финансовых рамках заключенных контрактов, и искать решение поставленной перед ними задачи с учетом их условий.

Во многих предприятиях и вузах отношение к программам для ЭВМ исторически базируется на подходе как к "искусству и художественному творчеству" отдельных специалистов (программирование "в малом"). При этом считается, что невозможно применять какие-либо экономические характеристики для определения и прогнозирования стоимости и результатов такого творчества, которое разработчики оценивают только с позиции выполняемых ими функций и "эстетики" их реализации. Такие программы не предназначены для массового тиражирования и распространения как программного продукта на рынке информационных технологий, их оценивают качественно и интуитивно, преимущественно как "художественные произведения". Разработчики подобных программ не знают и не применяют нормативных документов, регламентирующих их производство. Как следствие, жизненный цикл таких изделий имеет непредсказуемый характер по структуре, содержанию, качеству и стоимости основных результатов творчества. Их создание не определяется экономическими характеристиками и регламентированными производственными процессами и далее не рассматривается.

Для небольших относительно простых программ во многих случаях достаточно достоверными могут быть интуитивные оценки требуемых экономических ресурсов, которые получаются опытными руководителями, реализовавшими, например, несколько аналогичных проектов. Однако интуитивные оценки руководителями размеров и сложности программных проектов (программирование "в большом"), как правило, отличаются большими ошибками при планировании экономических характеристик: сроков, трудоемкости и стоимости создания продуктов. Во многих случаях это приводит к значительному запаздыванию завершения разработок и к превышению предполагавшихся затрат. Практика последних лет показывает, что по причине отсутствия экономического обоснования до 15 % проектов сложных программных комплексов не доходят до завершения. Почти половина сложных проектов не укладывается в выделенные ресурсы, бюджет и сроки, не обеспечиваются заданные характеристики качества. Типичны ситуации, когда отставание сроков внедрения промышленных автоматизированных систем управления и обработки информации полностью зависит от отсутствия готовности для них программных продуктов.

Основные концептуальные положения программной инженерии [1] сконцентрировались в целостном комплексе международных стандартов, регламентирующих практически все процессы жизненного цикла сложных программных средств. Задача состоит в их активном освоении и применении. Использование стандартов, аккумулировавших мировой опыт создания различных типов крупных комплексов программ [4], способствует снижению стоимости, трудоемкости, длительности, а также улучшению других технико-экономических показателей подобных проектов [5, 6], повышению производительности труда специалистов и качества создаваемых продуктов. Жизненный цикл крупных программных средств отражается в таких стандартах [2, 7, 8] набором процессов, этапов, частных работ и операций, в последовательности их выполнения и взаимосвязи. Они регламентируют ведение разработки, сопровождение и эксплуатацию, от анализа и подготовки требований до завершения испытаний ряда версий программного продукта и прекращения их использования. Тем самым уровень достигаемого качества 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. По мере воз-

растания размеров проектов и ответственности за качество и безопасность применения программных продуктов, совершенствуются технологии их создания, повышаются требования к полноте и корректности применения стандартов подобно тому, как это происходит в других отраслях промышленного производства.

Новая вычислительная техника позволяет воплощать в жизнь не реализуемые ранее сложные программные приложения. В результате программные комплексы достигли размеров и уровня сложности, намного превышающих аналогичные показатели у программных систем, реализованных на вычислительной технике предыдущих поколений. Массовое создание сложных программных продуктов промышленными методами и большими коллективами специалистов вызвало необходимость их четкой организации, планирования работ по требуемым ресурсам, этапам и срокам реализации. Возникла необходимость в новых технологиях, методах создания и управления сложными проектами больших программных систем.

Благодаря развитию операционных систем и инструментальных средств универсальных персональных ЭВМ и серверов накапливаются готовые программные компоненты и сокращается необходимость их программирования. Это приводит к повышению роли интеграции таких компонентов, соответствующих методов и инструментария программной инженерии. Однако по причине их принципиальной новизны и сложности, они трудно воспринимаются традиционными программистами малых компонентов и большинством преподавателей отечественной высшей школы. Коренные отличия между методами и инструментарием индивидуального, "художественного" программирования небольших программ и технологией планомерной, регламентированной, инженерной разработки крупных программных продуктов приводит к тому, что последние медленно осваиваются и входят в практику слаженной работы больших коллективов специалистов. Эти обстоятельства отражаются на существенном отставании от мирового уровня по конкурентоспособности, по количеству и качеству отечественных программных продуктов. Оригинальные отечественные программные продукты составляют на мировом рынке менее 1 %.

Крупные комплексы программ являются компонентами систем, реализующими обычно их основные, функциональные свойства и создающими предпосылки для последующих изменений их жизненного цикла. Реализация жизненного цикла, методология управления и изменения программных средств зависит от многих факторов, в том числе от квалификации специалистов, технических, организационных и договорных требований и сложности проекта. Большое число текущих состояний и модификаций компонентов в жизненном цикле сложных ПС приводит к тому, что менеджерам необходимо упорядочивать, контролировать их развитие и реализацию участниками проекта. Организованное, контролируемое и методичное отслеживание динамики изменений в жизненном цикле программ и данных, их разработка при строгом учете и контроле каждого изменения является ключевой задачей обеспечения эффективного, поступательного развития каждой крупной системы методами программной инженерии.

Далее внимание акцентируется на вопросах обеспечения эффективности процессов практической разработки и сопровождения сложных комплексов программ. Важнейшими факторами улучшения этих процессов являются количественные и качественные измерения результатов деятельности специалистов, обучение и повышение их квалификации. Для решения этих задач необходимо управление персоналом, освоение новых технологий и инструментальных средств, контроль состояния, качества и изменения компонентов проекта, эффективное использование всех видов ресурсов.

1. Проблема обеспечения качества продуктов в программной инженерии

Задача оценивания и выбора квалифицированных и надежных специалистов-подрядчиков, способных создавать сложные программные средства и информационные системы требуемого качества в разумные сроки, с учетом ограничений на выделяемые для этого ресурсы, стоит остро для многих заказчиков и пользователей современных комплексов программ. Для ее решения поставщикам таких комплексов, кроме программистов-кодировщиков, необходимо иметь системных аналитиков, архитекторов и топ-менеджеров проектов ПС, а также специалистов по комплексированию, испытаниям и обеспечению качества современных сложных программных продуктов. Они должны знать передовые индустриальные методы, технологии и международные стандар-

ты, поддерживающие и регламентирующие жизненный цикл ПС, а также инструментальные системы обеспечения качества, верификации, тестирования и сертификации программных средств. Для этого, прежде всего, необходима системотехническая квалификация предприятий и специалистов, которые берутся за создание крупных ПС высокого качества.

Задачи, связанные с неопределенностью применяемых понятий, требований и характеристик качества, характерны для сложных, наукоемких проектов комплексов программ. Однако многочисленные "спекуляции" разработчиков на их значениях приучили заказчиков не доверять рекламируемым достоинствам создаваемых ими программных продуктов. Во многих случаях контракты и предварительные планы на создание сложных программных средств и баз данных подготавливаются и оцениваются на основе неформализованных представлений заказчиков и разработчиков о требуемых функциях и характеристиках качества систем. Многочисленные провалы проектов ПС выявили необходимость формализации методов взаимодействия и, как следствие, обеспечения взаимного понимания разработчиков с заказчиком или с потенциальными пользователями создаваемого ПС. Такого понимания необходимо добиваться уже на начальном этапе реализации проекта с целью конкретизации его функций и уточнения требований к качеству.

Возрастание сложности и ответственности современных задач, которые решаются крупными системами, а также увеличение возможного ущерба от недостаточного качества комплексов программ значительно повысили актуальность освоения методов стандартизированного описания требований, а также оценивания характеристик качества на различных этапах жизненного цикла ПС. Обещания разработчиков в контрактах с заказчиками создать высококачественные ПС в согласованные сроки во многих случаях не выполняются по причине различий в понимании требуемого качества программ, а также неумения оценить ресурсы, необходимые для его достижения. Стратегической задачей в жизненном цикле современных систем стало обеспечение и совершенствование качества крупных программных продуктов при реальных ограничениях на использование ресурсов, выделяемых для их разработки.

Широкое многообразие классов и видов программ, обусловленное различными функциями систем, предопределяет формальные трудности, связанные с методами и процедурами доказательства соответствия ПС условиям контрактов и требованиям потребителей. По мере расширения сферы применения и увеличения сложности ПС выделились области, в которых ошибки или недостаточное качество программ или данных могут нанести ущерб, значительно превышающий положительный эффект от их использования. В этих критических случаях недопустимы аномалии и дефекты функционирования ПС при любых искажениях исходных данных, сбоях и частичных отказах аппаратуры и других нештатных ситуациях.

Задача формирования требований к функциональным характеристикам и качеству ПС [7, 8] включает анализ свойств, характеризующих качество его функционирования с учетом технологических и ресурсных возможностей разработчиков. При этом под качеством функционирования понимается совокупность свойств, которые определяют пригодность ПС обеспечивать надежное и своевременное представление информации потребителю для ее дальнейшего использования по назначению. Адекватный набор показателей качества программ зависит от функционального назначения и свойств каждого ПС. В соответствии с принципиальными особенностями программного продукта при его проектировании должны выбираться номенклатура и значения требований к характеристикам качества, необходимые для его эффективного применения пользователями, которые впоследствии отражаются в технической документации и в спецификации требований на конечный продукт. Для конкретных видов ПС доминирующие критерии качества при проектировании систем выделяются и определяются их функциональным назначением, а также требованиями технического задания.

Качество, которое проявляется в процессе использования, – это основное воспринимаемое пользователями свойство (характеристика) программной системы. Оно измеряется в терминах результата функционирования и применения программ. Качество ПС в среде пользователей может отличаться от качества в среде разработчиков, поскольку некоторые функции могут быть невидимы пользователю или не использоваться им. Пользователь оценивает только те атрибуты качества ПС, которые видимы и полезны ему в процессе практического применения.

Качество изменяется в течение жизненного цикла ПС. Этот факт означает, что его требуемое и реальное значение в начале ЖЦ почти всегда отличается от фактически достигнутого при завершении проекта и, как следствие, от качества поставляемой пользователям версии продукта. На практике важно оценивать качество программ не только в завершенном виде, но и в процессе их проектирования, разработки и сопровождения. Кроме того, оценки показателей качества могут быть субъективными и отражать различные точки зрения и потребности разных специалистов. Чтобы эффективно управлять качеством на каждом этапе ЖЦ, необходимо уметь определять и сочетать эти различные представления требуемого качества и его изменения. Требуемые характеристики качества ПС с различных позиций отражают их свойства и особенности и, в свою очередь, зависят от ряда факторов и ограничений. При системном анализе и проектировании программных средств необходимо определять и учитывать связи, влияние и взаимодействие следующих основных факторов, которые отражаются на их качестве:

– назначение, содержание и описание функциональных характеристик и атрибутов, определяющих специфические особенности целей, задач, свойств и сферы применения конкретного программного продукта – его функциональную пригодность;

– конструктивные характеристики качества, способствующие улучшению и совершенствованию назначения, функций и возможностей применения ПС;

– цели и особенности потребителей результатов оценивания характеристик качества ПС;

– внешние и внутренние негативные факторы, влияющие на достигаемое качество создания и применения ПС;

– доступные ресурсы, ограничивающие возможные величины реальных характеристик качества.

Задачи обеспечения полноты тестирования и испытаний комплексов программ. Многолетний опыт показал, что нельзя создать единственный, универсальный метод тестирования и следует применять упорядоченный набор значительно различающихся методов. Каждый метод отличается целевыми задачами тестирования, проверяемыми компонентами программы, методами оценки эффективности и результатами. Установлено, что только систематическое и совместное применение различных методов тестирования позволяет добиваться высокого качества функционирования сложных комплексов программ. Эти методы целесообразно автоматизировать с учетом обеспечения корректности требований к компонентам и модулям комплекса программ путем их верификации. Основная цель верификации ПС состоит в том, чтобы обнаруживать, регистрировать и устранять дефекты и ошибки, которые внесены во время последовательной разработки или модификации компонентов программ. Обычно она проводится сверху вниз, начиная от общих требований, заданных в техническом задании и/или спецификации на всю систему, до детальных требований на программные модули и их взаимодействие.

При выделении тестируемых маршрутов в программе разработчик должен указать критерий, по которому следует формировать такие маршруты. Кроме того, разработчику следует указывать метод для составления упорядоченного списка маршрутов, по которому надлежит планировать последовательность тестирования. Упорядочение маршрутов может производиться по длительности их исполнения или по вероятности реализации при случайных данных на входе программы. По этим маршрутам может рассчитываться полное число тестов и суммарная сложность тестирования структуры программы в соответствии с выбранным критерием выделения маршрутов, что позволяет оценивать реализуемость плана тестирования конкретного программного компонента или комплекса.

Задачи верификации корректности тестов необходимо решать для обеспечения высокого качества программного продукта параллельно с верификацией требований и программированием корректировок. Целесообразно в автоматизированном режиме верифицировать спецификации и сценарии тестов, отражающие методы и конкретные процедуры проверки реализации изменений требований. Тестовые спецификации могут использоваться для проверки согласованности, внутренней непротиворечивости и полноты реализации требований без исполнения программ. Для каждого требования к комплексу программ, к его архитектуре, функциональным компонентам и модулям должна быть разработана спецификация тестов, обеспечивающая проверку корректности, адекватности и возможности в по-

следующем реализовать тестирование компонента на соответствие этому требованию.

В результате совокупности спецификаций требований к тестам могут применяться при разработке как эталоны и вторая адекватная форма описания содержания программ для сквозной верификации спецификаций требований к тестам сверху вниз. Сами тесты должны подвергаться верификации на корректность соответствия исходным требованиям к компонентам текстов программ и данных разного уровня. Такие параллельные взаимные проверки спецификаций требований и текстов программ, а также спецификаций тестов способствуют выявлению и исключению большого числа вторичных дефектов и ошибок в ПС.

Задачи автоматизации испытаний качества продуктов в программной инженерии. Для обеспечения высокого качества крупных комплексов программ необходимы соответствующие проблемно-ориентированные интегрированные системы автоматизации тестирования, способные достаточно полно заменить испытания программ с реальными объектами внешней среды. Такие системы должны обеспечивать поэтапное и систематизированное формирование тестов разных типов, обеспечивающих проверку соответствия компонентов и комплексов программ всем требованиям, отраженным в контракте и в программе испытаний. Для сокращения неопределенностей и прямых ошибок при оценивании качества ПС необходимо до начала испытаний определять основные параметры внешней среды, при которых должен функционировать комплекс программ с требуемыми характеристиками при оценивании его качества эксплуатации. Средства автоматизации процессов тестирования и испытаний крупных комплексов программ реального времени должны обеспечивать:

– определение тестов – реализацию процесса тестирования разработчиком, включая ввод тестовых наборов, генерацию тестовых данных, ввод ожидаемых, эталонных результатов;

– управление тестами и участком программы между контрольными точками, для которого средство обеспечения тестирования может автоматически выполнять тестовые наборы;

– анализ и обработку тестовых результатов – возможность в автоматизированном режиме анализировать тестовые результаты, сравнивать ожидаемые и реальные результаты, сравнивать файлы, производить статистическую обработку результатов;

– анализ покрытия тестами исходного кода для обнаружения операторов, которые не были выполнены, процедур, которые не были вызваны, переменных, к которым не было обращения;

– анализ производительности тестируемой программы, включая загрузку центрального процессора, загрузку памяти, обращения к специфицированным элементам данных и/или сегментам кода, временные характеристики функционирования испытываемой программы.

Задачи автоматизированной имитации тестов и внешней среды в программной инженерии. Формирование тестов необходимо не только для оценивания достигнутых характеристик качества комплексов программ, но также для их комплексной отладки, квалификационного тестирования, испытаний и при создании версий программного продукта. При этом необходима, прежде всего, оценка эффективности и рентабельности создания имитационных моделей внешней среды для генерации тестов. Должна быть обеспечена адекватность моделей реальным объектам внешней среды и их способность достаточно полно покрыть тестами функционирование комплексов программ при испытаниях на соответствие предъявляемым к ним требованиям.

В крупных ПС принципиально невозможно исчерпывающее тестирование, гарантирующее полное отсутствие в них дефектов и ошибок. Число дефектов, обнаруживаемых в программах даже независимыми тестировщиками, имеет пределы. Проверка соответствия компонентов при верификации корректности сверху вниз от исходных требований к комплексу программ и тестирование покрытия компонентов на соответствие требованиям на каждом уровне их детализации может потребовать значительных вычислительных, трудовых (человеческих) и временных ресурсов. Реальные ограничения доступных ресурсов определяют число неустраненных дефектов, а также достижимую корректность компонентов и ПС в целом.

Задачи практического учета использованных ограниченных ресурсов и оценки момента времени, когда можно прекратить верификацию и тестирование программного средства. В процессе верификации и тестирования ПС необходимо определять, какая при этом будет достигнута корректность программ, способна ли она удовлетворить

заказчика и пользователя при эксплуатации программного продукта. Разработка сложных ПС, их верификация и тестирование требуют значительных ресурсов в течение всего жизненного цикла ПС. Наиболее критичным ресурсом при этом является допустимое время поэтапного выполнения этих процедур. Интенсивность выявления дефектов при тестировании программ практически экспоненциально убывает в зависимости от времени, затрачиваемого на их обнаружение, и, соответственно, возрастают интервалы времени между очередными проявлениями дефектов.

