ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ


Издается с сентября 2010 г.



## СОДЕРЖАНИЕ



**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 004.4

**Б.А. Позин,** д-р техн. наук, проф., ЗАО "ЕС-лизинг"

E-mail: contact@ec-leasing.ru

# ПРОБЛЕМЫ ПРОГРАММНОЙ ИНЖЕНЕРИИ НА СОВРЕМЕННОМ ЭТАПЕ ЕЕ РАЗВИТИЯ

*Работа посвящена анализу актуальных проблем программной инженерии на современном этапе ее развития. Рассмотрен предложенный в своде знаний по программной инженерии (SWEBOK) перечень процессов, входящих в эту область знаний. Особое внимание уделено процессу сопровождения программного обеспечения информационных систем в промышленном режиме, его особенностям в распределенных организациях и сопутствующим процессам.*

***Ключевые слова:*** *сопровождение программного обеспечения информационных систем, свод знаний по программной инженерии, управление требованиями, функциональное тестирование при сопровождении, нагрузочное тестирование при сопровождении*

Программная инженерия как область знаний сформировалась в последние 40–45 лет на основе работ, которые проводились как научные и прикладные исследования отечественными и зарубежными учеными. Эти работы были направлены на обеспечение больших государственных проектов (сначала преимущественно в оборонной области) и проектов крупных компаний в тех сферах человеческой деятельности, где создавались информационные системы различного назначения. Особую важность методы и автоматизированные технологии создания и сопровождения программного обеспечения имели для тех секторов национальной экономики, автоматизация которых существенно повышала эффективность лежащих в их основе бизнес-процессов. К их числу относятся банковская и финансовая деятельность, телекоммуникации, транспорт, оптовая торговля и дистрибуция, топливно-энергетический комплекс, управление объектами, процессами и ряд других.

Работы в области программной инженерии проводились и в нашей стране. Уже в 80-е годы прошлого века было реализовано несколько государственных программ в этой области [2]. В 1980–1990 гг. прошла целая серия научно-технических конференций по технологии программирования (первая и вторая конференции в Киеве, а также в Москве, Санкт-Петербурге, Новосибирске, Твери, Ереване, Риге, Таллине и других городах). В 1992 г. создана Ассоциация по программной инженерии, которая провела три конференции по CASE-технологии в Москве.

**SWEBOK.** Работа ученых во всем мире и накопление огромного опыта создания и многолетнего массового использования программных средств разного назначения привели к тому, что этот опыт обобщен ISO, IEEE в виде свода знаний в области программной инженерии (SWEBOK)[1]. В этом документе зафиксированы базовые достижения, отражающие переход программной инженерии на промышленный уровень. В нем систематизированы знания международного сообщества в области программной инженерии, описаны основные процессы и методы в этой области знаний, а также смежные области знаний в разрезе основной концепции ISO/IEC: реализации процессного и проектного подходов при создании систем и программного обеспечения на основе концепции тотального управления качеством. Сформулированы и описаны по составу и содержанию процессы программной инженерии.

- ***Software requirements*** – требования к программному обеспечению (ПО);
- ***Software design*** – проектирование ПО;
- ***Software construction*** – разработка ПО;
- ***Software testing*** – тестирование ПО;
- ***Software maintenance*** – сопровождение ПО;
- ***Software configuration management*** – конфигурационное управление ПО;
- ***Software engineering management*** – управление в программной инженерии;

- ***Software engineering process*** – процессы программной инженерии;
- ***Software engineering tools and methods*** – инструменты и методы программной инженерии;
- ***Software quality*** – качество программного обеспечения.

Вместе с тем не следует относиться к SWEBOK как к чему-то застывшему и незыблемому. В области создания и использования программного обеспечения остаются проблемы, которые не нашли пока должного отражения в этом документе. Появились новые вызовы рынка, требующие осмысления и разрешения с использованием методов, технологий и инструментальных средств, основанных на положениях программной инженерии.

**Особенности современного этапа в развитии программной инженерии.** В настоящее время в организациях (компаниях) разного масштаба и сфер деятельности активно эксплуатируется огромное число программных продуктов – как поставляемых пакетов прикладных программ, так и ПО информационных систем (ИС), непосредственно участвующих в бизнес-процессах этих компаний. По оценкам специалистов, стоимость сопровождения этого ПО составляет порядка 70...80 % от общей стоимости работ в его жизненном цикле (ЖЦ) и существенно влияет на совокупную стоимость владения информационными системами Заказчика. Решение вопросов снижения совокупной стоимости владения информационными системами и их программным обеспечением, управления качеством в жизненном цикле ПО и ИС является в настоящее время чрезвычайно актуальным. Соответственно, актуальными являются и аналогичные проблемы программной инженерии.

Целесообразно рассмотреть виды объектов программной инженерии, субъекты, участвующие в процессах ЖЦ этих объектов и особенности их ЖЦ. К видам объектов программной инженерии следует отнести:

- заказное ПО, в том числе ПО ИС, включая портируемое на различные платформы;
- типовое прикладное ПО, настраиваемое на условия применения (например, 1С, SAP R3 и др.);
- системное и инструментальное ПО, поставляемое серийно (ОС, middleware, компиляторы, GIS-системы и т.п.);
- исследовательское ПО.

Глобализация бизнеса приводит к тому, что всё больше накапливается массовый опыт использования программных средств разного типа **в производственных условиях**, в том числе в территориально распределенных компаниях. Для таких компаний характерна четкая структуризация, регламентация распределения обязанностей и ответственности как внутри компании, так и между компанией и ее подрядчиками, выполняющими работы по сопровождению и обслуживанию ИС, их ПО, в том числе и в условиях территориальной удаленности участников процесса. Необходимо развитие новых методов, методик и инструментов для повышения эффективности труда заказчиков, разработчиков, специалистов, сопровождающих далее (сопроводителей) и эксплуатирующих далее (эксплуатационников) ПО и снижения общей стоимости работ этих специалистов.

В подобных ситуациях основные проблемы ЖЦ ПО проявляются, пожалуй, наиболее ярко. По этой причине соответствующие проблемы будут рассматриваться, прежде всего, для программного обеспечения ИС.

**Проблемы сопровождения ПО ИС.** Для ПО этого вида наиболее существенными становятся процессы сопровождения, поскольку за время жизни ИС, как правило, достаточно часто вносятся изменения, а цена ошибки определяется финансовыми, материальными и другими потерями бизнеса, которые возникают в результате ошибок в ПО ИС. Именно поэтому приходится переосмысливать роль ряда процессов (управление требованиями, управление конфигурациями и изменениями и т.д.) при сопровождении ПО. Кроме того, крайне важно четкое понимание ролей субъектов программной инженерии (функциональных и фактических заказчиков ПО, разработчиков, сопроводителей, эксплуатационников) в процессах сопровождения и эксплуатации ПО ИС.

Процесс сопровождения ПО ИС протекает на фоне эксплуатации ИС. Именно поэтому процесс сопровождения ПО ИС необходимо рассматривать с позиций модели услуг, предусмотренной в ITSM (*Information technology – Service management*). Этот факт означает, что необходимо оценить совместимость процессов комплекса стандартов [3–8]. Результаты совместного анализа процессов сопровождения и процессов, предусмотренных в перечисленных стандартах, приведены в таблице.