Задачи удостоверения достигнутого качества функционирования крупных программных продуктов и методов обеспечения их жизненного цикла базируется на сертификации аттестованными проблемно-ориентированными испытательными лабораториями. Применение сертифицированных систем качества на предприятиях-разработчиках должно гарантировать высокое, устойчивое качество процессов обеспечения жизненного цикла конечного программного продукта. Основой сертификации должны быть детальные и эффективные методики испытаний конкретных ПС, специально разработанные тестовые задачи и генераторы для их формирования, а также квалификация и авторитет испытателей. Для этого заказчики должны выбирать подрядчиков-исполнителей своих проектов, имеющих системы обеспечения качества программных средств и сертификаты, удостоверяющие реализацию этих средств предприятием-разработчиком.

Обеспечение и удостоверение качества сложных программных продуктов должно базироваться на проверках и испытаниях:

– технологий обеспечения ЖЦ программных средств, поддержанных регламентированными системами качества;

– готового программного продукта с полным комплектом адекватной эксплуатационной документации.

Тесная взаимосвязь качества разработанных комплексов программ с качеством технологии их создания и с затратами на разработку становится особенно существенной, если возникает необходимость получения конечного продукта с предельно высокими значениями показателей качества. Это обстоятельство привело в последние годы к существенному изменению объектов, методологии и культуры в области создания и совершенствования ПС. Непрерывный рост требований к качеству ПС стимулировал создание и активное применение международных стандартов и регламентированных технологий, автоматизирующих процессы их жизненного цикла, начиная с этапа инициирования проекта.

2. Проблема безопасности и сокращения рисков неблагоприятных событий в ходе применения программных продуктов

Задача неопределенности концепции и требуемых характеристик безопасности конкретных систем, включающих программные продукты. Эта задача должна учитываться заказчиками, пользователями и разработчиками в течение всего ЖЦ таких систем. Чем сложнее системы и выше к ним требования безопасности, тем менее определенными могут быть функции и характеристики их безопасности. Неопределенности начинаются с требований заказчиков, которые при формулировке технического задания и спецификаций не полностью формализуют и принципиально не могут достоверно и в полном объеме представить адекватный набор характеристик и значений безопасности, которые должны быть обеспечены при завершении проекта и предъявлении конечного продукта заказчику. Эти требования итерационно формируются, детализируются и уточняются по согласованию между всеми участниками проекта вследствие естественной ограниченности первичных исходных данных, и их изменения под влиянием объективных и субъективных воздействий факторов на последовательных этапах ЖЦ.

В требованиях технических заданий и реализованных проектах сложных систем систематически умалчиваются и/или недостаточно формализуются понятия и метрики безопасности программного продукта, а также информация о том, какими характеристиками они должны описываться, как их следует измерять и сравнивать с требованиями, отраженными в контракте, в техническом задании или в спецификациях. Кроме того, некоторые из характеристик безопасности зачастую вообще отсутствуют в требованиях заказчика и согласованных документах на систему и/или ПС, что приводит к произвольному их толкованию, учету или к пропуску при испытаниях. По этой причине аккуратное проведение процедур специфицирования, оценивания уровня безопасности подлежащих автоматизации систем, разрабатываемого программ-

ного продукта и обрабатываемой с его помощью информации – ключевая проблема обеспечения информационной безопасности и эффективного применения создаваемого ПС.

Непрерывно возрастающая сложность и, как следствие, уязвимость систем и ПС от случайных и умышленных негативных воздействий, выдвинули ряд проблем, связанных с обеспечением их безопасности в разряд важнейших – стратегических. Методы и средства их разрешения определяют принципиальную возможность и эффективность применения таких ПС в административных системах, в промышленности и в военной технике. При этом выделились два направления анализа и обеспечения, а именно – **информационной безопасности**, связанные в основном с защитой от умышленных, деструктивных воздействий на информационные ресурсы систем, и направление **функциональной безопасности** [11], обусловленной отказовыми ситуациями и потерей работоспособности систем и ПС вследствие проявления неумышленных, случайных дефектов и отказов программ, данных, аппаратуры или причин из внешней среды.

Задачи обеспечения функциональной безопасности при случайных, дестабилизирующих воздействиях и отсутствии злоумышленного влияния на системы, ПС или информацию баз данных. Эти задачи существенно отличаются от задач информационной безопасности. Функциональная безопасность объектов и систем зависит от отказовых ситуаций, негативно отражающихся на работоспособности и реализации ими основных функций, причинами которых могут быть дефекты и аномалии в аппаратуре, программах, данных или вычислительных процессах. Основными источниками отказовых ситуаций могут быть некорректные исходные данные, сбои и отказы в аппаратуре, дефекты или ошибки в программах и данных функциональных задач, которые проявляются при их исполнении в соответствии с назначением. Катастрофические последствия таких отказов в ряде областей применения программных систем могут превышать по результатам ущерб от злоумышленных деструктивных воздействий. Они имеют свою природу, особенности и характеристики.

Задачи достижения требуемого уровня функциональной безопасности систем, содержащих программные продукты реального времени. Они решаются путем использования современных регламентированных технологических процессов и инструментальных средств обеспечения их ЖЦ. Для систематической, координированной борьбы с угрозами безопасности ПС необходимы исследования потенциально возможных факторов, влияющих на безопасность в конкретных системах и комплексах программ. Такие исследования позволяют целенаправленно разрабатывать методы и средства обеспечения безопасности критических ПС различного назначения при достижимом на практике снижении числа дефектов проектирования и разработки.

Для создания безопасных систем и программных продуктов, прежде всего, следует формализовать их назначение, функции и основные характеристики. На этой основе должны разрабатываться общие требования к безопасности и к другим характеристикам качества ПС, а также к обработанной с помощью такого средства информации, адекватной назначению и функциям систем. Требования к функциям систем и ПС их автоматизации, а также к безопасности их функционирования должны соответствовать ресурсам, которые выделяются для их реализации с учетом допустимого ущерба – рисков [9] неблагоприятных событий при неполном выполнении предъявляемых к системе и ПС требований. В результате сложность программ и информационных систем, а также ресурсы, выделяемые для их реализации, становятся косвенными факторами, влияющими на выбор методов разработки, на качество и безопасность программных продуктов, которые должны быть созданы.

Разработку систем должны завершать комплексные испытания, удостоверяющие достигнутый уровень безопасности их применения с программными продуктами. Такие испытания должны предусматривать возможность совершенствования характеристик систем и ПС путем соответствующих корректировок программ. Повышения уровня безопасности можно добиваться также путем анализа и оперативного восстановления вычислительного процесса, программ и данных (рестарта) после обнаружения аномалий и отказов функционирования программного продукта. Этому может способствовать накопление, мониторинг и хранение данных о выявленных дефектах, сбоях и отказах в процессе исполнения программ и обработки данных. Испытания и измерения уровня безопасности систем и ПС связаны с необходимостью определения возможного мате-

риального или иного ущерба (риска) при отказовых ситуациях. На практике в ряде случаев ущерб от реализации угроз целесообразно характеризовать интервалами времени между их проявлениями, нарушающими безопасность применения ПС, или наработкой на опасные отказы, отражающиеся на безопасности с большим уровнем ущерба.

Задачи выделения факторов, определяющих риски неблагоприятных событий при использовании программных средств. Оценки качества программных средств в ЖЦ могут проводиться с двух позиций: с позиции положительной – эффективности и непосредственной адекватности их характеристик назначению, целям создания и применения, а также с негативной позиции возможного ущерба – риска реализации деструктивного воздействия с неблагоприятным исходом при создании или использовании программного продукта или системы. Характеристики качества, риски возникновения таких событий и процессов обычно тесно связаны. На них влияют отмеченные факторы, которые с разных сторон отражаются в свойствах систем или комплексов программ. Риски неблагоприятных событий проявляются как негативные последствия дефектов и ошибок разработки, функционирования и применения ПС. Они способны нанести ущерб системе, внешней среде или пользователю в результате отклонения характеристик объектов или процессов от заданных требованиями заказчика и согласованных с разработчиками. Риски ПС могут проявляться в процессах проектирования, разработки и сопровождения при изменении и развитии комплексов программ и при применении созданного программного продукта по прямому назначению.

Задачи оценки и управления рисками. Анализ рисков неблагоприятных событий (далее для краткости изложения – рисков) должен начинаться с исследования понятий, требований и функций, способствующих разработке, одобрению и применению заказчиком и пользователями конкретного программного продукта. При этом должны быть определены требования к характеристикам ПС и оценки возможного ущерба при их нарушении. Управление рисками предполагает понимание и оценку внутренних, внешних причин и реальных источников угроз, влияющих на установленные характеристики качества ПС, которые могут привести к значительному ущербу. Главной целью управления рисками является обнаружение, идентификация потенциально возможных неблагоприятных событий, контроль за ситуациями и факторами, которые приводят к негативным (рисковым) результатам применения комплекса программ. Такие меры должны отражаться на применении регламентированных процессов, в которых факторы и угрозы рисков систематически идентифицируются, оцениваются и корректируются. В ЖЦ программных средств ущерб может проявляться в результате:

– искажений или неполной реализации функций, которые определяются назначением или взаимодействием ПС с компонентами системы или внешней среды – недостатков и дефектов функциональной пригодности комплексов программ;

– недостаточных и не соответствующих требованиям конструктивных характеристик качества ПС при его функционировании и применении по прямому назначению;

– нарушений принятых ограничений на использование экономических, временных или технических ресурсов при создании и применении ПС.

Риски при выполнении проектов могут быть обусловлены нарушениями технологий или принятых на этапе разработки проекта следующих ресурсных ограничений: бюджета, планов, коллектива специалистов, инструментальных средств, выделенных на разработку ПС. Результирующий ущерб в совокупности зависит от величины и вероятности проявления каждого негативного воздействия. Одним из косвенных методов определения величины риска может быть оценка совокупных затрат, необходимых для ликвидации отрицательных последствий в программном продукте, системе или внешней среде, возможных в результате проявления конкретного рискового события. Это приводит к необходимости анализа рисков неблагоприятных событий, которые могут возникнуть в процессе функционирования ПС в условиях, различающихся:

– источниками и причинами угроз и опасного проявления рисков;

– классами, категориями, уязвимостью ПС и системы, вероятностью проявления и величиной последствий рисков;

– возможными и реализуемыми контрмерами для сокращения рисков и их эффективностью.

Задачи оценки источников рисков при реализации проектов программных продуктов. Одной из самых распространенных причин и опасных источников рисков являются ошибки при оценке масштаба (размера) проекта или программного продукта. Эти ошибки чаще всего бывают случайными – неумышленными, вследствие недостаточной компетентности заказчика или разработчика-поставщика. Величина оцененного и согласованного между заказчиком и разработчиком масштаба ПС непосредственно отражается:

– на бюджете и трудоемкости разработки и обеспечения всего ЖЦ программного продукта;

– на затратах времени и планируемых сроках на всех этапах жизненного цикла ПС;

– на потребностях в численности и квалификации специалистов-разработчиков для реализации проекта и создания программного продукта в соответствии с требованиями заказчика.

Эти три ключевых источника рисков обычно тесно связаны и могут изменяться по инициативе заказчика или разработчика в процессе подготовки проекта как в сторону увеличения, так и уменьшения, в соответствии с их интересами, целями и возможностями. Практически всегда основным фактором, определяющим значения и взаимосвязь этих важнейших характеристик и их рисков, при создании сложных ПС является оценка и отслеживание масштаба проекта, а также его согласование между заказчиком и разработчиком. При анализе и управлении рисками целесообразно выделять и рассматривать, прежде всего, наиболее характерные этапы жизненного цикла ПС: технико-экономическое обоснование проекта ПС; разработку требований и спецификаций; проектирование; кодирование; тестирование; документирование.

Задачи сокращения рисков программных средств. Они должны рассматривать с практической, промышленной, коммерческой и с юридической точки зрения. Практическая позиция отражает тот факт, что существует недостаточное понимание заказчиками, разработчиками и пользователями значения и необходимости анализа, учета и сокращения рисков в ЖЦ сложных ПС. С промышленной точки зрения подразумевается, что управление рисками ПС является инженерной дисциплиной, комплексы программ всегда содержат риски, которые априори невозможно достоверно предсказать и контролировать. Управление рисками ПС с коммерческих позиций – необходимый компонент на уровне проекта, предприятия и/или отрасли. Подобные риски могут привести к снижению объема продаж или к увеличению текущих расходов на маркетинг программного продукта. Юридическая ответственность и риск предполагают защиту от неправомерных правовых действий в случае появления дефектов и конфликтов в ходе проектирования, разработки, а также при нарушениях технологии выполнения проекта. Эти риски могут отражать неадекватное использование прав на интеллектуальную собственность.

Для того чтобы гарантировать высокое качество и не превысить допустимые риски комплексов программ, целесообразно готовить специалистов, ответственных за соблюдение промышленной технологии создания и совершенствование программ, за измерение и контроль характеристик качества, за сокращение рисков систем и ПС в целом и отдельных их компонентов на всех этапах жизненного цикла. Для планомерных, координированных действий по уменьшению рисков в процессе реализации проектов ПС необходимо учить таких специалистов анализу и оцениванию конкретных факторов, влияющих на риски при разработке проекта, на этапе функционирования программного продукта. Такие факторы могут быть обусловлены как реально существующими опасностями, так и потенциально возможными дефектами в программах и данных, которые создаются в рамках конкретного проекта.

3. Проблема экономики в программной инженерии

Массовое создание сложных и дорогих программных продуктов промышленными методами и большими коллективами специалистов вызвало необходимость их достоверного экономического анализа и оценки, четкой организации производства, планирования работ по затратам, этапам и срокам реализации [5]. Для решения этих задач еще в 1980-е годы начала формироваться новая область знания и инженерная дисциплина – экономика производства крупных программных продуктов. Необходимо было научить специалистов анализу и оцениванию конкретных факторов, влияющих на экономические характеристики проектов программных продуктов со стороны, реально существующих и потенциально возможных воздействий деструктивного характера, а

также объему ресурсов, которые необходимы для выполнения проектов. Определенные действия и опыт на этом направлении привели к появлению новой области экономической науки и практики – экономики производственных процессов и жизненного цикла сложных программных продуктов как части экономики промышленности и вычислительной техники в общей экономике сферы промышленного производства. Ее основной задачей является прогнозирование, эффективное управление, распределение ресурсов и экономное использование необходимых быстро возрастающих капиталовложений в производство сложных комплексов программ высокого качества и различного назначения.

Развитие этой области экономики связано с большими трудностями, которые характерны для новых разделов науки и техники, появляющихся на стыке различных областей знания. В настоящее время менеджеры и разработчики комплексов программ, как правило, не знают даже основ экономики промышленного производства сложной продукции. Экономисты современного производства при этом не представляют физической сути и свойств объектов разработки – программных продуктов, а также особенностей экономики технологических процессов их проектирования, производства и применения.

Крупные программные продукты являются одними из наиболее сложных объектов, создаваемых человеком. В процессе их производства творчество специалистов направлено на поиск новых методов, алгоритмов, альтернативных решений и способов, позволяющих, во-первых, сформировать и декомпозировать требования к создаваемому продукту, а во-вторых, добиться выполнения заданных требований. Индустриализация производства комплексов программ позволяет автоматизировать нетворческие, технические и рутинные операции, а также облегчить творческие процессы за счет селекции, обработки и отображения информации, необходимой для принятия творческих решений. В производстве программ неуклонно повышаются размеры и сложность создаваемых продуктов, что приводит к возрастанию доли творческого труда, который затрачивается на производство условной единицы от общего объема программ. Даже при сокращении суммарных затрат на разработку программных компонентов за счет автоматизации нетворческого труда, все более определяющей для экономических характеристик создания сложных программных продуктов становится доля затрачиваемого на это творческого труда. Возрастают при этом и требования к творческим способностям при отборе и обучении специалистов для работы в этой сфере.

Задачи развития и освоения экономики программной инженерии. Приступая к разработке крупных проектов, их руководители прежде всего должны оценить целесообразность их создания, возможную эффективность применения готового продукта, а также окупятся ли затраты на его разработку и использование. В связи с этим подобные проекты традиционно должны начинаться с анализа и разработки технико-экономического обоснования предстоящего жизненного цикла и эксплуатации предполагаемого продукта. Заказчику проекта необходимо оценивать реальную потребность в его создании и возможную конкурентоспособность, а потенциальному разработчику предполагаемого продукта следует провести оценку реализуемости проекта, принимая во внимание условия и ресурсы, которые предлагаются для его выполнения заказчиком.

Следствием недостатков или отсутствием технико-экономического обоснования проектов новых ПС являются острые конфликты между заказчиками и разработчиками. Зачастую разработчики ПС не в состоянии привести заказчику или руководителю проекта достаточно убедительные доказательства того, что выполнение выдвигаемых ими требований невозможно, например, при предложенных ограниченных значениях бюджета и сроков. Это, в свою очередь, может привести к завышенной оценке выгод новой программной разработки, к недооценке роли других конкурирующих предложений при заключении контрактов на разработку и вследствие этого – к неизбежным перерасходам средств и к снижению качества ПС. Руководители отдельных проектов ПС зачастую не в состоянии достаточно обоснованно определять, сколько времени и трудозатрат потребуется для проведения работ на каждом этапе программной части проекта системы. Такое положение дел, как правило, приводит к тому, что программная часть проекта системы с самого начала выходит из-под экономического контроля. Как следствие вполне вероятно, что будут задержки с выполнением отдельных работ и завершением проекта всей системы в поставленные сроки с заданным качеством.

Задача экономического прогнозирования развития проектов в программной инженерии. Она за-

ключается, прежде всего, в помощи специалистам определять:

• целесообразно ли продолжать работы над конкретным проектом ПС в направлении детализации требований, функций и технико-экономических характеристик или следует его прекратить по причине отсутствия недостаточных ресурсов, специалистов, времени или возможной стоимости и трудоемкости разработки;

• следует ли при наличии достаточных ресурсов провести маркетинговые исследования для определения рентабельности выполнения проекта ПС в полном объеме, создания программного продукта для его поставки на рынок информационных технологий;

• достаточно ли полно и корректно формализованы требования к проекту ПС, на основе которых проводились расчеты технико-экономических показателей, или их следует откорректировать и выполнить повторный анализ с уточненными исходными данными;

• есть ли возможность применить имеющиеся на рынке готовые повторно используемые компоненты ПС, а если да, то в каком относительном размере от всего комплекса программ и рентабельно ли их применять в конкретном проекте ПС или проект в целом целесообразно разрабатывать заново.

Задачи анализа экономики производства программных продуктов. Экономические характеристики реальных завершенных разработок собираются, накапливаются и обрабатываются с начала 1980-х годов в разных отечественных организациях и за рубежом. Они позволили получать и прогнозировать основные обобщенные экономические показатели процессов производства комплексов программ. При этом обычно рассматривался полный технологический процесс производства от начала подготовки технического задания до завершения испытаний базовой версии продукта. Учитывались все категории специалистов, участвующих в создании программ и обеспечивающих разработку, а также все виды работ, связанные с созданием программного продукта на определенном для этого интервале времени.

Наиболее полно и подробно основные закономерности и влияние факторов на экономические характеристики процессов разработки представлены в 2000 г. под названием COCOMO II [10]. В этой модели на основе анализа более 160 реальных проектов разработки комплексов программ различной сложности определены рейтинги влияния выделенных факторов на основные экономические характеристики. Кроме того, в 1988 г. опубликованы результаты анализа экономических характеристик (комплексный проект ПРОМЕТЕЙ) на основе обобщения результатов разработки свыше 250 отечественных проектов сложных программных продуктов. В общем случае для оценки экономических характеристик новых проектов необходимы следующие исходные данные:

– обобщенные характеристики использованных ресурсов и экономические показатели завершенных разработок – прототипов, а также оценки влияния на их характеристики различных факторов объекта и среды разработки;

– реализованные и обобщенные перечни выполненных работ, а также реальные графики проведенных ранее разработок различных классов комплексов программ;

– цели и содержание частных работ в процессе создания сложных комплексов программ и требования к их выполнению для обеспечения необходимого качества продукта в целом;

– структура и содержание документов, являвшихся результатом выполнения частных работ.

В процессе разработки сложных программных продуктов уточняются и детализируются спецификации их требований, функции, структура и характеристики компонентов. По этой причине на начальных этапах, что особенно принципиально для новых проектов, трудно точно спланировать все последующие этапы. В результате планирование проводится итерационно в соответствии с последовательным повышением достоверности и точности сведений об объекте и среде разработки. Отсюда принципиальной особенностью разработки комплексов программ является активное участие руководителей и заказчиков проекта в составлении планов на базе исследованных характеристик прототипов завершенных разработок и их личного опыта. Организованное, контролируемое и методичное отслеживание динамики проектирования и производства, а также изменений в жизненном цикле программ и данных, их разработка при строгом учете и контроле ресурсов и каждого изменения является одной из важнейших задач, решение которой гарантирует эффективное, поступательное развитие каждого крупного комплекса программ и системы.

Задачи организации экономически эффективного производства программных продуктов. Стратегической проблемой в жизненном цикле совре-

менных систем стало обеспечение и совершенствование качества производства сложных программных продуктов при реальных ограничениях на использование доступных ресурсов. Для каждого крупного проекта, выполняющего ответственные функции, необходимо прогнозировать необходимые для его реализации ресурсы, разрабатывать и применять комплексную систему оценки качества, специальные планы и программу, методологию и инструментальные средства, обеспечивающие требуемые качество, надежность и безопасность функционирования программного продукта. Для этого при подготовке промышленных технологий, методик производства и испытания конкретных продуктов следует использовать современные методы и стандарты программной инженерии. Они должны однозначно отражать степень удовлетворения исходных требований заказчика и пользователей.

Базой эффективного управления проектом создания крупного программного комплекса должен быть план, в котором задачи исполнителей частных работ согласованы между собой и с выделяемыми на их выполнение ресурсами по результатам и срокам их достижения. По мере уточнения исходных данных об объекте разработки, внешней среде применения и ресурсах, в процессе системного анализа и проектирования возрастает достоверность планирования, которое должно проходить следующие этапы:

– обследование объектов и среды проектирования для предварительной формализации целей, назначения и задач проекта;

– первичное прогнозирование возможных характеристик и требований к программному продукту на базе обобщения данных по ранее реализованным подобным прототипам и создание концепции проекта;

– управление детализацией и реализацией плана производства, его оперативной корректировкой и перераспределением ресурсов в соответствии с особенностями развития компонентов программного комплекса;

– обобщение и накопление результатов планирования и управления конкретным проектом для использования впоследствии этих данных в качестве прототипов при производстве программных продуктов.

Задача создания и представления специалистам современных методов экономического анализа, оценивания и прогнозирования необходимых ресурсов при производстве комплексов программ. Внимание должно быть сосредоточено на концептуальной основе распределения затрат труда в процессе разработки компонентов и комплексов программ, на факторах, определяющих реальные трудозатраты и другие экономические характеристики, а также на исследовании их в ранее реализованных современных разработках. В ЖЦ сложных комплексов программ для обеспечения их высокого качества целесообразно выделять специалистов, ответственных за анализ, оценивание и прогнозирование экономических характеристик производства, за соблюдение промышленной технологии создания и совершенствование программных продуктов, за измерение и контроль затрат, качества комплексов программ в целом и их компонентов. Для этого необходима подготовка и воспитание квалифицированных специалистов в области экономики и производства сложных программных комплексов, их обучение методам и современной программистской культуре промышленного создания крупных программных продуктов высокого качества.

СПИСОК ЛИТЕРАТУРЫ

1. **Липаев В.В.** Программная инженерия. Методологические основы. М.: 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.

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

6. **Липаев В.В.** Анализ и сокращение рисков проектов сложных программных средств. М.: СИНТЕГ, 2005.

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

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

9. **ГОСТ** Р 51897–2002. Менеджмент риска. Термины и определения. М.: Изд-во стандартов, 2003.

10. **Boehm B.** Software Cost Estimation with Cocomo II. Pearson Education, 2000.

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

УДК 004.41;054; 519.257

А.Е. Колоденкова, канд. техн. наук, доц., зам. зав. кафедрой,
Уфимский государственный авиационный технический университет

E-mail: anna82_42@mail.ru

Анализ жизнеспособности – важная стадия жизненного цикла инновационных программных проектов

Обсуждаются концептуальные основы проблемы анализа жизнеспособности инновационных программных проектов. Обобщены и систематизированы различные точки зрения на оценку жизнеспособности проектов как отечественных, так и зарубежных специалистов. Предложен формальный подход к статистической оценке жизнеспособности программных проектов, основанный на сравнении альтернативных вариантов реализации проекта по критерию вероятного выполнения совокупности работ за выделенное на проект время.

Ключевые слова: *инновационный программный проект; жизнеспособность проекта; виды, критерии и показатели жизнеспособности проекта; задачи и методы оценки жизнеспособности проекта*

Одной из особенностей современных сложных программных проектов (ПП) является их разработка и реализация в условиях риска, возникающего в силу неопределенности факторов внутренней и внешней среды проекта, которые могут привести либо к ухудшению качественных показателей проектируемого программного продукта, либо к увеличению затрат и/или нарушению сроков осуществления проекта, либо к его провалу. В связи с этим все большую актуальность приобретает проблема анализа жизнеспособности ПП, являющаяся важнейшей стадией жизненного цикла проекта, которая связана с риск-менеджментом и направлена на снижение рисков проекта, возникающих на ранних стадиях его разработки [1–3].

Анализу и оценке жизнеспособности различных научно-исследовательских и опытно-конструкторских проектов посвящено значительное число работ, например [4–7]. Однако проблема анализа жизнеспособности ПП до сих пор остается открытой, а известные многофакторные модели экспертной оценки его жизнеспособности не позволяют учесть мнения независимых экспертов и обеспечить необходимый уровень объективности оценок. В настоящей работе рассмотрены некоторые актуальные аспекты проблемы анализа жизнеспособности ПП, обобщены и систематизированы различные точки зрения на ее оценку, а также предложен формальный подход к ее статистической оценке.

Работа выполнена в рамках исследований, проводимых при поддержке РФФИ (грант РФФИ № 10-08-00359-а).

"Кризис программного обеспечения" и проблема жизнеспособности инновационных программных проектов

Объективная потребность контролировать процесс разработки ПП, прогнозировать и гарантировать стоимость, а также сроки и качество его разработки привела к появлению ***программной инженерии*** (*software engineering*) [8] – области компьютерных наук и технологий, связанной с инду-

стриальным производством высококачественных, экономичных и надежных программных продуктов для решения определенной проблемы в виде комплекса взаимосвязанных программ и сопроводительной технической документации, необходимой для его эксплуатации (установке, настройке и использованию).

Несмотря на определенные успехи, достигнутые программной инженерией на современном этапе развития, реализация подавляющего большинства крупных ПП не укладывается в запланированные сроки, характеризуется низкой производительностью, существенным отставанием от графика, а созданные в результате программные продукты часто не вполне отвечают требованиям заказчиков-пользователей. Более того, все чаще программные проекты оказываются просто проваленными [8, 9]. Так, например, согласно статистике "Хаос" о состоянии дел в программной индустрии в 2008 г., опубликованной компанией *Standish Group* [10], из 30 000 проектов по разработке программного обеспечения 32 % проектов завершились успешно, 44 % проектов завершились с опозданием (превысили бюджет, выпали из сроков и пр.) и 24 % проектов полностью провалились. Все это является проявлением "кризиса программного обеспечения", в условиях которого, как отмечает Э. Йордан, "безнадежные проекты являются нормой, а не исключением".

В современной государственной научно-технической политике развитых стран важную роль играют так называемые инновационные проекты [11], которые реализуются в виде крупных межотраслевых проектов по созданию, освоению и распространению высоких технологий, способствующих кардинальным изменениям в технологическом базисе экономики, а также по развитию фундаментальных исследований, научно-техническому обеспечению социальных программ и международного сотрудничества. В настоящей работе под ***инновационным программным проектом*** (ИПП) понимается разновидность инновационного проекта, рассматриваемая как интеллектуальная деятельность коллектива исполнителей, направленная на достижение заранее определенного результата/цели ПП при заданных ресурсных и временных ограничениях с учетом требований к его качеству. При этом результатом ИПП является создание уникального инновационного программного продукта, который можно запускать, тестировать, исправлять и развивать [3]. В качестве типовых примеров таких ПП можно привести программные продукты компании *Primavera Systems*, *Inc.*, предназначенные для создания автоматизированных систем управления проектами. Так, ПП *SureTrak* предназначен для управления мелкими и средними проектами, а ПП *Primavera Project Planner Professional* (P4) предназначен для управления большими проектами или группами проектов. Важно заметить, что программные продукты компании *Primavera Systems*, *Inc*. для управления проектами могут быть настроены для решения любой проектно-ориентированной задачи, для работы в любой сфере деятельности (будь то информационные технологии, разработка нового продукта и др.).

Отличительной особенностью ИПП является их направленность в будущее, предвидение которого сопряжено с неопределенностью и, следовательно, с различными видами рисков, т.е. возникновением ситуаций, при которых поставленные в проекте цели могут быть не достигнуты полностью или частично. Успешность выполнения ИПП и возможные его риски можно прогнозировать на начальных стадиях жизненного цикла проекта, однако такой прогноз существенно усложняется наличием следующих проблем:

- *Неполнота исходных данных для проектирования*, не позволяющая дать точную оценку жизнеспособности проекта и, прежде всего, срокам его выполнения. Здесь следует отметить часто имеющее место игнорирование накопленного опыта исполнителей: показатели удачно реализованных проектов не обобщаются и не используются в качестве исходной базы для управления новыми проектами. Как заметил ДеМарко (T. DeMarco) [12], "риски, которые мы должны в первую очередь анализировать в новом проекте, – это проблемы проектов вчерашних".
- *Изменение требований, сроков и объема выделяемых ресурсов на проектирование*, прямо связанное с увеличением риска осуществления проекта с должным качеством.
- *Нехватка или высокая занятость требуемых исполнителей проекта*, приводящая к увеличению риска затягивания сроков осуществления проекта. Действительно, согласно мнению гендиректора компании *Primavera Systems*, *Inc*. Коппельмана (J. Koppelman), "ключевое условие успешной реализации проекта – тщательный подбор людей, обладающих требуемой квалифика-

цией" на основе принципа "синхронизации проектов и человеческих усилий".

◆ *Слабая структурированность теоретических и фактических знаний о проекте*, не позволяющая дать удовлетворительный прогноз рисков проекта на достаточно большом промежутке времени с применением неформализованных методов на основе только собственного опыта и интуиции.

Основные понятия, критерии и показатели жизнеспособности инновационных программных проектов

При создании масштабных и сложных ИПП большую роль играет анализ жизнеспособности проекта. *Анализ жизнеспособности проекта* является важнейшей стадией жизненного цикла ПП, позволяющей оценивать возможность осуществления проекта, формулировать его общественную и коммерческую значимость, а также выявлять ключевые факторы успеха его реализации. В настоящей работе под *жизнеспособностью проекта* понимается наличие необходимых ресурсов для осуществления и условий для реализации проекта.

Ввиду сложности протекания процесса осуществления и реализации ПП дать универсальный подход к его разделению на стадии весьма затруднительно. В связи с этим В.В. Липаев подчеркивает, что существующие стандарты ГОСТ ЕСПД, предусматривающие стадии и этапы (создание, сопровождение и совершенствование) жизненного цикла ПП "отражены недостаточно, а многие их положения устарели". Решая данную задачу, исполнители проекта должны руководствоваться своей ролью в проекте, накопленным опытом и конкретными условиями его выполнения. Здесь можно согласиться с мнением И.И. Мазура: "универсального подхода к разделению процесса реализации проекта на фазы не существует". В настоящей работе за основу принята структурная схема жизненного цикла ИПП, представленная на рис. 1, где стадия жизнеспособности проекта выделена штриховой линией.

Здесь весь жизненный цикл проекта разделяется на концептуальное и материальное проектирование программного продукта.

Концептуальное проектирование – это процесс разработки концепции программного продукта, включающий две стадии: *формулирование проекта* (анализ идеи, выбор целей и постановка задач) и *анализ жизнеспособности проекта* (сбор, анализ и структурирование информации о проекте в форме, позволяющей принимать решение о возможности его осуществимости и способах реализуемости).

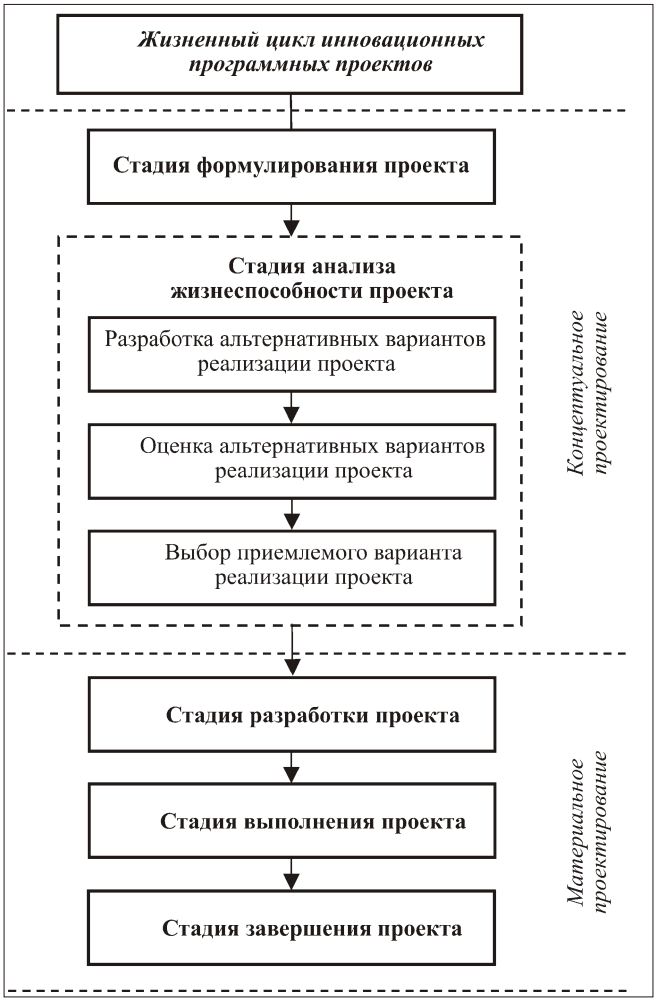


Рис. 1. Структура жизненного цикла ИПП

Материальное (*рабочее или техническое*) *проектирование* – это процесс физической реализации программного продукта, который включает три стадии: *разработка проекта* (происходит детальное планирование работы над проектом "от и до"), *выполнение проекта* (происходит непосредственная работа по реализации проекта) и *завершение проекта* (происходит полная реализация задуманного проекта, проводится анализ результатов).