Проведенный анализ показывает, что процессы сопровождения целесообразно "вписать" в указанную модель услуг, причем в качестве основного процесса сопровождения следует рассматривать процесс управления выпусками (*Software Release Management*). Такой подход позволит осуществлять планирование деятельности службы сопровождения и контроль качества всех остальных процессов, включая управление требованиями.

Если команда сопроводителей географически распределена, а ИС имеет несколько объектов базирования, в организации – владельце ИС требуется вести управление сопровождением ПО ИС более высокого уровня.

Такой подход обеспечивает возможность оперативно проводить мониторинг и управление ИС в целом, в которой ПО является одним из многих, но структурно сложным компонентом. Значительная часть процессов сопровождения может быть автоматизирована на основе комплексов инструментальных средств для автоматизации разработки. Однако достижение состояний готовности выпуска ПО ИС и его компонентов, а также состав реализованных регламентированных действий сопроводителей и аутсорсеров должны фиксироваться автоматически в целях использования этой

**Сопоставление процессов по ГОСТ 12207, 14764, 15504 и 20000**

| Процессы жизненного цикла ПО ИС по ГОСТ Р ИСО/МЭК 12207 | Процессы по ГОСТ Р ИСО/МЭК 20000 | | | | |
|---|---|---|---|---|---|
| | Управление инцидентами | Управление проблемами | Управление изменениями | Управление выпусками | Управление конфигурацией |
| **Основные процессы** | | | | | |
| Приобретение | | | | | |
| Поставка | | | | | |
| Разработка, в том числе управление требованиями (категория ENG 1 по ГОСТ 15504) | | | | | |
| Эксплуатация | | | | | |
| Сопровождение | | | | | |
| **Вспомогательные процессы** | | | | | |
| Документирование | | | | | |
| Управление конфигурацией | | | | | |
| Обеспечение качества | | | | | |
| Верификация | | | | | |
| Аттестация | | | | | |
| Совместный анализ | | | | | |
| Аудит | | | | | |
| Решение проблем | | | | | |
| **Организационные процессы** | | | | | |
| Управление | | | | | |
| Создание инфраструктуры | | | | | |
| Усовершенствование | | | | | |
| Обучение | | | | | |

информации при управлении выпусками, изменениями и конфигурацией модифицируемого ПО ИС.

**Обеспечение и управление качеством в ЖЦ ПО.** Рассмотрение процессов обеспечения и управления качеством ПО приводит к необходимости понимания того, как меняется ПО в качестве объекта системы управления качеством при переходе с этапа на этап ЖЦ ПО, а также каковы свойства процессов ЖЦ ПО, в рамках которых должно осуществляться управление качеством ПО. Понятно, что основные мероприятия по обеспечению качества должны проектироваться в соответствии с подходом, заложенным в CMMI [5].

Вместе с тем целесообразно учесть, что на этапах сопровождения и эксплуатации само ПО ИС представляется уже не в детальной структуре, как на стадии проектирования, а в виде укрупненных компонентов, размещаемых на объектах базирования ИС и обладающих определенной функциональностью, понятной на уровне ИС. Степень воздействия на каждый такой компонент при реализации процессов управления качеством различная и отличается от тех возможностей, которые существуют на стадии разработки. В настоящее время серьезные исследования по этой тематике пока не опубликованы, а само направление является чрезвычайно актуальным.

Особенности процессов обеспечения и управления качеством таких компонентов при сопровождении и эксплуатации пока также изучены недостаточно. Требуют уточнения модели и показатели качества, требования к качеству, свойства и характеристики дефектов, состав используемых метрик, методики их измерения, а также методики управления качеством. Пока еще не приобретен достаточный опыт применения стандарта [9] при выполнении больших программных проектов, а именно в ходе сопровождения и эксплуатации на длительном временном интервале ПО, созданного в рамках выполнения таких проектов.

**Управление процессами в распределенных организационных структурах.** Особой сложностью обладает сопровождение ПО ИС, осуществляемое географически распределенными коллективами. Такие проекты характерны для крупных заказчиков – промышленных предприятий и больших территориально распределенных компаний. В этом случае особенно важно четкое распределение обязанностей и ответственности между участниками за процессы сопровождения и отдельные артефакты этих процессов (требования, исходные тексты и т.п.). Принятые решения по распределению обязанностей и ответственности необходимо закреплять путем их регламентации, а именно выпуска внутри компании – владельца ИС документа, обязательного для выполнения всеми службами, вовлеченными в процессы сопровождения и эксплуатации. При привлечении аутсорсеров для выполнения определенных процессов и операций элементы регламентов должны включаться в соответствующие договоры. Необходимо также систематическое выполнение аудита хода процессов и систематическая оценка сос-

тояния процесса в целом (статус-митинги путем телеконференций). Только в этом случае чрезвычайно полезной, определяющей с позиций оперативности и эффективности процессов сопровождения, является автоматизация процессов сопровождения и эксплуатации ПО ИС.

В связи с изложенным очень востребованы разработки метрики процессов распределенного сопровождения и общие показатели качества процессов сопровождения и эксплуатации ПО ИС.

**Модели организации работ по сопровождению и эксплуатации ПО ИС.** При организации процессов сопровождения и эксплуатации ПО ИС в различных компаниях–владельцах ИС, эксплуатирующих системы как по централизованной, так и по распределенной схеме, важно использовать корректную модель деятельности, учитывающую интересы всех заинтересованных участников процесса. Наиболее важными при построении такой модели является четкое разделение обязанностей и сфер ответственности участников в условиях конкретной компании.

Типовая модель должна учитывать взаимоотношения (обязанности и ответственность):

- функционального заказчика (бизнеса);
- фактического заказчика (ИТ-службы);
- разработчика ПО ИС;
- сопроводителя (например, подразделения ИТ-службы), отвечающего за сопровождение ПО;
- оператора (службы, обеспечивающей эксплуатацию ИС);
- конечного пользователя, применяющего ПО ИС по назначению в бизнес-процессе (складской работник, экономист и т.п.);
- аутсорсера – организацию, которая на основе договора выполняет определенную роль в процессе сопровождения.

При построении процесса сопровождения с использованием стандарта 14764 определяют схему взаимодействия между ролями, предусмотренными в модели, фиксируя операции, переходы и артефакты, используемые в ходе сопровождения, а также временные и ресурсные ограничения на их выполнение, возможности их автоматизации. Несмотря на типовую схему таких моделей, при их адаптации к условиям каждой компании учитывается ее специфика, уровень зрелости, определяющий возможность реализации в компании тех или иных операций или процесса в целом.

**Инжиниринг и управление требованиями в ЖЦ ПО.** Информационная система создается для автоматизации бизнес-процессов заказчика. В этой связи требования к ИС и ПО ИС на стадии разработки должны формироваться поэтапно. В то же время при сопровождении требования могут уточняться и изменяться в связи с изменениями нормативной базы заказчика, диверсификацией его бизнеса, развитием его бизнес-процессов. Интенсивность изменения требований может достигать нескольких сотен в год. С точки зрения управления требования важно классифицировать по видам и по использованию в процессах ЖЦ ПО.

Требования по видам можно классифицировать на:

A. *Нормативные требования,* т. е. определяемые нормативными документами.

B. *Функциональные требования к бизнес-процессу.*

C. *Функциональные требования к ИС и ее ПО.*

D. *Нефункциональные требования* к показателям назначения ИС (доступность, производительность и т.п.).

E. *Требования к реализации ПО*, которые вырабатываются в процессе проектирования.

F. *Тестовые требования* к ПО ИС в целом.

Различные виды требований используются на разных этапах ЖЦ ПО, в том числе при сопровождении и эксплуатации. В качестве требований к реализации на этих этапах может применяться и документация на ИС или ПО, например, альбомы форм документов, выпускаемых ПО. Требования C, E, F используются при функциональном тестировании выпусков ПО. Требования D, E используются при нагрузочных испытаниях и тестировании, которые проводятся при необходимости оценить потенциальные характеристики модернизации технического обеспечения ИС (вычислительных комплексов, сети и т.п.).

Инжиниринг и управление требованиями при сопровождении позволяют заказчикам четко понимать, какими реальными характеристиками обладает ПО ИС текущего выпуска, а также спланировать достижение заданной функциональности в определенное время и при ограниченных ресурсах. Процессы инжиниринга и управления требованиями автоматизируются средствами работы с требованиями, из проверки, документирования и анализа. Заметим, что если требования к ИС или ПО ИС определены плохо или не определены, при тестировании приходится восстанавливать требования до уровня, позволяющего правильно организовать тестирование, особенно автоматизированное. Для систем, в которых требования к ПО определены хорошо (например, сетевые протоколы) удается автоматизировать генерацию тестов.

**Функциональное и нагрузочное тестирование при сопровождении и развитии ПО ИС.**

При сопровождении ПО ИС приходится периодически решать две принципиально различные задачи:

- проверять соответствие внесенных изменений спецификации (функциональным требованиям или требованиям к реализации) тому, какими новыми функциями должен обладать новый выпуск ПО ИС;
- проверять, будет ли после внесения изменений в ПО ИС сама ИС обладать необходимой производительностью, сможет ли ИС в течение длительного времени на том же оборудовании обеспечивать обработку перспективной нагрузки с заданной оперативностью (или производительностью).

***Первая задача.*** В этом случае речь идет о свойствах не отдельных компонентов, а о свойствах ПО ИС в целом. Объектом функционального тестирования является выпуск ПО ИС. При сопровождении появляется не-

обходимость в поставке ПО и автоматизирующего процессы регрессионного функционального тестирования ПО ИС на определенном комплексе тестов, который осуществляет проверку по сценариям. На практике удается построить комплексы тестов, обеспечивающие покрытие до 80 % функциональности ПО ИС. Комплексы тестов также требуют сопровождения, однако трудоемкость таких работ, например, для банковских систем составляет 2–3 чел.-год на одну ИС (ПО ИС). В процессе сопровождения должны быть развернуты комплексные стенды для автоматизированного тестирования и выделено специальное время для его проведения на регулярной основе.

***Вторая задача.*** Решение второй задачи сопряжено с необходимостью проверки функционирования ПО ИС на аппаратно-программном комплексе, максимально приближенном по составу и структуре к реальному комплексу, на котором функционирует ИС при реальном использовании. Это всегда достаточно дорогой полунатурный или натурный эксперимент, требующий длительного времени (от нескольких часов до нескольких суток). Однако и цена решений, которые будут приниматься по результатам такого эксперимента, тоже очень высока и на несколько порядков превышает стоимость нагрузочного эксперимента. Для проведения нагрузочного тестирования должен быть построен сложный аппаратно-программный комплекс, обеспечивающий настройку испытываемого ПО ИС и соответствующих баз данных, наполнение их тестовыми данными. Такой комплекс должен поддерживать генерацию потоков требований на тестируемую систему, подачу требований в нужном темпе, измерения характеристик функционирования ИС под воздействием тестовой нагрузки, анализ и представление результатов тестирования испытателям как в ходе эксперимента, так и по его завершении. Решение такой задачи регрессионно, с определенной периодичностью, позволяет проводить раннее обнаружение признаков деградации системы при внесении в нее изменений.

**Документирование ПО и ПО ИС.** В настоящее время чрезвычайно актуальным является повышение качества документации на ПО ИС и на отечественные реализации ПО различного типа. Требуют обновления стандарты групп 34 и 19, посвященные документированию автоматизированных систем и программного обеспечения, необходима их гармонизация со стандартами ИСО/МЭК по этой теме. Практика последних десятилетий привела к формированию общего представления о составе ролей персонала, использующего документацию на различных этапах ЖЦ ИС и ПО и их потребностях в материале, включаемом в состав документации.

**Совокупная стоимость владения ПО и *Open Source*.** В последние несколько лет на государственном уровне решается вопрос об использовании свободного ПО (*Open Source*) в качестве замены ряда операционных систем и офисных пакетов прикладных программ. Качество реализации программ *Open Source* очень высоко. Этот факт создает иллюзию того, что совокупные затраты на его обслуживание и сопровождение будут также минимальными или существенно меньшими, чем такие аналоги, как, например, Windows. Однако следует учесть, что документирование, обслуживание, обучение требуют значительных затрат. Кроме того, крайне важно, кто предоставляет гарантии качества свободного ПО, т.е. очень важен юридический аспект реализации такого подхода. Проблема использования свободного ПО является безусловно актуальной, однако она требует выработки условий и разработки комплекса услуг, которые позволят обеспечить его применение в различных сегментах рынка, в том числе в ряде ответственных приложений.

## СПИСОК ЛИТЕРАТУРЫ


1. **Software Engineering** – Guide to the Software Engineering Body of Knowledge (SWEBOK). ISO/IEC TR 19759: 2005 (E).

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

3. **ГОСТ Р ИСО/МЭК 12207—99.** Информационная технология. Процессы жизненного цикла программных средств. М.: Издательство стандартов, 1999.

4. **ГОСТ Р ИСО/МЭК 14764–2002.** Информационная технология. Сопровождение программных средств. М.: Издательство стандартов, 2002.

5. **ISO/IEC TR 15504:1998**. InformationTechnology – Software Process Assessment (parts 1–9). ISO and IEC, 1998.

6. **ISO/IEC TR 20000–1: 2005**. Information technology – Service management – Part 1: Service management system requirements.

7. **ISO/IEC TR 20000–2: 2005**. Information technology – Service management – Part 2: Code of practice.

8. **ISO/IEC TR 20000–3:2009**. Information technology – Service management – Part 3: Guidance on scope definition and applicability of ISO/IEC 20000–1.

9. **ISO/IEC 9126–1:2001**. Software Engineering Product Quality – Part 1: Quality Model. ISO and IEC, 2001.

УДК 004.41 (075.8)

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

E-mail: alexlip@mail.ru

# Инженерная психология при производстве компонентов программных продуктов

*Рассмотрены основные понятия инженерной психологии в коллективах, занимающихся производством сложных программных комплексов как направления, которое позволяет описывать психологические особенности лидеров таких коллективов, составляющих их производственных групп и отдельных специалистов.*

***Ключевые слова:*** *инженерная профессиональная психология, человеческие факторы лидеров коллективов, психологические характеристики специалистов, особенности психологии производственных групп, компоненты программных продуктов*

## 1. Основные понятия инженерной психологии при производстве сложных программных продуктов

Влияние человеческих факторов на профессиональную трудовую деятельность людей изучается ***инженерной профессиональной психологией*** [1, 6, 7]. Руководителю проекта при создании крупной системы необходимо ***уметь оценивать*** особенности психологии специалистов, с которыми предстоит работать и возможное их влияние на успешное выполнение проекта. Для этого ему полезно знать основы профессиональной психологии и связанные с ней методы эффективного управления человеческими ресурсами. Элементы и понятия этой науки важно использовать в процессе обучения специалистов на различных этапах производства ***сложных программных продуктов***. Фундаментом этих знаний являются общие психологические свойства личностей – субъектов производственных процессов.