Стадия анализа жизнеспособности ИПП включает следующие три этапа:

Рис. 2. Классификация видов, критериев и показателей жизнеспособности ИПП

• *разработка альтернативных вариантов реализации проекта* (разработка возможных способов достижения поставленной цели);

• *оценка альтернативных вариантов реализации проекта* (расчет показателей, характеризующих достоинства и недостатки альтернатив);

• *выбор приемлемого варианта реализации проекта* (выбор альтернативы с наиболее благоприятными последствиями).

Для оценки жизнеспособности ИПП предлагается классификация видов, критериев и показателей жизнеспособности проекта, представленная на рис. 2.

Виды жизнеспособности проекта. В процессе планирования и организации инновационной деятельности проводят оценку жизнеспособности проекта, которая включает следующие виды его анализа: *технический* (рассмотрение альтернативных вариантов исполнения проекта и оценки их реализуемости, сроков осуществления проекта в целом и всех его стадий, составление календарных планов), *финансовый* (определение соотношения финансовых затрат и результатов, обеспечивающих необходимую норму доходности), *экономический* (отражение эффективности проекта с точки зрения интересов всего общества в целом, например, поступлений средств в различные бюджеты), *коммерческий* (определение каналов продвижения программных продуктов на рынок, анализ и оценка конкурентов).

Критерии жизнеспособности проекта. Выбор проектов с помощью критериев направлен на выявление общего представления о проекте (его достоинствах и недостатков). При составлении перечня критериев необходимо использовать лишь те из них, которые соответствуют приоритетным целям и задачам. К ***критериям выбора проектов*** относятся: *научно-технические* (перспективность используемых научно-технических решений, применения полученных результатов), *производственные* (данные о наличии персонала, доступности сырья, материалов), *финансово-экономические* (ожидаемая норма прибыли, срок окупаемости), *маркетинговые* (оценка рыночного потенциала).

Показатели жизнеспособности проекта. Критерии выбора проектов могут оцениваться прямыми и косвенными показателями. *Прямые показатели* непосредственно характеризуют критерий жизнеспособности проекта (время разработки, число исполнителей и потребителей и др.). *Косвенные показатели* используются в случае, когда невозможно получить значения прямых показателей проекта (емкость рынка, стоимость работ, вероятность успеха и т.д.).

Следует особо подчеркнуть многокритериальность понятия жизнеспособности ИПП. Дейст-

вительно, квалифицированная оценка жизнеспособности проекта является векторной и предполагает достаточную широту охвата учитываемых показателей жизнеспособности (как прямых, так и косвенных), которые обозначим унифицированно через $_1$, $_2$, ..., $_n$:

$$(\ _1,\ _2, ..., \ _n),$$

а их заданные значения – через $_1$, $_2$, ..., $_n$:

$$(\ _1,\ _2, ..., \ _n).$$

Тогда требования, предъявляемые к жизнеспособности проекта, определяются его функциональным назначением. При этом возможны следующие *три способа формулировки* данных требований:

◆ в виде *допустимого уровня жизнеспособности* проекта, т.е. в требовании непревышения показателей жизнеспособности их заданных допустимых значений:

$$_i \quad _i, \ i \quad \overline{1, n};$$

◆ в виде *заданного уровня жизнеспособности* проекта, т.е. в требовании точного или приближенного равенства показателей жизнеспособности их заданных допустимых значений:

$$_i \quad _i, \ i \quad \overline{1, n};$$

◆ в виде *оптимального уровня жизнеспособности* проекта, т.е. в требовании экстремальности (возможной предельности) значений показателей жизнеспособности проекта:

$$_i \quad \text{extr}, \ i \quad \overline{1, n}.$$

Следует заметить, что на протяжении всей истории развития программного инжиниринга отмечалось стремление к предельному совершенствованию тех или иных показателей жизнеспособности ПП, следуя принципу: "Мы не настолько богаты, чтобы проектировать неоптимально". В связи с этим наибольшую популярность в практике ИПП получают именно постановки задачи обеспечения оптимальных (максимальных, либо минимальных в зависимости от особенностей проекта) показателей жизнеспособности проекта.

Современные задачи и методы оценки жизнеспособности инновационных программных проектов

На разных стадиях и этапах жизненного цикла ИПП возникает широкий спектр задач оценки жизнеспособности проекта, которые являются следствием манифеста программной инженерии: "максимизация качества проекта при минимуме затрат". Очевидно, что успешное решение данных задач обеспечивает успешное выполнение проекта.

В общем случае задача оценки жизнеспособности ПП является задачей весьма сложной и современная программная инженерия предлагает все новые подходы к ее решению, однако до сих пор нет надежной универсальной методики, дающей гарантированный результат. Как подчеркивает Ф. Брукс: "слабо развиты наши методы оценок. В сущности, они отражают молчаливое и совершенно неверное предположение, что все будет идти хорошо". На основе систематизации обзора литературных источников на рис. 3 представлена классификация существующих задач оценки жизнеспособности ИПП.

Задачи исследования операций, связанные с поиском путей рационального использования имеющихся ресурсов для реализации поставленной цели проекта. Здесь можно выделить следующие задачи:

- *задачи управления запасами*, связанные с созданием запаса материальных ресурсов или предметов потребления при проектировании в целях удовлетворения спроса на заданном интервале времени;
- *задачи распределения ресурсов*, связанные с необходимостью выполнения определенного набора работ при проектировании с учетом ограниченных ресурсов;
- *задачи упорядочивания*, связанные с формированием очередности работ при проектировании в порядке их значимости.

Задачи программного инжиниринга, связанные с оценкой жизнеспособности проектов в условиях ограниченных временных ресурсов. Здесь для большинства проектов ключевыми являются следующие оптимизационные задачи:

- *Оптимизация упущенной выгоды* [13–15] . В связи с тем, что в условиях дефицита финансовых ресурсов нет возможности вести одновременно календарные планы реализации всех проектов, то необходимо выбрать первоочередные проекты,

Рис. 3. Классификация задач оценки жизнеспособности ИПП

реализация которых обеспечивает наибольший эффект. Реализация остальных проектов отодвигается на более поздние сроки и ставится задача разработать план реализации программы, при котором упущенная выгода минимальна.

Так, например, в работе [13] рассматривается задача распределения ресурсов по проектам таким образом, чтобы минимизировать упущенную выгоду. Для ее решения предложен эвристический алгоритм, в основе которого лежит упорядочение проектов по убыванию приоритетов $q_i \quad c_i / w_i$ (c_i – доход после завершения проекта, w_i – объем его финансирования).

• *Оптимизация проектов по стоимости и срокам* [16–18]. Общая стоимость проекта зависит от стоимости выполнения каждой работы, а также от любых дополнительных переменных или постоянных расходов. Снижение продолжительности выполнения некоторых работ можно осуществлять с помощью дополнительных ресурсов. В связи с этим ставится задача минимизации стоимости и сроков выполнения проекта.

К примеру, в работе [18] рассматривается задача нахождения условий, при которых существует решение задачи минимизации сроков проекта, отличное от "нормального" (все требования сначала будут собраны, а затем реализованы) выполнения проекта. Решения данной задачи предполагает предварительную оценку сложности проекта и сводится к решению систем неравенств.

• *Оптимизация затрат при формировании команды исполнителей* [19, 20]. Поскольку скорость вывода новых программных продуктов на рынок в современных условиях является конкурентным преимуществом, то, естественно, возникает задача формирования такой команды исполнителей проекта, которая бы осуществила выполнение всех работ в заданные сроки, и при этом затраты на оплату труда были бы минимальны.

Например, в работах [19, 20] рассматривается задача минимизации затрат при формировании команды. Предложенная задача представляла собой классическую задачу о "ранце" (класс задач линейного программирования), для решения которой используется метод ветвей и границ.

• *Ранжирование работ, проектов* [21, 22]. В условиях ограниченности финансовых ресурсов для предприятий крайне важно реализовывать наиболее эффективные проекты, поэтому на первом этапе необходимо выстроить проекты в порядке убывания их значимости для того, чтобы далее производить отбор. В связи с этим необхо-

димо разрабатывать методики, в которых были бы прописаны показатели и принципы, лежащие в основе ранжирования.

В работе [21] рассматривается задача выбора наиболее приоритетных работ в рамках проекта $U(M)$ u_i (M – подмножество работ; u_i – полезность i-й работы) при заданном ограничении на время выполнения общего набора работ t_i T (t_i – время выполнения i-й работы; T – общее время выполнения проекта). Для решения данной задачи используются эвристические алгоритмы метода дискретной оптимизации с ранжированием полезности работ.

Безусловно, указанные задачи могут быть положены в основу технологии компьютерной оценки параметров жизнеспособности ИПП и принятия решений по дальнейшему управлению проектом.

Следует заметить, что непосредственные усилия исполнителей (их трудозатраты) обеспечивают большую часть стоимости разработки программного продукта, причем методы их анализа дают оценки, которые впоследствии могут быть преобразованы в продолжительность или стоимость проекта. В настоящее время на практике при анализе трудозатрат проекта широко применяются экспертные оценки. Однако данный подход сопряжен с рядом проблем: основания для получения оценки не являются явными; трудно находить квалифицированных экспертов для каждого нового проекта и др. В связи с этим было разработано множество различных моделей анализа трудозатрат, которые варьируются от моделей, основанных на эмпирических данных (модель COCOMO II [2, 7, 8]) до чисто аналитических моделей. *Эмпирические модели* используют данные предыдущих проектов, чтобы оценить текущий (анализируя закономерности, прослеживаемые с помощью имеющихся баз данных предыдущих проектов).

В работах по программному инжинирингу, например в [2, 23], подчеркивается, что одним из условий повышения эффективности оценки жизнеспособности ПП является использование формально математических методов.

Вероятностно-статистический подход к оценке жизнеспособности инновационного программного проекта на основе метода "Обращение–Сдвиг"

Рассмотрим статистический подход к оценке жизнеспособности ИПП, предложенный автором в работе [24]. Идея подхода заключается в сравнительном анализе альтернативных вариантов сетевых графиков выполнения работ в рамках проекта на основе специально сформированного *скалярного статистического показателя его жизнеспособности*, который для i-го варианта проекта представляется в виде вероятности выполнения совокупности работ за выделенное время:

$$^{(i)} \quad \min_i\{p_j^{(i)}\},\ i \quad \overline{1,m},\ j \quad \overline{1,k_i},$$

где m – число вариантов реализации проекта, k_i – число видов работ в i-м варианте реализации проекта, $p_j^{(i)}$ – вероятность выполнения j-й работы в i-м варианте реализации проекта, представленной на i-м сетевом графике при заданных ограничениях на время $T_j^{(i)}$ выполнения каждой работы. При этом полагаем, что вероятность $p_j^{(i)}$ зависит от времени $T_j^{(i)}$, а также от исполнителей j-й работы i-го варианта реализации проекта. Данный подход позволяет на стадии анализа жизнеспособности проекта принять решение об отказе от анализируемого варианта проекта, о запуске его в производство либо о генерации его новых альтернативных вариантов.

Пусть задана производительность исполнителей T_n $_{j,n}(V_j^{(i)})$ (T_n – время выполнения работ n-м исполнителем), $i \quad \overline{1,m}$, $j \quad \overline{1,k_i}$, а также варианты сетевых графиков выполнения работ с указанием объема для каждой работы $V_j^{(i)}$. Для расчета скалярной характеристики $^{(i)}$, описывающей жизнеспособность i-го альтернативного варианта реализации проекта, предлагается алгоритм, который содержит следующие четыре этапа:

♦ *на первом этапе* рассчитываются ожидаемые времена выполнения работ $T_j^{(i)}$ для различных m вариантов сетевых графиков выполнения работ с указанными объемами работ $V_j^{(i)}$ и производительностями исполнителей;

♦ *на втором этапе* рассчитываются вероятности выполнения работ $p_j^{(i)}$ для различных m вариантов сетевых графиков с ожидаемыми временами выполнения работ $T_j^{(i)}$;

♦ *на третьем этапе* формируется скалярная характеристика жизнеспособности программного проекта $^{(i)}$ для каждого i-го альтернативного варианта реализации проекта, которая определяется минимальной вероятностью выполнения работ $^{(i)}$ $\min_i p_j^{(i)}$;

♦ *на четвертом этапе* осуществляется принятие решений на основе анализа множества характеристик $^{(i)}$, i $\overline{1, m}$.

Обратимся к первым двум этапам алгоритма решения задачи анализа жизнеспособности проекта. Ожидаемые времена выполнения работ и вероятности выполнения для каждой работы рассчитываются по распределениям вероятности объема, времени окончания работ и производительностям исполнителей проекта. Последние находятся на основе известных архивных данных, полученных по предыдущим аналогичным проектам, которые, как правило, собираются, накапливаются и обрабатываются в проектных организациях. При этом следует отметить, что для нахождения производительности исполнителей проекта используется непараметрический метод, описанный в работе [25].

Положим, что в контексте рассматриваемой задачи роль времени выполнения работ $T_j^{(i)}$ играет случайная величина X, а объемов работ $V_j^{(i)}$ – случайная величина Y, значения которых будем обозначать соответственно через x и y. Считаем, что производительность исполнителей T_n $_{j,n}$ $(V_j^{(i)})$, т.е. зависимость y = (x) – известная монотонная, возрастающая функция.

Тогда нетрудно заметить, что первые два этапа алгоритма решения поставленной задачи сводятся к нахождению по имеющимся выборочным значениям y случайной величины Y и известной функциональной зависимости y (x) закона распределения $F_1(x)$ случайной величины X. Данная задача, являющаяся обратной задачей по отношению к известной классической прямой задаче оценивания закона распределения функции случайного аргумента [26], в литературных источниках не рассматривалась. Для ее решения,

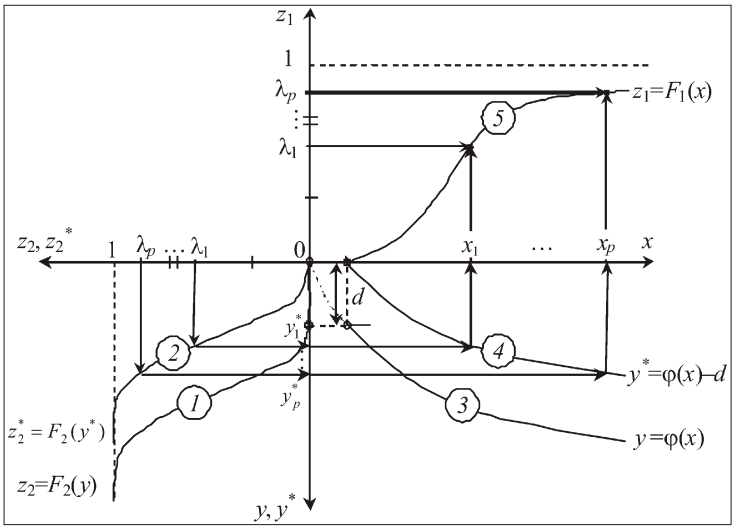


Рис. 4. Графическая иллюстрация метода "Обращение–Сдвиг"

т.е. построения искомого закона распределения $F_1(x)$ случайной величины X, предлагается использовать метод "Обращение–Сдвиг", предложенный автором в работе [24] и состоящий из следующих шести шагов (графическая иллюстрация метода представлена на рис. 4):

• *на первом шаге* рассчитываются математическое ожидание $M[y]$ и среднеквадратическое отклонение $[y]$ величины Y по ее выборочным значениям y;

• *на втором шаге* рассматривается условие $M[y]/[y]$ 3,5, при выполнении которого осуществляется сдвиг функции $z_2 = F_2(y)$ (кривая *1*) на величину d;

• *на третьем шаге* осуществляется построение функции z_2 $F_2(y)$ (кривая *2*) с учетом сдвига по преобразованным данным y_i y_i d (i $\overline{1, N}$, N – объем выборки) и осуществляется сдвиг функции y (x) (кривая *3*) на величину d;

• *на четвертом шаге* формируется выборка $\{y_p\}$ на основе функции z_2 $F_2(y)$ (кривая *2*), причем при достаточно малом шаге $_p$ $[0, 1]$, p 1, 2... изменения значения закона распределения $F_2(y)$ $[0; 1]$ можно получить выборку достаточно большого объема $\{y_p\}$;

• *на пятом шаге* формируется выборка $\{x_p\}$ посредством обращения функции y (x) d (кривая *4*);

• *на шестом шаге* осуществляется построение искомой функции $z_1 = F_1(x)$ (кривая *5*) по выборке $\{x_p\}$.

Относительно параметра сдвига d, используемого на втором и третьем шагах метода, следует отметить, что при построении функции $z_2 = F_2(y)$ (кривая *1*) может образоваться "хвост" интегральной функции распределения [27], которым для упрощения вычислений целесообразно пренебречь, т.е. преобразовать исходные данные $\{y_i\}$ на величину сдвига d $M[y]$ 3,5 $[y]$. При этом значения y случайной величины Y предлагается рассматривать внутри интервала величиной 3,5 , поскольку вероятность нахождения их вне этого интервала весьма мала.

Здесь мы рассмотрели частный случай задачи нахождения функции распределения $F_1(x)$ с учетом сдвига. Однако описанный подход к формированию выборок $\{y_p\}$, $\{x_p\}$ применим и для общего случая, когда условие $M[y]/[y]$ 3,5 не выполняется и сдвиг функции $z_2 = F_2(y)$ (кривая *1*) нецелесообразен.

Перспективные направления исследований жизнеспособности инновационных программных проектов в условиях неопределенности

Современная программная инженерия приблизилась к границе, за которой, по выражению Негойцэ (C.V. Negoita), "существенную роль начинают играть способы учета неопределенностей", т.е. так называемых НЕ-факторов внешней и внутренней среды ПП, отражающих *не*полноту знаний, их *не*достоверность, а также *не*четкость и *не*точность, относящихся к их содержанию. Очевидно, что для преодоления данной неопределенности необходим неформальный акт, связанный с привлечением тех или иных гипотез поведения факторов, порождающих неопределенность.

В настоящее время существуют различные конкурирующие гипотезы, опираясь на которые удается "устранить" неопределенность и придать проблеме управления ИПП количественную определенность. Среди различных способов формализации неопределенности наибольшее распространение получил стохастический (вероятностно-статистический) подход и, в частности, описанный выше подход, основанный на методе "Обращение–Сдвиг", позволяющий в терминах случайности моделировать неполноту знаний об объеме работ и производительности исполнителей проекта. Данный подход является весьма эффективным при анализе жизнеспособности, например, программного обеспечения для информационной поддержки контроля и управления состоянием территориальных систем [27]. Дело в том, что в силу многомерности и многосвязности данных систем, а также недостаточной изученности протекающих в них процессов, при их изучении особое место занимают именно статистические модели, ориентированные на обработку малых по объему и низких по точности исходных данных.

Однако на пути широкого распространения стохастических моделей неопределенности часто возникают затруднения, вызванные некорректным либо неправомерным использованием методологии теории вероятностей и математической статистики [28]. Как утверждает Касти (J. Casti): "Нет априорных математических оснований полагать, что механизм, порождающий неопределенность, по своей природе непременно стохастичен". В литературе принято считать неопределенными величины, для которых не обнаруживается статистическая устойчивость. Это обстоятельство явилось основанием для высказываний Калмана (R.E. Kalman): "Мы должны отрицать, что классические вероятностные структуры классической теории вероятностей, на самом деле, имеют научное отношение к описанию неопределенности", а также Н.Н. Моисеева: "Стохастические задачи, т.е. задачи, содержащие случайные величины или функции, мы не относим к числу задач, содержащих неопределенные факторы". В связи с этим в последние годы все большее распространение находят различные интервальные модели [29], а также нестохастические модели неопределенности типа [30–31]: субъективная вероятность Севеджа (L.J. Savage), верхняя и нижняя вероятности Демпстера (A.P. Dempster), правдоподобие и доверие Шеффера (G. Shafer), емкость Шоке (G. Choquet), возможности Заде (L.A. Zadeh) и Шейкла (G.L.S. Shackle), безразличную неопределенность В.И. Иваненко и В.А. Лабковского, возможность и правдоподобие Ю.П. Пытьева и др.

Весь арсенал используемых статистических методов можно распределить по трем потокам [32]: высокие статистические технологии, классические статистические технологии и низкие статистические технологии.

Несмотря на большое разнообразие подходов к оценке жизнеспособности ИПП с использованием указанных моделей неопределенности, наиболее перспективными являются комплексные подходы, получившее название высоких статистических технологий [32]. Под "высокими статистическими технологиями" понимаются пять "точек роста": непараметрическая статистика, робастность, бутстреп, статистика интервальных и нечисловых данных. Для задач оценки жизнеспособности ИПП наиболее перспективным является статистика интервальных и нечисловых данных.

СПИСОК ЛИТЕРАТУРЫ

1. **Макконенелл С.** Сколько стоит программный проект. М.: Русская редакция. СПб.: Питер, 2007.

2. **Архипенков С.Я.** Лекции по управлению программными проектами. М., 2009. URL: http://www.arkhipenkov.ru/resources/sw_project_management.pdf.

3. **Колоденкова А.Е.** Задачи программного инжиниринга сложных систем на основе критерия жизнеспособности проекта // Проблемы управления и моделирования в сложных системах: Труды XII Междунар. конф. Самара: Самарский НЦ РАН, 2010. С. 593–598.

4. **Мандрикова Л.В., Манжос Ю.С., Лучшев П.А.** Методы оценки стоимости и затрат на создание программного продукта, основанные на нечеткой логике // Радиоэлектронные и компьютерные системы. 2008. № 2. С. 115–118.

5. **Мандрикова Л.В., Манжос Ю.С., Хоменко В.В.** Метод идентификации рисков программного проекта на основе вероятностного подхода // Радиоэлектронные и компьютерные системы. 2009. № 7. С. 207–211.

6. **Бендиков М.А.** Оценка реализуемости инновационного проекта // Менеджмент в России и за рубежом. 2001. № 2. С. 27–43.

7. **Boehm B.** Cocomo II Model Definition Manual // Computer Science Department, University of Southern California, 1997.

8. **Липаев В.В.** Программная инженерия. Методологические основы: Учебник. М.: ГУ-ВШЭ, ТЕИС, 2006. 608 с.

9. **Карпенко С.Н.** Введение в программную инженерию: Учебно-методические материалы по программе повышения квалификации "Информационные технологии и компьютерное моделирование в прикладной математике". Нижний Новгород: Нижегородский ГУ им. Н.И. Лобачевского, 2007. URL: http://www.unn.ru/e-library/aids.html?pscience=6&posdate==2007

10. **Standish Group Report:** CHAOS Summary 2009. (http://www.standishgroup.com/newsroom/chaos_2009.php).

11. **Гордова И.Б.** Управление инновационными процессами. Кемерово: КемТИПП, 2005.

12. **ДеМарко Т., Листер Т.** Вальсируя с Медведями: Управление рисками в проектах по разработке программного обеспечения. М.: Компания p.m.Office, 2005.

13. **Бурков В.Н., Квон О.Ф., Цитович Л.А.** Модели и методы мультипроектного управления. М.: ИПУ РАН, 1997.

14. **Бурков В.Н., Джавахадзе Г.С.** Экономико-математические модели управления развитием отраслевого производства. М.: ИПУ РАН, 1997.

15. **Баркалов С.А., Портных В.А., Семенов П.И.** Минимизация упущенной выгоды в случае n независимых операций // Теория активных систем: Труды Междунар. науч.-практич. конф. Москва, 2001. Т. 1. С. 21–22.

16. **Романов В.С.** Задача управления стоимостью компании – дискретный случай // Управление большими системами. М.: ИПУ РАН, 2006. С. 142–152.

17. **Уандыков Б.К.** Оптимизация программ по стоимости // Теория активных систем: Труды Междунар. науч.-практич. конф. Москва, 2003.

18. **Маркаров Д.А.** Частная задача оптимизации сроков при управлении проектами // Управление большими системами. М.: ИПУ РАН, 2005. С.53–59.

19. **Галинская Е.А., Половинкина А.И., Рентеева Е.Л.** Модель минимизации затрат при формировании команды информационного проекта // Управление большими системами. Вып. 14. Воронеж: ВГАСУ, 2006. С. 29–33.

20. **Беляев Ю.А., Врублевская С.С., Половинкина А.И., Рентеева Е.Л.** Модель минимизации затрат при формировании команды проекта // Сборник научных трудов СевКавГТУ, 2007. № 3.

21. **Бекмурзаев В.А., Шеховцов А.А.** Использование метода динамического программирования и его оптимизация при решении задач управления проектами // Журнал научных публикаций аспирантов и докторантов. 2008. (http://www.jurnal.org/articles/2008/inf18.html).

22. **Черникова А.А.** Формирование инвестиционных пакетов, удовлетворяющих целям регионального развития // Инновационный Вестник Регион. 2007. № 2. С. 38–41.

23. **Беркун С.** Искусство управления IT-проектами. СПб.: Питер, 2007.

24. **Колоденкова А.Е.** Статистический подход к оценке реалистичности программных проектов для автоматизированных информационно-управляющих систем // Мехатроника, автоматизация, управление. 2010. № 4. С. 52–60.

25. **Гвоздев В.Е., Колоденкова А.Е.** Непараметрическое оценивание функциональных зависимостей по эмпирическим данным // Мехатроника, автоматизация, управление. 2005. № 8. С. 12–18.

26. **Пугачев В.С.** Теория вероятностей и математическая статистика. М.: ФИЗМАТЛИТ, 2002.

27. **Гузаиров М.Б., Гвоздев В.Е., Ильясов Б.Г., Колоденкова А.Е.** Статистическое исследование территориальных систем: Монография. М.: Машиностроение, 2008.

28. **Филимонов Н.Б.** Смена парадигм неопределенности в задачах управления // Интеллектуальные системы: Труды Шестого междунар. симп. М.: РУСАКИ, 2004. С. 173–178.

29. **Яценко Б.Н.** Оценка эффективности инвестиционных проектов и принятие инвестиционных решений в условиях большой неопределенности интервального типа // Аудит и финансовый анализ. 2006. № 1. С. 167–180.

30. **Клементьева С.В.** Применение теории нечетких множеств для измерения и оценки эффективности реализации наукоемкой продуктовой инновации // Заводская лаборатория. Диагностика материалов. 2006. № 11. С. 65–69.

31. **Пытьев Ю.П.** Возможность. Элементы теории и применения. М.: УРСС, 2000.

32. **Орлов А.И.** Высокие статистические технологии // Заводская лаборатория. 2003. Т. 69. № 11. С.55–60.

УДК 004.4

К.А. Костюхин, канд. физ.-мат. наук, стар. науч. сотр.,
Научно-исследовательский институт системных исследований РАН, г. Москва

Модель разработки программного обеспечения с открытым исходным кодом

Представлена модель разработки программного обеспечения с открытым исходным кодом, построенная с учетом положений отечественных и зарубежных стандартов в области программной инженерии.

Ключевые слова: *открытый исходный код, лицензирование, стандарты*

Программное обеспечение автоматизированных информационных систем

Вначале рассмотрим ряд базовых понятий, связанных с автоматизированными системами (далее АС) и, в частности, понятие программного обеспечения (далее ПО). Область, связанная с процессами разработки и внедрения АС, регламентируется отечественными государственными стандартами (далее ГОСТ) 34-й серии. Основные термины и определения представлены в ГОСТ 34.003–90 [7]. Согласно ГОСТ АС представляет собой *систему, состоящую из персонала и комплекса средств автоматизации его деятельности, реализующую информационную технологию выполнения установленных функций.*

В ГОСТ [7] выделяется ряд классов АС, включая:

- автоматизированные системы управления (АСУ);
- системы автоматизированного проектирования;
- автоматизированные системы научных исследований.

Последняя редакция ГОСТ 34-й серии относится к 1990-м годам. Рассмотрим, как понятие АС определяется в современной научно-технической литературе. Воспользуемся словарем-справочником по информационным технологиям [4], согласно которому под АС понимается:

- *совокупность управляемого объекта и автоматических управляющих устройств, в которых часть функций управления выполняет человек-оператор;*
- *комплекс технических, программных, других средств и персонала, предназначенных для автоматизации различных процессов.*

В данном словаре-справочнике дается также следующее определение автоматизированной информационной системы (АИС):

- *в прямом (узком) значении: комплекс программных, технических, информационных, лингвистических, организационно-технологических средств и персонала, предназначенный для решения задач справочно-информационного обслуживания и/или информационного обеспечения пользователей информации;*
- *в расширенном значении: комплекс программных, технических, информационных, лингвистических, организационно-технологических средств и персонала, предназначенный для сбора, обработки (первичной), хранения, поиска, обработки (вторичной) и выдачи данных в заданной форме (виде) для решения разнородных профессиональных задач пользователей системы.*

При этом в работе [4] отмечается, что в различных практических применениях термины

АИС и АС часто используются в качестве эквивалентов.

Важными характеристиками АИС, которые отличают их от информационно-поисковых систем, обеспечивающих только одну функцию – поиск информации, согласно [4] являются:

- многофункциональность (способность решать разнообразные задачи);
- независимость процессов сбора, обработки (первичной), ввода данных и их обновления (актуализации) от процессов их использования прикладными программами;
- независимость прикладных программ от физической организации баз данных.

Определения, представленные в ГОСТ [7] и словаре-справочнике [4], не противоречат друг другу. При этом определения из словаря-справочника [4] можно рассматривать как расширяющие и детализирующие определения, данные в стандарте.

С понятием АС тесно связаны понятия различных типов их обеспечения. Выделяются следующие типы обеспечения: организационное; методическое; техническое; математическое; программное; информационное; лингвистическое; правовое; эргономическое. Программное обеспечение при этом определяется следующим образом: *совокупность программ на носителях данных и программных документов, предназначенная для отладки, функционирования и проверки работоспособности АС*.

Приведем определение "программы" согласно Единой системе программной документации (далее – ЕСПД) [5]: программа – *данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма*.

В стандарте [7] при определении понятия "программное изделие в автоматизированной системе" используется термин "программное средство" (далее – ПС). Понятие ПС очень активно используется в современных отечественных стандартах программной инженерии, например [6, 8]. Согласно стандарту [8]: программное средство (далее – ПС, англ. *software*) – *объект, состоящий из программ, процедур, правил, а также, если предусмотрено, сопутствующих им документации и данных, относящихся к функционированию системы обработки информации*.

Примечание. Программное средство представляет собой конкретную информацию, объективно существующую как совокупность всех значимых с точки зрения ее представления свойств каждого из материальных объектов, содержащих в фиксированном виде эту информацию.

Приведем также и определение программного продукта согласно того же стандарта [8]: программный продукт (англ. *software product*) – *ПС, предназначенное для поставки, передачи, продажи пользователю*.

В дальнейшем будем использовать понятие ПС и, если это не приводит к противоречиям, будем полагать, что ПО АС состоит из набора ПС.

Программное обеспечение с открытым исходным кодом

Термин "программное обеспечение с открытым исходным кодом" (ПО с ОИК) является переводом англоязычного термина *"open source software"*, за которым скрывается модель разработки, внедрения и использования ПС, исторически сложившаяся вокруг так называемых "открытых лицензий". Подобную модель, для краткости, будем именовать моделью *open source*. В дальнейшем под ПО с ОИК будем понимать ПС, удовлетворяющие модели open source и используемые либо в составе ПО АИС, либо самостоятельно.

Характерными чертами модели *open source* являются:

- возможность свободно (не значит "бесплатно" для его пользователя) распространять ПС;
- доступность исходного кода программ, входящих в состав ПС;
- право создавать на основе ПС свои собственные разработки.

Прежде чем непосредственно переходить к детальному исследованию модели *open source*, рассмотрим ряд родственных понятий, включая свободное ПО и открытые стандарты.

Свободное программное обеспечение

В середине 1980-х годов Ричард Столлмэн основал фонд свободного программного обеспечения *Free Software Foundation* (далее – FSF) [11]. Основной целью фонда является создание законной альтернативы проприетарному (закрытому, частному) ПО, т.е. юридическая и правовая поддержка разработчиков свободных программных продуктов. Слово "свободное" в данном случае интерпретируется следующим образом: во-первых, как свобода копирования программного

продукта и передачи его другим пользователям, во-вторых, как свобода изменения программы при полном доступе к ее исходному тексту. При этом проприетарным считается всякий программный продукт, защищенный авторским правом (*copyright*).

Основным проектом фонда FSF стал проект GNU (*Gnu's Not Unix*, GNU – это не *Unix*). В рамках этого проекта была разработана специальная лицензия GPL (подробнее см. в разделе об открытых лицензиях), защищающая открытость распространяемых под этой лицензией программ. В настоящее время в проекте GNU задействовано большое число разработчиков со всего мира, а некоторые программные продукты GNU "дефакто" являются стандартом для своих проприетарных аналогов. Например, набор компиляторов *gcc* (GNU *Compiler Collection*) и отладчик *gdb* (GNU *Debugger*).

После того как в рамках проекта GNU стало появляться все больше и больше программных продуктов, отдельные части которых можно было использовать повторно в собственных разработках сторонних компаний, проявилась абсолютно неприемлемая для большинства игроков рынка ПО ситуация с лицензией GPL. Необходимо было смягчить эту лицензию, чтобы дать возможность разработчикам ПО, с одной стороны, использовать существующий свободный программный код, а с другой – не принуждать их раскрывать собственный код всех остальных компонентов разрабатываемого программного продукта.

Таким образом, в 1997 г. возникла новая организация *Open Software Initiative* (далее – OSI) [12]. Именно усилиями этой организации окончательно оформилась модель *open source*. В отличие от FSF модель OSI позволяет разработчикам более гибко использовать ПС, разработанные под открытой лицензией, в своих собственных проектах.

Открытые стандарты

Понятие открытости ПО можно трактовать различными способами. Первый способ связан с реализацией возможности изучения и модификации исходных текстов программ, входящих в состав ПО. Второй способ подразумевает использование при проектировании и реализации ПС некоторых стандартизованных решений, в частности, использование основанных на доступных (открытых) стандартах механизмов взаимодействия с ПС.