***Человеческие факторы***, психологические характеристики специалистов, личная профессиональная квалификация, организация крупных коллективов ***определяют*** качество и трудоемкость при производстве сложных программных продуктов. Дефекты, трудоемкость и длительность отдельных операций и частных работ существенно зависят от индивидуальных, психологических особенностей их исполнителей, функционального назначения и характеристик компонентов и конкретного проекта. Даже при сокращении суммарных затрат на разработку программных компонентов за счет высокой автоматизации труда все более определяющей для производства становится доля затрат на творческий труд и, как следствие, ***возрастают требования*** к творческим профессиональным способностям конкретных специалистов. Отсюда принципиальной особенностью производства сложных комплексов программ является необходимость активного участия руководителей-менеджеров при ***отборе и подготовке профессиональных специалистов – создателей программных продуктов и их компонентов***. Такой отбор можно проводить на базе анализа характеристик прототипов завершенных специалистами разработок.

Анализ и учет человеческого фактора в программной инженерии связан с большими трудностями, характерными для новых разделов науки и техники, которые появляются ***на стыке различных профессий и областей знаний***. В данном случае особенность состоит в том, что ***менеджеры и разработчики*** комплексов программ, как правило, не знают даже основ психологии, необходимых для успешного производства сложной интеллектуальной продукции, а ***психологи производства*** не представляют сущность и свойства объектов производства – программных продуктов, а также особенностей соответствующих технологических процессов. Положение осложнено трудностями оценки и учета психологических характеристик и профессиональной квалификации руководителей и специалистов, участвующих в создании таких объектов. Технология регламен-

тированного проектирования и производства сложных программных продуктов большими коллективами специалистов с позиций ***психологии значительно отличается от индивидуальной разработки небольших программ*** [2, 5, 6]. Руководители должны уметь различать психологические характеристики и типы личностей ***в коллективе специалистов***, использовать их общие и профессиональные характеристики, стимулировать производственные особенности творческих личностей, осуществлять подбор специалистов с необходимым набором качеств.

В связи с расширением сфер применения и повышением уровня критичности функций, выполняемых программами, возросла необходимость ***гарантировать их высокое качество, потребность в*** корректном и регламентированном режиме формирования требований к характеристикам создаваемых комплексов программ. Вследствие этого ***повысилась психологическая и юридическая ответственность коллективов, конкретных руководителей и специалистов за качество*** создаваемых программных продуктов. Для гарантии качества и безопасности применения сложных программных продуктов руководителям и специалистам целесообразно знать и учитывать свойства дефектов и ошибок в комплексах программ, их типы и источники, факторы, влияющие на их проявление и обнаружение, а также возможные негативные последствия.

Для эффективного использования человеческого фактора ***необходимо психологическое воспитание и профессиональное обучение*** руководителей и специалистов различных категорий и квалификации методам и процедурам обеспечения и гарантии качества производственных процессов сложных программных продуктов. Для этого необходимо изучать различные тематические сферы и потребности в различных категориях специалистов по программной инженерии, реальные процессы и психологические аспекты деятельности, методы обучения, оценки профессиональной квалификации и роли в коллективах. Это требует непрерывного совершенствования обучения и повышения квалификации заказчиков, разработчиков и пользователей в области программной инженерии, освоения ими современных методов, процессов и международных стандартов, а также ***высокой корпоративной культуры работы коллективов специалистов***, обеспечивающих жизненный цикл сложных и особенно критических программных продуктов. Для этого руководителям и специалистам полезно знать основы современной психологии поведения людей в больших коллективах при промышленном производстве ***программных продуктов***.

Понятие профессиональной задачи ориентирует специалиста на будущее ее решение. При этом он получает задание от другого специалиста, стоящего выше в административной иерархии (или руководителя), в чем проявляется ***социально-психологический характер производственных отношений*** [1, 3]. Статус стоящего выше в административно-производственной иерархии специалиста позволяет давать задание, которое подчиненный ему специалист ***обязан*** выслушать, оценить возможность его выполнения и в случае положительной оценки подтвердить свое согласие и/или готовность выполнять задание. В случае несогласия подчиненный должен дать мотивированные предложения по корректировке задания для его обсуждения в соответствии с принятым административно-производственным регламентом. В задании указывается цель, средства, сроки и путь, показывающий как можно прийти к требуемому результату. Продумывание возможности и способов реализации, оценка сложности выполнения задания специалистом должны быть направлены на его ***решение***.

Содержание профессиональной деятельности составляют не только технологические, но и психологические компоненты. Квалифицированный специалист должен всегда осознавать и, как следствие, учитывать ***психологические аспекты трудовой деятельности***. Сложность стоящей перед специалистом задачи определяется тем, насколько быстро и легко удается ему освоить критические операции, необходимые для ее выполнения, а также временем решения задачи в целом. Для оценки сложности задачи решающее значение имеет ***опыт специалиста***. Факторы, определяющие сложность практической задачи: неопределенность, новизна, неожиданность событий и объектов окружающего мира, неполнота, неясность и неточность информации, поступающей к специалисту. Задачи становятся более сложными при временном дефиците и жестких сроках, отводимых на их решение. Для этого предполагается построение особых планов и стратегий, для которых характерны неполнота, неопределенность, незавершенность. Сложные задачи связаны с ***высокой ответственностью*** за возможные дефекты и, как следствие, социальным (или административным) давлением на исполнителя.

***Психологическое содержание динамики труда*** профессионалов ***характеризуется*** [6, 7]:

- сбором информации об объекте, системе и среде, анализом ситуации в целом, выделением вариантов и определением их последствий, координацией действий, контактами с различными специалистами, особенностями трудовой деятельности;
- особенностями психологического содержания труда специалистов, включающими скорость поступления и объемы информации, сложность решаемых задач, ответственность, нерегулярный характер появления проблемных ситуаций и скорости их развития, как правило, длительные периоды наблюдения и пассивного ожидания;
- факторами, определяющими трудности решения задачи на практике, в числе которых: неопределенность, новизна и неожиданный характер событий и объектов в среде окружения, неполнота, неточность информации, поступающей к специалисту, неопределенный характер факторов, влияющий на сроки завершения планов, стратегий и вариантов действий, временным дефицитом, социальным давлением на исполнителей;

• выполнением действий в процессе реализации задания, которые сопровождаются анализом ситуации в целом, выделением вариантов и определением их последствий, моментами выбора и исполнением отдельных действий;

• анализом последствий предпринятых действий, которые включают коллективный разбор результатов, разное видение произошедшего и различные интерпретации последовательности событий, осмысление сделанного и оценка последствий, трудности адекватного осознания и интерпретации полученных результатов, влияние коллектива на формирование стандартов выполнения операций у каждого отдельного специалиста.

Чтобы улучшить труд специалистов, необходимо выйти за рамки непосредственных трудовых ситуаций и понаблюдать их во время обучения, например, на групповых разборах, при обмене опытом, во время тренажерной подготовки, где при отработке уникальных, критических или аварийных ситуаций обнаруживаются скрытые особенности человеческого характера. Рассмотрение труда специалистов как творческого процесса должно быть направлено на разрешение психологических вопросов, связанных с неудовлетворенностью отношениями в коллективе и возникающими по этой причине конфликтами, а также должны учитываться общие основы современной инженерной профессиональной психологии, включающие:

– понятие профессиональной задачи специалиста и ее социальный психологический характер;

– способы управления поведением специалистов для предотвращения и выявления человеческих ошибок;

– психологические аспекты динамики труда профессионалов в области создания сложных систем;

– то обстоятельство, что труд может приносить большее удовлетворение специалистам, у которых высокий уровень интеллектуальной активности, глубокие и разносторонние знания и большой опыт работы.