Со вторым способом связано понятие "открытой системы", при построении которых обычно выделяются следующие принципы [4]:

- переносимость (*portability*), позволяющая легко переносить данные и программное обеспечение между различными платформами;
- взаимодействие (*interoperability*), обеспечивающее совместную работу устройств разных производителей;
- масштабируемость (*scalability*), гарантирующая сохранение инвестиций в информацию и программное обеспечение при переходе на более мощную аппаратную платформу.

Для того чтобы обеспечить соблюдение этих принципов, ведется активная работа по стандартизации интерфейсов взаимодействия программ на всех уровнях эталонной модели *Open System Interconnection* (далее – OSI). В настоящее время существует несколько организаций, занимающихся разработкой и внедрением стандартов открытых систем.

Одним из главных разработчиков стандартов можно считать рабочие группы *Portable Operating System Interface* (далее – POSIX), которые занимаются стандартизацией интерфейсов операционных систем. В [9] приведен перечень рабочих групп POSIX, регулярно выпускающих документы по стандартизации.

Открытые лицензии

Все проекты ПО с ОИК базируются на понятиях "авторского права", авторы ПО с ОИК владеют своими разработками, о чем свидетельствует наличие открытых лицензий. Целью лицензирования в модели *open source* не является защита от нарушения или кражи интеллектуальной собственности. Напротив, открытые лицензии гарантируют автору, что его программный продукт будет распространяться именно в таком виде, в котором автор того пожелал. Для этой цели существует достаточно большое число типов открытых лицензий, которые будут рассмотрены далее.

При отсутствии какой бы то ни было лицензии, правообладателем программного продукта может стать любой его пользователь. Кроме того, этот пользователь волен зарегистрировать свое авторское право на данный продукт и распространять его уже под любой, в частности проприетарной, собственной лицензией. Таким об-

разом, приходим к выводу, что наличие лицензии, в первую очередь, необходимо для того, чтобы ПО с ОИК оставалось открытым.

Определение открытой лицензии

Организацией OSI было предложено официальное определение открытой лицензии *Open Source Definition* (далее – OSD) [10]. Исходная версия этого определения была сформулирована Брюсом Пересом в 1997 г.

В OSD есть несколько неоднозначно трактуемых положений, которые требуют дополнительного разъяснения. Далее под лицензиатом будет пониматься пользователь, принимающий лицензию, под которой распространяется программный продукт, а под лицензиаром – владелец лицензии на конкретный продукт или изделие.

Первое из таких положений гласит: *лицензиат вправе использовать программный продукт для любых целей*.

Открытая лицензия не должна накладывать ограничений на использование программного продукта, в том числе "только для некоммерческого использования".

Второе положение: *лицензиат может делать копии программного продукта и распространять их безо всяких отчислений лицензиару*.

Это положение вовсе не означает, что лицензиар не может продавать свой продукт. Просто лицензиат, единожды купив этот продукт, может сам делать его копии и распространять (в том числе и за плату) под установленной лицензией.

Третье из рассматриваемых положений: *лицензиат может создавать собственные программные продукты на основе данного безо всяких отчислений лицензиару*.

Его суть в том, что качественное ПС создается на основе уже существующих продуктов. Это положение важно, поскольку у разработчиков должна быть возможность использования имеющегося качественного кода в своих целях.

Четвертое положение: *лицензиат может получить и использовать исходный код программного продукта*.

Это положение означает, что лицензиар должен предоставить исходный код программного продукта лицензиату, не взимая за это дополнительную плату. Ошибочной трактовкой данного пункта является предположение, что лицензиар обязан раскрыть и предоставить в широкий доступ исходный код своего программного продукта для всех желающих. Это не совсем так. Исходный код может (и должен) распространяться вместе с самим продуктом и только среди пользователей этого программного продукта. Следует отметить, что существует ряд открытых лицензий, не требующих обязательного предоставления исходного кода.

Пятое из рассматриваемых положений: *лицензиат может использовать ПО с ОИК вместе с проприетарными программными продуктами*.

Открытые лицензии не могут налагать ограничения на совместное использование ПО с ОИК и проприетарного ПО. Это положение, однако, не означает, что лицензиар не может налагать ограничения для тех лицензиатов, кто создает и распространяет собственные ПС на основе данного программного продукта.

Интеллектуальная собственность

Законы об интеллектуальной собственности различают произведения искусства (живописи, музыки, литературы) и результаты научной деятельности (научно-технические инновации). Произведения искусства находятся под защитой законов об авторском праве (*copyright*), в то время как результаты научной деятельности призваны защищать законы о патентовании (*patent*). На настоящее время программа может трактоваться и как произведение искусства, и как результат научной деятельности.

Следуя представленному ранее (согласно ЕСПД [5]) определению "программы", необходимо отметить, что в Российской Федерации данное понятие должно трактоваться исключительно как результат научной деятельности, а значит подпадать под действие законов о патентовании. На практике, однако, ПС также защищается и законами об авторском праве.

Нужно отметить еще один аспект открытых лицензий. Свобода интеллектуальной собственности разработчика ПС для пользователей в рамках лицензии совсем не означает той свободы интеллектуальной собственности, каковой является, например, свобода произведений Шекспира и Толстого. Свобода интеллектуальной собственности разработчика определена в рамках формулировок и ограничений конкретной, выбранной разработчиком лицензии. В этой связи еще раз отметим: *открытые лицензии не нарушают прав интеллектуальной собственности разработчика!* Наоборот, у ПС, которые распространяются по

открытым лицензиям, есть владельцы в лице компаний или индивидуальных разработчиков, и такие программы также защищены соответствующими законами об авторском праве и патентовании.

Отметим, что кроме авторского права и патентов есть еще и понятие торговой марки (например, операционная система (далее – ОС) *Linux* является торговой маркой, зарегистрированной на Линуса Торвальдса, автора ОС *Linux*). Такие открытые лицензии, как GPL, не рассматривают вопрос о защите торговой марки. В том числе и по этой причине на настоящее время существует большое число клонов ОС *Linux*, распространяемых под собственным названием. Вместе с тем многие компании (например, *Mozilla*, *Apache*) разрабатывают собственные лицензии, включающие защиту своих торговых марок.

Таксономия лицензий

Можно выделить следующие типы открытых лицензий [2]:

- академические;
- взаимно обязывающие;
- промышленные;
- стандартные;
- контентозависимые.

Рассмотрим каждый из выделенных типов подробнее.

Академические лицензии

Лицензии этого типа создаются академическими институтами для распространения собственных разработок. Характерной особенностью подобных лицензий является отсутствие требований к лицензиату по распространению исходного кода. Такая особенность является привлекательной для тех разработчиков, которые хотят использовать некоторые фрагменты ПО с ОИК, однако не хотят при этом раскрывать свой исходный код. Наиболее известной лицензией этого типа является лицензия *Berkeley Software Distribution* (далее – BSD) Калифорнийского университета. Следует отметить, что лицензию BSD часто используют в качестве шаблона для создания собственных лицензий. Примером может служить лицензия *Massachusetts Institute of Technology* (далее – MIT) Массачусетского Технологического Института. Основной особенностью лицензии MIT является возможность изменять политику лицензирования компонентов, разработанных на основе исходного продукта. В частности, эти компоненты могут иметь другие ограничения по распространению и использованию или модификации.

Еще одна лицензия, о которой необходимо упомянуть в настоящем разделе, это лицензия *Artistic* (художественная). Такое название обусловлено тем обстоятельством, что это единственная лицензия, гарантирующая неприкосновенность исходного программного продукта, а такая ситуация как раз и наблюдается в среде лицензирования произведений искусства. Смысл лицензии *Artistic* заключается в том, что только правообладатель программного продукта имеет право модифицировать (или разрешать другим модифицировать) исходный программный пакет (как исходный код, так и бинарный).

Взаимно обязывающие лицензии

Взаимно обязывающие лицензии в историческом плане – наиболее "древние" из открытых лицензий. Они требуют от лицензиата обязательного предоставления исходного кода продукта любому пользователю по его запросу. Типичным представителем такого сорта лицензий является лицензия *General Public License* (далее – GPL) Ричарда Столлмэна. Основной особенностью лицензии GPL является так называемый "вирус *copyleft*", "заражающий" остальные компоненты программного пакета с лицензией GPL. Например, если разработчик использует в своем ПС библиотеку, распространяемую под лицензией GPL, то итоговый программный продукт автоматически приобретает лицензию GPL и статус ПО с ОИК со всеми вытекающими отсюда последствиями (например, необходимость предоставлять по запросу исходный код всего продукта). Многие разработчики неправильно трактуют этот момент, заранее отказываясь от GPL. На самом деле ПС "заражается" GPL только в том случае, если и само ПС, и компонент с лицензией GPL распространяются как части единого программного продукта в одном дистрибутиве. Если же программный продукт просто требует для своей работы наличия в системе GPL-библиотеки, то он сам вполне может иметь собственную лицензию. Для устранения описанного выше недостатка была разработана более мягкая лицензия *Library* GPL (далее – LGPL). Впоследствии сокращение LGPL стало расшифровываться как *Lesser* GPL. По этой лицензии можно использовать, напри-

мер, LGPL-библиотеки без раскрытия собственного кода.

Промышленные лицензии

К лицензиям этого типа относятся лицензии конкретных компаний, работающих на рынке ПО с ОИК. Наиболее грамотно составленная лицензия принадлежит компании *Netscape Corporation* – разработчику web-браузера *Mozilla*. Лицензия получила название *Mozilla Public License* (далее – MPL).

Лицензия MPL разбивает программный продукт на две части:

- часть, соответствующую ПО с ОИК, называемую "защищенной" ("*covered*");
- оставшуюся часть, которую добавляют сторонние разработчики.

Такая классификация позволяет разработчикам добавлять свои собственные программы (файлы) и распространять их вместе с немодифицируемым ядром. Если разработчик изменил защищенную часть, то он должен предоставить измененные файлы в соответствии со специальными OSD-правилами. Лицензия довольно просто и ясно составлена, что облегчает совместную разработку проприетарного ПО и ПО с ОИК. Лицензия MPL по некоторым своим положениям близка к коммерческим лицензиям и включает такие стандартные положения коммерческих лицензий, как ответственность разработчика и пользователя.

Юристы компании IBM разработали собственную открытую лицензию *Common Public License* (далее – CPL). Главной ее особенностью является наличие утверждения о том, что лицензия "сопровождает" программу. Оно означает, что прежде чем начать загрузку программы с сайта разработчика, установку программы из дистрибутива или перед первым ее использованием, необходимо принять условия лицензии посредством нажатия некоторой программной кнопки. Кроме того, она представляет собой доработанный вариант лицензий GPL и LGPL, в котором четко описаны возможности по модификации кода и по лицензированию программных продуктов, использующих CPL-код. В таком случае изменения возможны только в виде отдельных модулей, не использующих код продуктов с лицензией CPL.

Следует отметить, что большинство промышленных лицензий вышло из академических. Таковой, например, является лицензия *Apache*. Причина широкого использования подобного способа лицензирования компаниями-разработчиками обусловлена возможностью распространять программный продукт, не открывая исходный код. Важное отличие лицензии *Apache* от лицензии BSD заключается в том, что последняя требует обязательного упоминания во всех рекламных материалах, описывающих программный продукт, того факта, что этот продукт содержит компоненты, разработанные в Калифорнийском университете. Лицензия же *Apache* требует простого упоминания об используемых компонентах *Apache* в пользовательской документации или в самом исходном коде. Такой подход гораздо больше подходит компаниям, намеревающимся создавать собственные проприетарные решения на основе ПО с ОИК.

Стандартные лицензии

Эти лицензии разрабатывались с целью обеспечить гарантии того, чтобы существующие реализации стандартов в виде ПО и документации были свободно доступными для всех разработчиков. В качестве примера можно привести лицензию *Sun Industry Standards Source License* (далее – SISSL).

Контентозависимые лицензии

В настоящее время идеи открытых лицензий и модели *open source* проникают в другие области науки и искусства. Уже имеются книги, публикуемые в Интернете под лицензиями, гарантирующими их свободное распространение. Наибольшей популярностью пользуется лицензия *Academic Free License* (далее – AFL).

Другие лицензии

Все открытые лицензии обязательно должны пройти проверку в организации OSI и получить статус *"OSI sertified"*.

Кроме отмеченных выше, следует упомянуть еще несколько лицензий, не совсем вписывающихся в идею модели *open source*, однако претендующих на некоторую открытость. Это так называемые разделяемые лицензии. Такой является, например, лицензия библиотеки *Qt*. Пользователь вправе использовать и распространять библиотеку и все созданные на ее основе программы, если не подразумевается коммерческое использование этих программ. Таким образом у лицензии *Qt Free Edition License* имеется два основных отличия от GPL:

- не допускается внесение изменений в библиотеку *Qt*;

• запрещается коммерческое распространение полученного программного продукта.

Разработчики проприетарного ПО должны купить *Qt Professional Edition License*, оплачивая, тем самым, возможность коммерческих разработок. Следует отметить, что от *Qt* зависит графическая пользовательская среда KDE, которой пользуется большое число пользователей ОС *Linux*. До тех пор, пока ОС *Linux* не получила широкого распространения в коммерческих и государственных учреждениях, оставаясь ОС любителей и энтузиастов, можно было распространять KDE под лицензией *Qt Free Edition License*. Теперь, когда ОС *Linux* завоевала нишу среди популярных ОС, разработчики программ для KDE встали перед необходимостью оплачивать профессиональную лицензию в случае коммерческого распространения своих программных продуктов.

Другим показательным примером является *Microsoft Shared Source License* – лицензия компании *Microsoft*, по которой предоставляется часть исходного кода продуктов этой компании. В этой лицензии разрешается некоммерческое использование полученного исходного кода. В формулировке данной лицензии речь идет об использовании кода, который лицензиат смог запомнить после получения доступа к исходному коду. К исходным предпосылкам, которые стимулировали компанию *Microsoft* к открытию части исходного кода, можно отнести следующие. Во-первых, для сторонних разработчиков появилась возможность получить доступ к различным форматам данных и интерфейсам объектов, что помогает обеспечить совместимость различных форматов и улучшить взаимодействие между сторонними программами и компонентами, разработанными с компанией *Microsoft*. Во-вторых, такой шаг может улучшить показатели информационной безопасности и надежности продукта за счет активного привлечения к его созданию и развитию большого числа независимых разработчиков. Подобный шаг одной из ведущих IT-компаний мира только подтверждает позиции сторонников модели *open source*, утверждающих, что доступность исходного кода является одним из факторов, повышающих надежность и безопасность ПО.

В середине октября 2007 г. *Microsoft* предприняла беспрецедентный шаг. Две лицензии этой компании: *Microsoft Public License* (далее – Ms-PL), бывшая *Microsoft Permissive License*, и *Microsoft Reciprocal License* (далее – Ms-RL) получили официальный статус открытых лицензий, так называемый знак *"OSI Certified"*. Лицензия Ms-PL представляет собой фактически укороченную GPL. Разница между Ms-PL и Ms-RL состоит в том, что в Ms-RL особым образом оговаривается статус отдельных файлов исходного кода проекта. С учетом отмеченных обстоятельств, Ms-RL больше напоминает лицензию BSD.

Следует отметить и лицензию компании *Artifex Software*, занимающейся распространением пакета *Ghostscript* под лицензией *Aladdin Free Public License*. Несмотря на название, эта лицензия не является свободной и не сертифицирована организацией OSI. Данной лицензией запрещается коммерческое распространение *Ghostscript* или программных продуктов, содержащих этот пакет. Для коммерческого распространения необходимо приобрести другую лицензию *Artifex Commercial License*.

Предмет лицензирования

В модели *open source* обычно выделено несколько архитектурных уровней, на которых располагается ПО под различными лицензиями.

Уровень первый. Операционные системы. На уровне операционной системы (ОС) в основном используется лицензия GPL. Причина в том, что ОС – наиболее широко распространяемое ПО. Число пользователей конкретной ОС всегда заведомо выше числа пользователей какого-то конкретного приложения, которое работает под управлением этой ОС. Здесь GPL играет стандартизирующую роль, покрывая все лицензии, которые могли бы появиться на этом уровне, и оставляя при этом код свободным и открытым. На этом базовом уровне модели *open source* коммерческое использование заключается в изготовлении собственных, конкурентоспособных в данной предметной области дистрибутивов. Самым распространенным примером ПО этого уровня является ОС *Linux*.

Уровень второй. Пакеты разработчика. На этом уровне ПО может быть как коммерческим, так и свободно распространяемым. Существенным является тот факт, что один и тот же пакет не может изменять своего состояния (быть для одних коммерческим, для других свободным), т.е. нарушать требование OSD, которое запрещает дискриминацию определенных групп пользователей, например, студентов или разработчиков коммерческого ПО.

Уровень второй (продолжение). Расширения пакетов разработчика и дополнительные библиотеки. Как было показано в описании лицензии MPL, расширения базовых пакетов могут распространяться уже на другой (например, коммерческой) основе. В этом случае пользователь должен внимательно ознакомиться с предполагаемой лицензией.

Уровень третий. Приложения. На этом уровне многие разработчики стремятся защитить свою интеллектуальную собственность посредством закрытия некоторых фрагментов исходного кода. Зачастую, даже лицензии LGPL для этого оказывается недостаточно. По этой причине наиболее оправданным в этом случае является использование лицензий типа BSD.

Сообщества разработчиков

Одним из важных отличий традиционных разработок ПО с ОИК является отсутствие вертикали "разработчик–пользователь". Каждый разработчик является пользователем, а каждый пользователь при желании может быть разработчиком.

Существует следующая диаграмма (см. рисунок) распределения ролей исполнителей проекта по созданию ПО с ОИК [1].

Нередко разработчиком в проекте по созданию ПО с ОИК является всего один человек. Однако, для выполнения большинства проектов, как правило, формируется определенная группа исполнителей, обладающая характеристиками, которые подробно представлены в работе [3].

Централизация управления проектом. В некоторых проектных командах прослеживается четкая иерархия:

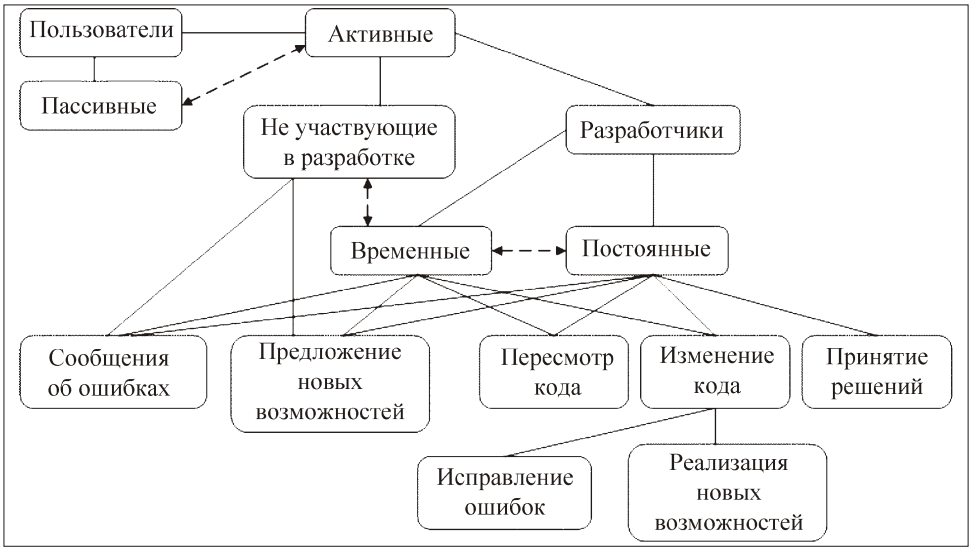
- обычные разработчики, предлагающие изменения в коде;
- рецензенты, проверяющие и комментирующие предложенные изменения;
- привилегированные разработчики, имеющие право вносить изменения в код на основе собственного мнения и комментариев рецензентов.

В других группах все разработчики равноправны и имеют возможность вносить изменения на основе достигнутого между собой консенсуса. Как правило, это касается небольших проектов, требующих незначительного числа разработчиков.

Меритократия (от англ. *merit* – достоинство, заслуга). В процессе своего вовлечения в проект обычный пассивный пользователь может перейти в группу активных, а затем присоединиться к разработчикам. Такое продвижение пользователя является признанием его заслуг перед группой разработчиков проекта по созданию ПО с ОИК.

Заключение

При грамотном использовании программное обеспечение с открытым исходным кодом может существенно уменьшить сроки и трудозатраты на разработку собственного программного продукта, одновременно обеспечивая его соответствие при-



Структура выполнения традиционного *open source* проекта

нятым стандартам и требованиям по обеспечению надлежащего уровня качества и надежности.

Надеемся, что рассмотренная в настоящей статье модель разработки *open source* окажется полезной как разработчикам программного обеспечения, так и специалистам по его стандартизации.

СПИСОК ЛИТЕРАТУРЫ

1. **Gacek C., Arief B.** The many meanings of open source // IEEE Software. 2004. Т. 21. № 1. С. 34–40.
2. **Lawrence R.** Open Source Licensing: Software Freedom And Intellectual Property Law. Prentice Hall PTR, 2004. 432 p.
3. **Lussier S.** New tricks: How open source changed the way my team works // IEEE Software. 2004. С. 68–72.
4. **Воройский Ф.С.** Информатика. Новый систематизированный толковый словарь-справочник (Введение в современные информационные и телекоммуникационные технологии в терминах и фактах). 3-е, доп. изд. М.: Физматлит, 2003. 760 с.
5. **ГОСТ 19781–90.** Государственный стандарт Союза ССР. Единая система программной документации. Обеспечение систем обработки информации программное. Термины и определения. М.: Изд-во стандартов.
6. **ГОСТ 28806–90.** Государственный стандарт Российской Федерации. Информационная технология. Качество программных средств. Термины и определения (Software quality. Terms and defnitions). М.: Изд-во стандартов.
7. **ГОСТ 34.003–90.** Государственный стандарт союза ССР. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения.; Введ. 01.01.1992. М.: Изд-во стандартов, 1992.
8. **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.
9. **Кузнецов С.Д.** Открытые системы, процессы стандартизации и профили стандартов [Электронный ресурс]. Электрон. текст. дан. URL: http://www.citforum.ru/database/articles/art_19.shtml, свободный.
10. **Официальное** определение "открытой лицензии" (Open Source Definition) [Электронный ресурс]. URL: http://www.opensource.org/docs/osd/, свободный.
11. **Официальный** сайт организации "Free Software Foundation" [Электронный ресурс].URL: http://fsf.org, свободный.
12. **Официальный** сайт организации "Open Software Initiative" [Электронный ресурс]. URL: http://www.opensource.org/, свободный.

УДК 004.41 (681.3.06)

А.Н. Терехов, д-р физ.-мат. наук, проф. зав. каф. СПбГУ, ген. дир. ЗАО "Ланит-Терком"

E-mail: andrey.terekhov@lanit-tercom.com

Что такое программная инженерия

Обсуждаются исторические аспекты возникновения и развития программной инженерии, международные образовательные стандарты в данной области и вопросы дополнительного производственного обучения.

Ключевые слова: *программная инженерия, технологии программирования, образовательные стандарты, производственное обучение*

Чуть-чуть истории

Первая[1] ЭВМ ENIAC была создана под руководством Джона Моучли в 1946 г. в Пенсильванском университете (г. Филадельфия). В СССР первая ЭВМ МЭСМ была разработана под руководством С.А. Лебедева в Киеве в 1951 г. Однако в Киеве он проработал меньше двух лет и вернулся в Москву, где в 1953 г. в Институте точной механики и вычислительной техники АН СССР (ИТМ и ВТ) создал БЭСМ – самую быструю ЭВМ в Европе того времени.

В США в 1951 г. Д. Моучли запустил в серию ЭВМ UNIVAC (было выпущено 47 шт.) В СССР первой серийной ЭВМ была БЭСМ-2 (67 шт.), которая выпускалась с 1958 г. Практически сразу же за БЭСМ-2 была запущена в серию М-20; также серийно выпускались ЭВМ Урал-1 (183 шт. начиная с 1957 г.), с 1960 г. – Минск-1 (220 шт.).

В начале 50-х гг. XX века первые программисты работали один на один с ЭВМ, программируя непосредственно в ее кодах. В качестве устройства ввода использовались перфоленты и перфокарты.

Я начал программировать в 1963 г. на ЭВМ "Урал-1", где в качестве носителя устройства ввода использовалась кинолента. Мое программирование было не совсем добровольным – мой отец – полковник, инженер по электронике самолетов – уйдя в запас, был вынужден переучиваться с аналоговой техники на цифровую. Поскольку он был уже старым, как мне тогда казалось (тогда ему было 49 лет), он решил, что будет лучше, если сначала я прочитаю книгу Китова и Криницкого [1], а уж потом объясню ему.

В первые годы программирование занимало малый процент всей работы, нужно было масштабировать (арифметики с плавающей запятой на многих ЭВМ не было), для чего необходимо было крутить ручку арифмометра, долго набивать код программы на весьма неудобных устройствах, а вершиной отладочных средств была прокрутка – покомандный интерпретатор.

Еще раз повторяю, что первые программисты абсолютно все делали сами. Постепенно пришло понимание, что лучше накапливать библиотеки математических функций, отладочных средств, программ форматного ввода/вывода и т.д. В больших вычислительных центрах появились специальные люди – хранители и толкователи библиотек подпрограмм. В 1957 г. появились языки FORTRAN (FORmula TRANslator), созданный под руководством Джона Бэкуса в IBM. Те люди, которые собирали первые библиотеки полезных программ, придумали FORTRAN и реализовали

[1]Обычно в качестве первой вычислительной машины (если не считать механическую машину Чарльза Бэббиджа – 1837 г.) называют Mark I, созданную в 1943 г. в Гарвардском университете под руководством Говарда Эйкена (Howard H. Aiken), но это была релейная, т.е. электромеханическая машина, а первой действительно электронной (на триггерах) стала именно ENIAC.

первые трансляторы с него в коды ЭВМ, создали новую науку – системное программирование.

В СССР системное программирование начало развиваться очень рано. Профессор МГУ А.А. Ляпунов еще в 1953 г. начал работы по операторному методу в программировании, в 1953–1954 гг. он читал лекции по программированию будущему академику А.П. Ершову, среди его учеников были уже упомянутые А.И. Китов и Н.А. Криницкий, очень уважаемый мною И.В. Поттосин и многие другие известные программисты. В 1956 г. были опубликованы работы Ю.И. Янова по схемам программ, чуть позже работы С.С. Лаврова и А.П. Ершова по экономии памяти на основе раскраски графов. Работы Ю.И. Янова, С.С. Лаврова и А.П. Ершова во всем мире признаны классическими, заложившими основы теоретического программирования.

Системное программирование

Практически все первые теоретики программирования были и великими практиками, они создавали трансляторы, операционные системы, оптимизаторы, шахматные программы – чемпионы мира и т.д. Однако постепенно теория стала отдаляться от практики. Я навсегда запомнил и часто повторяю своим студентам гневную речь А.А. Ляпунова на панельной дискуссии Всесоюзной конференции по программированию в Академгородке: "Не понимаю я современных молодых ученых. В мои годы практика ставила задачи, а теоретики их решали. Часто теория шла впереди практики. А что мы слышали на этой конференции? Оптимизация памяти, но без массивов, корректность программ, но без процедур, goto и присваиваний. Кому это нужно?". Тем не менее, когорта первых системных программистов в СССР была настолько сильна, что практическое направление системного программирования не только не ослабело, но и бурно развивалось.

Таким образом, можно сказать, что первые системные программисты занимались разработкой различных инструментальных средств и новых методов, теорий и алгоритмов, необходимых в этой многотрудной деятельности.

Постепенно системные программисты как специалисты существенно более высокой квалификации, чем обычные прикладные программисты, стали привлекаться к самым трудным задачам – от расчета атомных бомб до управления предприятиями. Иногда это было вынужденной мерой, например, я помню, как трудно шло внедрение первого в СССР транслятора с языка Алгол 68 для ЕС ЭВМ и первых графических технологий. Сначала прикладники (особенно если это военные люди) "забивают микроскопом гвозди", а потом говорят, что технология плохая. Приходилось засучивать рукава и лезть в самые разные области.

Особого упоминания заслуживают встроенные системы реального времени. Множество параллельно протекающих процессов, особо высокие требования к времени отклика и надежности, непосредственная работа с оборудованием – эти и многие другие особенности систем реального времени создают им ореол самых трудных задач. Я обожаю слушать рассказы А.Н. Томилина о первой в мире системе противоракетной обороны "Система А" (1961 г.), в разработке программного обеспечения для которой он принимал участие.

Технология программирования

Термин "технология промышленного программирования" в наш коллектив привнес капитан 1-го ранга профессор В.П. Морозов в конце 1980 г. Поначалу мы были в ужасе, так далеко это было от наших университетских традиций. Я даже специально ездил в Академгородок к А.П. Ершову за советом. Он посмеялся над моими страхами, назвал несколько известных людей, про которых я и не знал, что они генералы или адмиралы. А.П. Ершов поддержал наши начинания, написал развернутый отзыв [2] и даже более двух лет работал у нас академиком-консультантом.

Поначалу такие вещи, как отчуждение программы от ее автора, контроль за ходом разработки, графические схемы ситуаций мы называли "шпионскими штучками", но постепенно пришло понимание, что работа коллективов из сотен программистов без таких средств невозможна. Поскольку в западных странах "площадь соприкосновения с пользователем" (выражение А.П. Ершова) у программирования была несравненно больше, там понимание особенностей промышленного программирования пришло несколько раньше. Впервые термин "инженерия программного обеспечения" прозвучал на одноименной конференции НАТО в 1968 г. Я хорошо помню довольно толстую книжку в твердой обложке черного цвета с русским переводом трудов

этой конференции [3], но тогда особых откликов она у меня не вызвала. Подумаешь, "управление проектами и коллективами" – мы и так все можем! Кстати, на Западе многие ученые тоже восприняли новый термин с прохладцей.

Понадобились годы, чтобы понять, что системное программирование разделилось на две части: более теоретическую информатику (*Computer Science*) и более практическую программную инженерию (*Software Engineering*). Сегодня вполне обособленно развиваются также науки, связанные с базами данных, информационным поиском и многие другие.

Программная инженерия

Более 20 лет использование терминов *Computer Science* и *Software Engineering* было полностью делом индивидуального вкуса. Например, когда в 1996 г. создавалась новая кафедра в СПбГУ, которой я руковожу до сих пор, она по-русски называлась "Кафедра системного программирования", а по-английски – "*Software Engineering Chair*". Уже тогда я хотел подчеркнуть существенно большую практическую направленность новой кафедры по отношению к "Кафедре математического обеспечения ЭВМ", первым выпускником которой я был. Кстати, именно в момент создания новой кафедры старая была переименована в "Кафедру информатики" (*Computer Science*).

С информатикой, т.е. с теоретическим программированием, вроде бы все было ясно. Уже в 1991 г. была опубликована первая международная программа обучения этой науке [4], в 2001 г. эта программа была пересмотрена [5]. По крайней мере, все основные определения были зафиксированы, сегодня смешно трактовать информатику не так, как ее определяет международный стандарт.

С программной инженерией все было сложнее. Многие ее направления, особенно связанные с планированием, бюджетированием и управлением воспринимались "в штыки" университетским математическим сообществом. Приходилось буквально прятать эти предметы в спецкурсы с более общими названиями, примерно так же, как это делали генетики ЛГУ за 50 лет до этого. Инженерия не может изучаться в классическом университете! Ей место в технических вузах! Я прятался за термином "технология программирования" [6], даже как-то раз поплакался Д. Кнуту. Его горячая поддержка мне сильно помогла.

В 2004 г. была, наконец, опубликована международная программа обучения программной инженерии [7], в которой, так же как и в программе по информатике, были даны соответствующие определения:

- установление и использование правильных инженерных принципов (методов) для экономичного получения надежного и работающего на реальных машинах программного обеспечения [8];
- программная инженерия является такой формой инженерии, которая применяет принципы информатики и математики для получения рентабельных решений в области программного обеспечения;
- применение систематического, дисциплинированного, поддающегося количественному определению подхода к разработке, эксплуатации и сопровождению программного обеспечения [9].

Эти определения можно трактовать в чуть более упрощенном виде: программная инженерия – это наука, которая изучает вопросы создания, сопровождения и внедрения программного обеспечения с заданным качеством, в заданные сроки и в рамках заранее определенного бюджета.

Международные образовательные стандарты

Кажется, основные международные организации IEEE и ACM, имеющие отношение к компьютерам и программированию, всерьез взялись за стандартизацию. В 2005 г. был опубликован стандарт *Computing* [10], включающий в себя:

- *Computer Engineering* (Разработка компьютеров);
- *Computer Science* (Информатика);
- *Information Systems* (Информационные системы);
- *Information Technology* (Информационные технологии);
- *Software Engineering* (Программная инженерия).

В этом стандрате достаточно точно и ясно описано, что следует понимать под каждой из пяти перечисленных наук.

Понятно, что "Разработка компьютеров" (стандарт опубликован в 2004 г.) ориентирован на тех, кто придумывает и создает новые компью-

теры для самых разных применений. В наш век FPGA, VHDL и Verilog этой работой все больше занимаются математики, хотя роль инженеров по-прежнему велика.

Пожалуй, стандарт по информатике наиболее полный и проработанный. В каком-то смысле он является базовым, поскольку на него ссылаются и используют другие стандарты. В 2008 г. он был пересмотрен, но, на мой взгляд, изменения не так существенны, как в 2001 г. по сравнению с 1991 г.

Стандарт "Информационные системы" (опубликован в 2002 г., пересмотрен в 2010 г.) чем-то похож на то, что в советские времена мы называли АСУ (автоматические системы управления).

Стандарт "Информационные технологии" (опубликован в 2006 г., пересмотрен в 2008 г.) для меня лично оказался самым загадочным, по крайней мере, под этим термином я понимал нечто другое. Это стандарт для тех, кто строит компьютерные сети, используя маршрутизаторы, прокси-серверы, firewalls и т.д. В России таких специалистов называют системными администраторами, а большие организации – системными интеграторами.