## 2. Психологические особенности лидеров коллективов создателей программных продуктов

Для того чтобы сотрудник коллектива мог эффективно решать поставленную задачу, необходимо: понимание цели работы; умение ее выполнять; возможность и желание ее реализовать. Для этого руководитель-лидер должен уметь эффективно выполнять свои функции и решать возникающие в связи с этим задачи. К их числу относятся: обеспечение общего понимания сотрудниками целей и стратегий их достижения; возможность быть наставником и образцом для подражания, помогать сотрудникам в работе, обеспечивать их всем необходимым для успешной работы; обеспечение адекватной мотивацией участников коллектива (далее используется термин производственного коллектива) на протяжении всего времени выполнения проекта. По ходу проектирования и производства программного продукта мотивы людей, как правило, изменяются и самооценка зачастую неадекватно завышается. Эффективные команды обычно концентрируются вокруг признанного лидера. Наиболее эффективные процессы производства программного продукта складываются в самоорганизующихся группах – рабочих командах. Для них характерны: ясность общих целей, самоконтроль, взаимопомощь, взаимозаменяемость, коллективная ответственность за результаты труда, всемерное развитие и использование индивидуального и группового потенциалов.

Руководителю группы (коллектива) разработки программного продукта недостаточно быть хорошим управленцем, он должен быть лидером, признанным этой группой. Для этого необходимо признание коллективом профессиональной компетентности и превосходства руководителя, как следствие – доверие коллектива к действиям и решениям руководителя. Обязательны признание его исключительных человеческих качеств, убежденность в его честности, порядочности, вера в его искренность и добросовестность. В зависимости от состава, квалификации и задач участников коллектива, при делегировании полномочий его лидеру могут рассматриваться различные исходные посылки (мотивы): руководителем в новый коллектив; в известный слаженный дружественный коллектив; в коллектив, настроенный критически, где каждый считает, что сам может быть руководителем. Во всех ситуациях лидер обязан последовательно организовать эффективно действующий производственный коллектив, для чего зачастую приходится пройти следующие этапы [1, 6, 7]:

• формирования коллектива, когда люди должны преодолеть внутренние противоречия, "переболеть конфликтами" прежде, чем сформируется действительно слаженный, объединенный общими идеями и установками на совместную деятельность коллектив;

• разногласий и конфликтов, на которых неизбежные сложности или неудачи порождают конфликты и "поиск виновных", когда участники команды методом проб и ошибок вырабатывают наиболее эффективные процессы взаимодействия;

• становления коллектива, когда растет доверие его членов друг к другу, люди начинают замечать в коллегах не только проблемные, но и сильные стороны, закрепляются и оттачиваются наиболее эффективные процессы взаимодействия;

• состояния, когда коллектив работает эффективно, высок его командный дух, люди хорошо знают друг друга и умеют использовать сильные стороны коллег (это лучший период для раскрытия индивидуальных талантов).

Лидер в эффективном коллективе должен целенаправленно стимулировать творческое инакомыслие и интеллектуальную конкуренцию. Ни одно предлагаемое участником команды решение не принимается на веру, возможные негативные последствия или упущенные возможности активно анализируются при принятии решения, конфликты носят исключительно производственный характер. Во внимание принимается проверенное практикой обстоятельство: все, что человек

делает с удовольствием, он, как правило, делает максимально эффективно. В результате ***достигается высокая эффективность*** производственного коллектива. Соответственно, специалисты должны овладеть максимально полной информацией о сути работ, выполняемых коллективом, и ролях, которые должны играть отдельные его члены. Они должны понимать важность таких сторон совместной деятельности, как дисциплина, необходимость придерживаться установленных сроков и разумно сочетать как индивидуальную производительность каждого, так и командную. Лидеру следует разрешать противоречия в интересах выполнения стоящих перед проектом целей, находя для этого приемлемые компромиссы в рамках существующих ограничений (стоимость, время, знания, существующие ограничения административного характера и др.). Специалисты должны выполнять задания, в том числе содержащие противоречивые и изменяющиеся требования.

Проектировать решения в предметных областях приходится, используя подходы программной инженерии, ***сочетающие и балансирующие этические, общественные, юридические и экономические интересы различных заинтересованных сторон***. Руководителю необходимо понимать достоинства и недостатки различных альтернатив и последствия выбора того или иного подхода в каждой конкретной ситуации. Для успешного выполнения крупного программного продукта и/или его значимой компоненты ***лидер группы должен обладать необходимым талантом и квалификацией. Как следствие, он должен уметь следующее***:

- руководить процессом выявления, конкретизации и формирования требований заказчика продукта;
- осуществлять проверку спецификаций требований к программному комплексу, чтобы удостовериться, что они соответствуют реальной концепции заказчика, представленной детальными функциями;
- квалифицированно вести переговоры с заказчиком, пользователями и разработчиками, определять и поддерживать равновесие между тем, чего хочет заказчик, и тем, что может создать коллектив разработчиков за ресурсы и время, выделенные заказчиком для реализации продукта;
- рассматривать конфликтующие пожелания, поступающие от различных участников проекта и находить компромиссы, необходимые для определения набора функций, представляющих наибольшую ценность для максимального числа участников и проекта в целом.

Для крупных комплексов программ целесообразно ***формировать иерархию лидеров-руководителей***, адекватно отражающих структуру и взаимодействие базовых функций и компонентов программного продукта [2, 4]. Основному руководителю всего проекта должны быть подчинены руководители отдельных групп, выполняющих его ключевые функции. Руководителей таких групп следует выбирать с учетом основных психологических свойств и характеристик лидеров соответствующего уровня ответственности. В едином процессе управления крупным проектом можно выделить несколько ***типовых взаимодействующих уровней руководителей***, обеспечивающих реализацию основных целей и политики производства, консолидирующих его корпоративные знания. На высшем уровне формируются общие цели, критерии их достижения, а также принципы и схемы взаимодействия с окружением. Формализуются общая причинно-следственная модель деятельности, цели, подлежащие измерению способы отображения уровня достижения целей. Ответственность за правильность и полноту представления данного уровня возлагается на руководителей высшего звена.

***Лидеры группы*** должны быть в курсе повседневной деятельности коллектива, гарантируя эффективную его работу и тесное сотрудничество с заказчиком проекта при планировании деятельности по его реализации. Лидер, как правило, назначаемая должность, он подотчетен главному менеджеру проекта. Такой лидер может и не быть лидером коллектива в прямом смысле этого слова. Он, в первую очередь, должен вести группу специалистов только в технических вопросах, относящихся к проекту. Если в группу назначается лидер, которого не признают члены команды, это может привести к достаточно напряженной обстановке. Такая напряженность весьма вероятна для такой быстро изменяющейся области производства, как программная инженерия, где некоторые члены коллектива могут владеть более современными знаниями и опытом, чем лидеры групп, имеющие только практический опыт. После того как группа будет сформирована, менеджер проекта должен перейти к осуществлению непосредственного руководства и продемонстрировать необходимое качество лидера группы.