Поскольку программная инженерия является основной для данной статьи, разберем стандарт, опубликованный в 2004 г., подробнее. В стандарте *Software Engineering Curricula* 2004 [11] рассказана история создания этого документа, основные принципы и цели обучения. Я настоятельно рекомендую своим студентам полистать стандарт, но вовсе не с целью узнать программы отдельных курсов, это, скорее, нужно преподавателям. На мой взгляд, самая интересная часть стандарта – это короткие эссе, в которых объясняется, что общего и разного между программной инженерией и традиционной инженерией, между программной инженерией и информатикой, между информатикой и математикой. Что-то я не помню в российских образовательных стандартах упоминаний о важности участия в профессиональных сообществах, соблюдения профессиональной этики, умения работать в коллективах, не создавая конфликтов, и т.д. Именно благодаря участию в переводе текста стандарта на русский язык я познакомился с классификацией Блума (знание, понимание, применение), узнал о такой полезной вещи, популярной на Западе, но, к сожалению, не у нас, как SWEBOK [12] – что должен знать и уметь программный инженер через 4 года после окончания вуза.

Итак, программная инженерия включает в себя следующие области знаний:

- основы компьютинга (основы информатики, технологии и средства разработки, формальные методы);
- основы математики и инженерии (в том числе инженерная экономика ПО);
- профессиональная практика (работа в команде, навыки коммуникации, этика);
- основы моделирования (анализ, работа с требованиями, спецификации);
- проектирование ПО (концепции и стратегии проектирования, проектирование человеко-машинного интерфейса, средства поддержки проектирования);
- верификация и аттестация ПО (основы, рецензия кода, тестирование, оценка пользовательского интерфейса, анализ проблем);
- эволюция ПО (на мой взгляд, совершенно справедливо в стандарте говорится об эволюции, а не о сопровождении ПО);
- процессы разработки ПО;
- качество ПО (стандарты качества ПО, процессы обеспечения качества ПО, процесса, продукта);
- управление программными проектами (концепции менеджмента, планирование и отслеживание выполнения проектов, управление персоналом, управление конфигурацией ПО).

Каждая из перечисленных 10 областей подробно расписана, снабжена примерной оценкой временных затрат, приведены программы соответствующих курсов, часто в нескольких вариантах.

Привязка к российским образовательным стандартам

Стандарт *Software Engineering Curricula* 2004 разработан очень демократично, предусмотрено множество траекторий обучения, приводятся различные программы отдельных курсов для технических вузов и классических университетов, даны конкретные рекомендации. Приводятся даже примерные учебные планы с разбивкой по годам для разных стран. К сожалению, Россия в эти примеры не попала, поэтому мы с Андреем Тереховым (младшим) взяли на себя труд и риск разработки примерного плана на основе российского образовательного стандарта 01.04 (Информа-

ционные технологии). Во всех случаях, когда *Software Engineering Curricula* 2004 предлагает какой-то выбор, мы на основании консультаций со многими коллегами такой выбор делали, в тех случаях, когда подходящие курсы в 01.04 находились, мы этим пользовались (с необходимыми уточнениями), если же аналогов не было, то добавляли новые курсы (иногда для этого приходилось немного сокращать российский стандарт). Весь этот процесс подробно описан в [13].

Должен сказать, что стандарт 01.04, подготовленный под руководством профессора В.А. Сухомлина (факультет ВМ и КМГУ), оказался достаточно хорошей основой для выравнивания с международным стандартом. Во-первых, В.А. Сухомлин опирался на *Computer Science Curricula* 2001, а во-вторых, в нем было предусмотрено достаточно много часов для курсов по выбору вуза, студента, для самостоятельной работы. Мы этим воспользовались.

В настоящее время (лето 2010 г.) накоплен четырехлетний опыт преподавания по этой программе. К сожалению, выдержать план обучения с точностью до семестра не удалось, многие профессиональные курсы были сдвинуты на 1–2 семестра ближе к концу бакалаврской программы. Тому есть две важные причины:

- Традиционно высокий уровень математической подготовки, который потребовал существенно больше часов на младших курсах.
- Военная кафедра, которая отнимала целый день в неделю на 2–4-х курсах. Почему-то в западных университетах такой предмет не предусмотрен.

Недавно был принят закон об особом статусе Московского и Санкт-Петербургского университетов, в рамках которого им можно устанавливать свои образовательные стандарты, что мы, естественно, и сделали. Но в этом же законе сказано, что наши университеты должны играть ведущую роль при подготовке программ обучения для других вузов России.

Дополнительное производственное обучение

Стандарт *Software Engineering Curricula* 2004 достаточно полон и конструктивен. Меня очень порадовало, что в нем много внимания уделяется производственной практике, работе над проектами, подчеркиваются даже важность и преимущества коллективной работы над дипломным проектом.

Мы к подобным принципам пришли более десяти лет назад. Я и сейчас не верю, что можно научить составлению недельных отчетов, управлению конфигурацией ПО, QA (обеспечение качества), работе в команде и т.п. в классе у доски.

Нужно как-то организовывать работу студентов в небольших коллективах над реальными проектами. Но что значит "реальный"? Производственный проект ни один уважающий себя заказчик студентам не отдаст, исследовательские проекты требуют большого внимания научного руководителя. Например, наша кафедра отвечает за подготовку примерно 50 студентов на каждом из третьих, четвертых и пятых курсов. Где мне взять несколько десятков квалифицированных научных руководителей для 150 студентов ежегодно?

Поскольку параллельно с преподаванием в Университете я еще руковожу довольно большим предприятием, занимающимся наукоемким бизнесом в области информационных технологий, ответ на поставленный вопрос почти очевиден. Пусть сотрудники предприятия руководят группами студентов. Но зачем им это надо? Всегда на сотню сотрудников найдется два-три человека, которым нравится преподавать, даже бесплатно, но вряд ли на этой основе можно наладить регулярную массовую подготовку кадров.

Как обычно, для решения, которое устроило бы разные стороны, необходимо совпадение многих интересов и обстоятельств. В данном случае мне на руку сыграло то обстоятельство, что в начале двухтысячных годов рынок кадров в нашей области деятельности был сильно перегрет. Даже весьма слабые специалисты с сильно завышенными требованиями были нарасхват. Еще более трудной представляется проблема удержания кадров. В Санкт-Петербурге в программистских организациях текучка кадров составляет 15–20 %.

Таким образом, нужно решать две задачи – нахождение и удержание кадров одновременно [14]. Мы видим это решение следующим образом [15]:

- во втором семестре второго курса мы приглашаем студентов принять участие в 10–12 проектах, каждым из которых руководят не менее двух сотрудников предприятия. Например, в этом году мы предлагали разработку приложений для *iPhone и Android*, реализацию ОС реального времени, разработку технологии разработки ПО на PHP, CASE-средства на базе библиотеки *Qt*, раз-

работку нашей версии *mindmap* (Comapping.com), нескольких игр. Никаких коммерческих результатов не ожидается (но если что-то получится, никто не будет против), организация проектов вполне "взрослая" (планирование, версионный контроль, еженедельные отчеты, QA и т.д.);

• летом после второго курса для лучших студентов мы организуем летнюю школу (1 месяц по 4 часа в день), причем платим им дополнительную стипендию, чтобы как-то компенсировать частичную потерю летнего отпуска;

• начиная с третьего курса мы постепенно, по мере появления вакансий, приглашаем студентов на стажерские позиции в предприятии.

Накопленная за десять лет статистика свидетельствует, что студенты, пришедшие после такой подготовки на работу в наше предприятие, работают у нас много лет, быстро растут профессионально и карьерно. Более того, если считать только прямые затраты, оказывается, что находить сотрудников традиционным способом через службу HR обходится существенно дороже затрат на обучение и привлечение студентов. Дело в том, что нового специалиста нужно еще 1,5–2 месяца готовить, вводить в коллектив, отвлекая для этого других сотрудников.

Таким образом в выигрыше оказываются все:

• студенты имеют возможность бесплатно получать дополнительную подготовку, освоить навыки работы в производственном коллективе, изучить новейшие технологии, заранее подыскать себе место работы по душе;

• предприятие получает готовых специалистов, хорошо знающих порядки на предприятии;

• университет получает бесплатно опытных руководителей курсовых и дипломных работ и доступ к новейшим актуальным технологиям.

Заключительные замечания

Программная инженерия – это то, чему мы и наши партнеры учим студентов днем в Университете, а вечером на производстве.

Я всячески приветствую коллективные работы и преемственность курсовых работ с переходом в дипломные. С этой целью я подталкиваю сотрудников на масштабные, даже амбициозные студенческие проекты, разумеется, с выделением начальных, более простых этапов. Чаще всего такой подход дает хорошие результаты.

Сегодня наш метод производственного обучения взяли на вооружение Санкт-Петербургские отделения *Intel, Yandex, Google*, EMC, *Exigen* и некоторые другие компании, несколько тем предложила компания Транзас, активно работает с нашими студентами *IntelliJ*. Таким образом, можно сказать, что способ подготовки хороших программных инженеров, разработанный нами, вышел из стадии экспериментов и доказал свои преимущества.

СПИСОК ЛИТЕРАТУРЫ

1. **Китов А.И., Криницкий Н.А.** Электронные цифровые машины и программирование. Изд. второе. М.: Физматгиз, 1961.

2. **Терехов А.Н.** Как я общался с А.П. Ершовым в формальной и неформальной обстановке // Сб. тр. конф. Перспективы систем информатики. Новосибирск, 2010.

3. **Naur P., Randell B. (Eds.).** Software Engineering // Proc. of Report on a Conference Sponsored by the NATO Science Committee (7–11 October 1968). Brussels. Scientific Affairs Division. NATO. 1969.

4. **Computing** Curricula 1991. New York: ACM Press, 1991.

5. **Computing** Curricula 2001. Computer Science, 2001.

6. **Терехов А.Н.** Технология программирования. М.: ИНТУИТ:БИНОМ, 2007.

7. **Software** Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering.

8. **Bauer F.L.** Software Engineering // Information Processing. N 71. 1972.

9. **IEEE STD** 610.12–1990. IEEE Standard Glossary of Software Engineering Terminology // IEEE Computer Society. 1990.

10. **http:**//www.acm.org/education/education/curric_vols/CC2005-March06Final.pdf.

11. **Рекомендации** по преподаванию программной инженерии и информатики в университетах. М.: ИНТУИТ, 2007.

12. **Software** Engineering Coordinating Committee. Guide to the Software Engineering Body of Knowledge. IEEE Computer Society, 2001.

13. **Терехов А.Н., Терехов А.А.** Computing Curricula: Software Engineering и российское образование // Открытые системы. 2006. № 8.

14. **Terekhov A., Terekhova K.** The Economics of Hiring and Staff Retention for an IT Company in Russia // Proc. of 4th International Conference SEAFOOD. 2010.

15. **Гагарский Р.К.** Программа подготовки специалистов в IT-компании // Системное программирование. Вып. 3. Сб. статей. СПб: Изд. СПбГУ, 2008.

Государственный университет Высшая школа экономики подписал соглашение с IEEE CS

Государственный университет Высшая школа экономики (ГУ ВШЭ), один из ведущих национальных исследовательских университетов России, 6 мая 2010 г. в г. Москве подписал соглашение с IEEE CS (*Institute of Electrical and Electronics Engineers – Computer Society*, компьютерным обществом института инженеров электротехники и электроники) о профессиональной международной аттестации студентов по программе "Сертифицированный специалист по разработке программного обеспечения" (*Certified Software Development Associate*, CSDA).

Таким образом, ГУ ВШЭ стал еще одним в мире и первым в России высшим учебным заведением, принявшим к действию не только руководство "Совокупность знаний по программной инженерии" (*Software Engineering Body of Knowledge*, SWEBOK), но и программу аттестации CSDA IEEE CS.

В соответствии с подписанным соглашением студенты отделения программной инженерии ГУ ВШЭ будут проходить аттестацию по программе CSDA. Подготовка к аттестации будет способствовать приобретению знаний и навыков в области программной инженерии на международном уровне и позволит выпускникам отделения быть конкурентоспособными во всем мире.

"Эта сертификация имеет международное признание и будет подтверждать, что наши выпускники, имеющие такие сертификаты, являются высококвалифицированными специалистами, профессионалами по программной инженерии, способными внести свой вклад в становление инновационной экономики России", – говорит профессор С. Авдошин, руководитель отделения программной инженерии ГУ ВШЭ.

Программа CSDA предназначена для разработчиков программного обеспечения, начинающих свою карьеру после окончания бакалавриата. Она является одной из двух программ сертификации профессиональных разработчиков программного обеспечения, предоставляемых IEEE CS. Вторая программа – "Сертифицированный профессионал по разработке программного обеспечения" (*Certified Software Development Professional*, CSDP) – предназначена для специалистов профессионального уровня. Такую сертификацию смогут пройти выпускники магистерской программы "Управление разработкой программного обеспечения" отделения программной инженерии, поскольку в учебный план магистратуры начиная с первого курса входит практика работы в компаниях – лидерах ИТ-индустрии. Подробную информацию по сертификациям можно найти на сайте: http://www.computer.org/getcertified.

Обе программы (CSDA и CSDP) основаны на руководстве SWEBOK, всеобъемлющем документе, содержащем профессиональные стандарты отрасли и общепринятые принципы программной инженерии. Впервые разработанное в 2004 году руководство SWEBOK в настоящее время находится в процессе совершенствования. Более полную информацию по руководству SWEBOK можно найти на сайте: http:www.computer.org/swebok.

Сертификаты CSDA и CSDP признаются крупнейшими компаниями мира, такими как *Accenture, Boeing, Borland Software, Cisco Systems, Ernst & Young LLP, Hewlett Packard, HP Corporation, IBM Corporation, IBM Global Services, Infosys Technologies, Intel Corporation, Microsoft Corporation, Oracle Palm, Samsung, Schlumberger, Siemens* и др.

Приказами Минобрнауки РФ № 542 (бакалавриат), № 543 (магистратура) от 09 ноября 2009 г. с начала 2010 г. введены в действие Федеральные государственные образовательные стандарты (ФГОС) высшего профессионального образования по новому направлению 231000 "Программная инженерия". ГУ ВШЭ получил первую в России лицензию на право подготовки бакалавров и магистров по направлению 231000 Программная инженерия (приказ Рособрнадзора № 1093 от 05 мая 2010 г.), и уже в 2010 г. проведен прием студентов на новое направление.

"При разработке программ для подготовки бакалавров и магистров мы использовали рекомендации руководства SWEBOK и международных образовательных стандартов *Computing Curricula* 2005, *Computer Science* 2001 и *Software Engineering* 2004. Это должно способствовать карьере наших выпускников по программной инженерии", – говорит профессор С. Авдошин, – "Наши студенты приобретают компетенции в экономике, управлении, компьютерных науках и разработке программного обеспечения, что помогает им стать отличными специалистами в области программной инженерии и руководителями программных проектов".

Учебные планы новой образовательной программы получили признание не только в России, но и за рубежом. Они прошли международную экспертизу с участием официальных представителей IEEE CS на соответствие международным рекомендациям по преподаванию программной инженерии в высших учебных заведениях.

В настоящее время в России специалисты в области программной инженерии весьма востребованы. "Инновационная экономика требует высокообразованных специалистов по программной инженерии в различных отраслях промышленности и науки", – говорит С. Авдошин, – "Так как программные продукты разрабатываются на основе международных стандартов, то люди, которые создают эти продукты, должны получать образование, также основанное на международных стандартах. Программа CSDA дает хорошую возможность оценить соответствие уровня подготовки профессиональным международным стандартам".

О Государственном университете Высшая школа экономики

ГУ ВШЭ, национальный исследовательский университет России, находится в Москве и имеет филиалы в Санкт-Петербурге, Нижнем Новгороде и Перми. В настоящее время в университете обучается около 20 000 студентов, в том числе – около 17 000 будущих бакалавров, около 2000 будущих магистров и 576 аспирантов. В числе 1500 профессоров и преподавателей имеются специалисты из сфер бизнеса, исследовательских институтов и государственных учреждений, 345 докторов и 909 кандидатов наук.

ГУ ВШЭ был первым университетом в России, который внедрил принятую во всем мире практику "4+2": четыре года для подготовки бакалавров и два года для подготовки магистров.

Отделение программной инженерии было создано в ГУ ВШЭ в 2006 г. Студенты отделения успешно участвуют в конкурсах и конференциях по компьютерным наукам и программной инженерии в России и за рубежом.

О Компьютерном обществе IEEE

Компьютерное общество IEEE (*Institute of Electrical and Electronics Engineers – Computer Society*) является лидирующей в мире организацией профессионалов в области вычислительной техники, насчитывающей около 85 000 членов. Основанное в 1946 г. и являющееся самым большим из 39 обществ IEEE, компьютерное общество нацелено на развитие теории и практики компьютерных и информационных технологий, является всемирно известным благодаря своей деятельности в области развития компьютерных стандартов.

Компьютерное общество IEEE обслуживает информационные и профессиональные запросы современных исследователей и практиков в области вычислительной техники, издавая журналы, сборники, проводя конференции и публикуя их материалы, предоставляя книги и сетевые курсы. Проводимые обществом аттестации по программам CSDP (для профессионалов) и CSDA (для специалистов) подтверждают уровень навыков и образования тех, кто работает в соответствующей области. Цифровая библиотека Компьютерного общества (CSDL), являющаяся отличным инструментом для исследователей, содержит более 250 000 статей и материалов 1600 конференций, а также 26 периодических изданий Компьютерного общества начиная с 1988 г.

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 08.07.10 г. Подписано в печать 07.09.10 г. Формат 60 88 1/8. Бумага офсетная. Печать офсетная.
Усл. печ. л. 5,88. Уч.-изд. л. 5,80. Заказ 671. Цена свободная.

Отпечатано в ООО "Подольская Периодика". 142110, Московская обл., г. Подольск, ул. Кирова, 15.