***Менеджерам крупных проектов целесообразно учитывать психологические особенности лидеров производственных коллективов, в том числе:***

- реальный талант и квалификацию лидера для управления и решения производственных задач определенным коллективом;
- размеры и сложность задач, доступных для лидера, управляющего определенным коллективом, созданным для их решения;
- стиль обсуждения и принятия производственных решений в коллективе (демократический, либеральный или автократический);
- карьерную ориентацию каждого лидера при управлении доверенным ему коллективом на консолидацию и стабильность коллектива, на создание атмосферы независимости отдельных специалистов при действиях в интересах проекта;
- лидер-руководитель обязан уметь последовательно, поэтапно организовать эффективно действующий производственный коллектив.

Помимо навыков планирования, организации, руководства и контроля, которые необходимы для управления проектом, ***любой руководитель должен обладать рядом общих психологических свойств***. В том числе ему необходимо умение:

– оказывать влияние на эмоциональную реакцию других, на перемены в поведении их действий;

– вырабатывать в себе привычку к строгой самооценке, которая не должна зависеть от внешнего подтверждения правильности предпринятых действий;

– разрешать затруднения, с которыми сталкивается группа;

– доверять другим специалистам, контролировать свой стиль поведения и ставить интересы коллектива выше личных предпочтений;

– изучать самооценку других и выяснять их потребности в подтверждении правильности предпринятых действий;

– разрешать конфликты к взаимному согласию;

– убедительно доводить замысел, план или решение задачи до подчиненных;

– понимать первопричины мотивации других, умение поставить себя на место другого специалиста.

## 3. Психологические особенности специалистов производства программных продуктов

Трудовая деятельность отдельных специалистов, участвующих в коллективном производстве программного продукта, является целенаправленной и мотивированной. Под ***мотивацией*** в научном и практическом менеджменте понимается процесс побуждения себя и других к деятельности для достижения личных и общих для коллектива целей. Совпадение личных и общественных целей служит основой управления трудовой деятельностью коллектива на любом производственном и/или научном предприятии. Причина в том, что выбор и обоснование единой производственной или экономической цели является одной из главных функций современного менеджмента, задачей специалистов на всех уровнях управления [2, 4]. Мотивацию следует рассматривать с позиции заинтересованности отдельного члена коллектива соответствовать требованиям конкретной задачи или даже положительно отличиться в ходе ее решения.

Включаясь в трудовую деятельность, каждый специалист в составе коллектива интересуется не только общими целями и результатами работы, но и возможностью удовлетворить свои личные потребности. Каждого работника далеко не в последнюю очередь интересуют его ***личные цели и задачи, трудозатраты и результаты***. В том числе ему необходимо знать: что он должен делать и в каких условиях; какие физические и умственные усилия от него потребуются; какими свободами он должен жертвовать во имя общего дела; с какими людьми и как ему предстоит взаимодействовать; как будет оцениваться и вознаграждаться его труд. От этого и ряда других факторов зависит не только удовлетворенность человека собой, своим взаимодействием с другими членами коллектива, но и ***мотивация его личного отношения к работе***, величина трудового вклада в общие производственные и финансовые результаты деятельности предприятия. ***Сочетание личных и общественных мотивов деятельности*** человека в коллективе является одной из важнейших задач управления производством любого, в том числе и программного продукта.

***Системы стимулирования*** относятся к тем организационным аспектам, которые побуждают к действию, направляют или поддерживают поведение специалистов. К ним относятся поощрительные, административные и социальные системы стимулирования, направленные на эффективное решение задачи. Мотивационные свойства системы поощрительного стимулирования тесно связаны с ожиданием того, что дополнительные усилия будут вознаграждены повышением оплаты труда, а сама величина такой оплаты послужит дополнительным стимулом для конкретного лица. Система стимулирования решения задачи отражает способ повышения внутренней мотивации к делу, когда отдельное лицо выполняет работу, которая ему интересна или является для него вызовом. Система административного стимулирования побуждает к действию, направляет и/или поддерживает поведение сотрудника посредством целого ряда методов. В системе социального стимулирования эффективно мотивированные отдельные лица реагируют на нормы и санкции, применяемые руководителями коллектива, в составе которого он работает.

При общении в производственном коллективе или при проведении совещаний полезно признавать и учитывать индивидуальные отличия специалистов, что способствует продуктивности их делового взаимодействия. Для удовлетворения потребности в оценке сотрудников важно давать им понять ***роль*** в коллективе [2, 4]. Открытое признание их достижений – наиболее простой и эффективный способ удовлетворения этой потребности. Кроме того, отдельные члены коллектива должны чувствовать, что их работа оплачивается на должном уровне, который определяется их знаниями и опытом. Чтобы удовлетворить их потребности в самореализации, важно предоставить каждому сотруднику определенный ***уровень ответственности за сделанную работу***.

В зависимости от предметной области и затрат на решение задачи в группу исполнителей могут быть ***приглашены профессионалы высокой квалификации*** на временной или постоянной основе. Это может быть высококвалифицированный программист, администратор, специалист по инструментальным средствам разработки программ, специалист по операционным системам или языкам программирования, специалист по тестированию программных продуктов или отдельных компонентов системы. Уровень производительности труда (по условной шкале от более к менее квалифицированным программистам) может разниться в 25 раз. По этой причине нужно с наибольшей эффективностью использовать возможности лучших в профессиональном отношении специалистов, обеспечив им необходимую поддержку.

Однако следует учитывать, что при приглашении высококвалифицированных специалистов в коллектив могут возникать осложнения психологического характера [2]. Например, талантливые разработчики компо-

нентов, программисты, тестировщики встречаются нечасто, в то время как организация группы может быть основана на таком компетентном ведущем программисте или тестировщике в качестве ее руководителя. Если он совершает ошибки, то обсудить его решения, как правило, не с кем. В группе, основанной на демократичных отношениях, каждый специалист может обсудить решение и таким образом обнаружить ошибки и избежать их. К осложнениям в коллективе может привести ситуация, когда ведущий программист (тестировщик) оказывается ответственен за полное выполнение выделенного компонента проекта и, как следствие, он может ***взять на себя заслуги в случае успешного выполнения задания***. Однако члены группы могут не согласиться, если их роль в проекте не будет признана в достаточной мере. В таком случае будут не удовлетворены их потребности в оценке персональных заслуг.

***Возможен срыв сроков сдачи продукта заказчику*** в случае болезни или увольнения ведущего профессионала. Как следствие, руководители коллективов, выполняющие сложные проекты, предвидя такое развитие событий, могут не пойти на риск. Организационная структура предприятия–исполнителя проекта может оказаться не способной обеспечить подобный тип коллектива. В этой связи ***структура группы с ведущим специалистом*** может оказаться для формирующего ее предприятия весьма ***рискованной***. Однако, как показывает практика, необходимо поддерживать талантливых профессионалов, выделяя при этом их помощников, дублеров.

***При подготовке и выборе профессионалов высокой квалификации целесообразно учитывать общие психологические факторы***:

- психологические причины мотивации деятельности личностей, включая внутренние – личные цели и заинтересованность, и/или внешние – материальные и моральные интересы;
- систему, факторы и степень персонального стимулирования, в том числе административное воздействие, степень удовлетворенности, значимость, справедливость, ожидание оценок или поощрения;
- психологическую ориентацию и формирования карьеры личностей;
- профессиональные принципы и коллективную дисциплину, ориентированные на решение крупных проблем;
- хорошие коммуникативные навыки, способность критически оценивать конкурирующие решения.

## 4. Психологические особенности коллективов специалистов

Основная часть сложных программных продуктов разрабатывается большими коллективами специалистов (до нескольких сотен человек). Поскольку сложно организовать эффективную работу над одной крупной задачей в большой команде, ее, как правило, делят на группы. Каждая группа отвечает за создание определенной части комплекса программ и обычно работает над одной функциональной задачей или компактной подсистемой. При грамотном подборе группа специалистов обычно состоит не более чем из восьми человек. В группах небольшого размера легче снизить риск появления сложностей во взаимоотношениях между ее членами. Для таких групп не возникает необходимости применять сложные средства коммуникаций, и для эффективного взаимодействия их члены должны обладать следующими качествами: независимостью, готовностью к совместной работе, работоспособностью, умеренностью в отношении соперничества и толерантностью. Создание групп представляет собой процесс планируемого и хорошо продуманного стимулирования к эффективному труду при одновременном сведении к минимуму трудностей и препятствий, мешающих проявлению профессиональных навыков и изобретательности отдельных членов группы.

***Эффективность результатов коллективной работы группы*** определяют следующие основные факторы:

- группа должна иметь рациональное соотношение профессиональных навыков, опыта и личностных качеств отдельных ее членов;
- члены группы должны воспринимать себя как единую команду, а не как простую совокупность индивидуумов, работающих над одной задачей;
- между членами группы должны быть дружеские отношения;
- необходимо организовать группу таким образом, чтобы каждый чувствовал важность своей роли и был ею удовлетворен в общих результатах.

Необходимо, чтобы в группе было сбалансированное соотношение технических навыков, опыта и степени выражения индивидуальности. В такой команде будет присутствовать дух товарищества, который мотивирует отдельных сотрудников к успеху всей команды для достижения собственных целей. По этой причине менеджеры должны стимулировать деятельность, направленную непосредственно на ***"строительство команды"***, чтобы содействовать формированию чувства преданности ее интересам, которые в той или иной степени влияют на групповую работу. Наилучший способ воспитать дух команды – дать возможность каждому почувствовать, что он несет ***определенную долю ответственности*** и ему доверяют, а также гарантировать доступ к проектной информации всех членов группы.

***Формирование группы – команды***, которая могла бы эффективно работать над отдельным компонентом или функцией комплекса программ, является достаточно сложным процессом. Необходимо обучить подчиненных, поощрять их профессиональную работу во взаимодействии и по графику [2, 4]. Для обеспечения приверженности подчиненных выполняемой ими работе следует привлекать специалистов, цели которых совпадают с целями предприятия–исполнителя проекта, а также высоко ценящих предлагаемую им работу. Ориентироваться следует на специалистов, способных реально оценить работу в целом и затраты на ее

выполнение, а не на цели и стремления каждого члена группы в отношении его личной карьеры.

Для создания ***благоприятной рабочей обстановки в коллективе*** целесообразно давать отдельным его членам разнообразные, интересные для них задачи, обеспечивать социальные гарантии, подбирать сотрудников, с которыми приятно работать, добиваться положительных откликов на их работу со стороны заказчиков. Необходимо также вовлекать сотрудников в процесс принятия решений и поощрять хорошую работу продвижением по службе и предоставлением особых прав, отдельных рабочих мест, дополнительных рабочих помещений и вспомогательного персонала. Для поддержания производительности труда на должном уровне рекомендуется разработать удобную программу оценки и контроля сроков и качества выполненных работ, а также обучения персонала. Следует создать у подчиненных уверенность в том, что администрация готова решать их проблемы, справедливо распределять премиальный фонд, способствовать профессиональному росту сотрудников, продвигать новые идеи, сохранять лояльность к работникам, предоставлять возможности для роста кадров. Если менеджеры проекта сделают все возможное для достижения перечисленных целей, сотрудники в ответ проявят ***ответственность за результаты труда***, будут уверены в его значимости, станут преданными коллективу и предприятию–исполнителю проекта, используют свои возможности и проникнутся чувством собственной значимости.

Хорошо организованный и сплоченный коллектив исполнителей (команда) имеет определенные ***преимущества, а именно***:

- возможность установления стиля, стандарта работы и взаимодействий членов команды, определяемые всей группой единогласно, которые легче контролировать, чем чужие стандарты, навязываемые извне;
- члены команды поддерживают тесные рабочие контакты, люди учатся друг у друга, уменьшается вероятность затягивания сроков работы, вызванных незнанием отдельных факторов, влияющих на ее выполнение, или неосведомленностью;
- члены команды ознакомлены с деятельностью друг друга, чем достигается возможность продолжения работы даже после ухода из команды одного из ее членов;
- возможно внедрение в практику группы безличного программирования, когда программный продукт должен быть собственностью всей команды, а не отдельной личности.

***Безличное программирование*** [2, 4] означает определенный стиль работы, при котором продукты, компоненты программы и документация считаются собственностью всей команды, а не отдельного человека, который занимался их разработкой. Если разработчики психологически воспринимают свою работу именно таким образом, то они охотнее отдают ее на проверку другим членам группы, легче воспринимают критику, стремятся работать над усовершенствованием компонентов. При этом группа разработчиков становится более сплоченной, так как каждый чувствует себя в ответе за разработку всей системы. Кроме того, что безличное программирование улучшает качество системной архитектуры, программ и документации, оно также совершенствует взаимоотношения внутри группы, стимулирует непринужденное обсуждение задач, независимо от социального положения, опыта и пола сотрудников.

Для группы по разработке программных продуктов необходим ***развитый коммуникационный фактор***, общение и хорошие средства связи между членами группы. Работники должны информировать друг друга о том, как идет их работа, о решениях, которые предпринимались в отношении проекта, и тех необходимых изменениях, которые вносились в предыдущие версии. Группа должна обсуждать предстоящую работу со всеми членами коллектива, а задания назначаться в соответствии с возможностями и опытом конкретных сотрудников. Техническим лидером команды может стать сотрудник, который способен лучше управлять процессом разработки.

## 5. Особенности принятия решений в коллективах специалистов

***Групповое принятие решений*** необходимо для сплочения группы, особенно в том случае, если она состоит из подгрупп, специализирующихся на выполнении конкретных функций. По мере увеличения группы повышается вероятность появления большего числа персонально заинтересованных сторон, причем их интересы могут быть учтены в критериях, применяемых в процессе принятия решений. Вовлечение отдельных лиц с особым практическим опытом в процесс принятия решений позволяет повысить вероятность того, что в данном процессе будут использованы более точные причинно-следственные допущения.

Важным для руководителя группы является выбор собственного ***стиля принятия решений***, который может изменяться в зависимости от конкретных обстоятельств – от автократического до участия в этом процессе всей группы. При автократическом или направляющем стиле решения проблем менеджер определяет и тщательно анализирует подлежащую решению задачу, а затем формирует, оценивает и выбирает наиболее подходящее решение. Кроме того, при таком подходе команда может получить вводную информацию в виде определяемой руководителем стратегии решения задачи. После этого он использует группу в качестве источника информации для получения сведений, позволяющих определить причину появления именно этой стратегии.

Если руководитель склоняется к некоторой форме группового принятия решений, в этом случае важно назначить какого-либо участника группы, который будет руководить или способствовать упрощению данного процесса. Этот специалист должен взять на себя ответственность направлять группу на протяжении всего процесса принятия решений. ***Когда решения при-***

***нимаются в группе***, участники стремятся прийти к варианту, который удовлетворял бы каждого члена группы. Если группа действует успешно, отдельный участник видит в этом некоторую личную выгоду, реализуемую, в частности, посредством премирования за достижение групповых целей. Этот успех может являться подтверждением компетентности и отдельного лица. Для этого необходимо, чтобы отдельный участник понимал, что выполняемая им роль оказывает существенное влияние на успешную деятельность всей группы.

***Конфликты являются обычным фактом жизни любого*** трудового коллектива, а в первую очередь – интеллектуально сильного коллектива разработчиков программного продукта. Поскольку руководителям проектов создания программных продуктов неизменно приходится иметь дело с конфликтами, они должны научиться их распознавать и предотвращать самыми разными способами, способствовать их прекращению и уметь вести переговоры. Каждая сторона, участвующая в конфликте, несет ответственность за ясное изложение своих целей и выяснение у другой стороны ее целей. Конфликты существенно отличаются от нормального режима общения членов коллектива своей интенсивностью и внешними проявлениями. Они могут мешать общению специалистов, препятствовать ему, изменять его стиль, оказывать влияние на то, как воспринимается вводная информация, а также ограничивать и искажать ее содержание. Причины конфликтов коренятся в следующих ***двух типах разногласий***:

- критерии, интересы и цели;
- причинно-следственные убеждения, теории и допущения.

Если существует конфликт по поводу критериев, интересов и целей, то он может быть разрешен в процессе принятия решений. ***Эмоциональный конфликт*** по поводу убеждений, теорий и допущений может продолжаться довольно долго. Это обстоятельство приводит к тому, что другая сторона начинает в своих действиях руководствоваться личными эмоциями, которые, зачастую, еще долго негативно сказываются на ходе выполнения проекта даже после окончательного их разрешения.

***Методы урегулирования руководителями конфликтов*** требуют [1]:

– затрачивать дополнительное время и энергию для получения взаимовыгодных результатов;

– энергично приступать к решению задачи, однако не прибегать к вопросам персонального характера;

– внимательно относиться к мнению другой стороны, а именно слушать, понимать, признавать, ценить и предпринимать ответные действия;

– стремиться к прямому, открытому и доверительному общению;

– вмешиваться только по мере необходимости во время обсуждения проблемы с третьей стороной, в том числе слушать, наставлять или советовать, упрощать процесс обсуждения, договариваться, выступать в качестве посредника;

– толерантно относиться к наличию конфликтов, считая при этом, что конфликт может быть полезным, важным, ценным и выгодным (в большинстве случаев скрывать конфликт нецелесообразно).

Для каждого коллектива, предназначенного для проектирования и производства ***сложных программных продуктов***, характерны определенные процессы развития, которые целесообразно учитывать менеджерам. Большое значение при этом имеют ***психологические характеристики специалистов***, их квалификация и зрелость всего коллектива. Следует стремиться к ***сохранению стабильности коллектива***. Этими процессами надо управлять методами, к которым относятся:

- регулирование процессов перемещения, продвижения и понижения работников;
- оценка результатов работ и специалистов;
- профессиональная подготовка и повышение квалификации специалистов;
- поддержание нормального социально-психологического климата в коллективе;
- управление мотивацией и стимулированием специалистов;
- выбор оптимальной системы материального и морального стимулирования специалистов.

Большую роль в деле ***стабилизации*** коллектива играет ***социально-психологический климат***. Он зависит от среды и уровня развития коллектива, непосредственно влияет на эффективность деятельности его членов, на осуществление основных его функций. Благоприятным является климат такого коллектива, отношения в котором отвечают ***задачам его развития***. Как следствие, при таком подходе у членов коллектива достаточно развита потребность в труде как сфере актуализации личности. В межличностных отношениях развиты взаимное доверие и уважение друг к другу, взаимовыручка и взаимная ответственность. Руководитель обязан обеспечивать формирование благоприятного климата, используя методы социального управления, преодоление конфликтов и развитие сплоченности коллектива, включая его комплектование с учетом психологической совместимости отдельных его членов.

***Принятие производственных решений в коллективах*** зачастую рассматривается как единый процесс, реализуемый посредством ***совещаний*** [1, 7]. Такие совещания обычно включают: обсуждение, исследование вопроса, аргументацию, конфликт, сравнение, компромисс. У каждого совещания должна быть ***повестка дня***. ***Председателя совещания*** или направляющего данный процесс желательно выбирать из числа незаинтересованных специалистов. Он направляет совещание в нужное русло, руководит всем процессом, делает содержание совещания доступным каждому его участнику, соблюдает его повестку, подводит итог обсуждению и следит за регламентом. ***Участниками совещания*** могут быть специалисты, обладающие необходимым опытом и квалификацией для внесения предложений, выражения идей, предложения вариантов, высказывания догадок, проявления активности и опытности. Они работают на

заказчика или руководителя проекта, как правило, являются главным кадровым резервом предприятия–исполнителя проекта и выбираются с учетом их возможного вклада в решение рассматриваемых вопросов. Они играют активную и ответственную роль на тех совещаниях, где происходит обмен информацией, решаются вопросы, принимаются оперативные решения, составляются планы или делаются оценки результатов проектирования и производства программного продукта.

***Умение вести переговоры*** относится к необходимым, важным качествам любого руководителя. ***Переговоры*** – это процесс вывода отдельных людей или групп из затруднительного или конфликтного положения и создание ситуации, которая способствует выработке согласованного решения. При этом две или более сторон с общими или противоположными интересами испытывают потребность или имеют поручение добиться согласия относительно разделения, распределения ресурсов или устранения технических разногласий. К методам ведения переговоров относятся планирование, проведение и достижение согласия, критика и принятие последующих мер. Благодаря внесению прямых и ответных предложений, обсуждения и уступок, обе стороны достигают взаимопонимания относительно наилучшего результата, которого им удается добиться.

Во всякой иерархии каждый сотрудник коллектива имеет тенденцию достигать ***своего уровня некомпетентности***. Если человек успешно справляется со своими обязанностями, его считают подходящей кандидатурой для повышения его статуса в коллективе. После ряда выдвижений он вполне может достигать того статуса, на котором обнаруживается его недостаточная компетентность, когда его новые обязанности оказываются ему не по силам. В этом случае дальнейшее повышение оказывается нецелесообразным. Такая ситуация вполне реальна, однако ее последствия несравнимы с отсутствием должного уровня компетентности у претендентов на руководящую должность в коллективе. Этот процесс приводит к тому, что многие, особенно ***руководящие должности могут быть заняты некомпетентными людьми***, долго остающимися на своих постах. Целесообразно отказываться от всякого повышения в должности, пока специалист еще находится в статусе, соответствующем уровню его компетентности.

***Культура взаимодействия в коллективе*** представляет собой характерную совокупность знаний и убеждений людей, связанных общим делом и ценностями, которыми они руководствуются в своей жизни. К элементам культуры относятся образование и убеждения, социальная организация и политическое воспитание в широком понимании, манера общения. При анализе и организации производства программного продукта необходимо рассматривать два комплекса вопросов. Первый из них в основном охватывает социальные проблемы, второй – технические и оба совместно – экономические. ***Социальное поведение*** отдельного лица в коллективе тесно связано с его особенностями. Необходимо, чтобы отдельный участник коллектива понимал, что выполняемая им роль оказывает существенное влияние на успешную деятельность всей группы. При этом ***целесообразно учитывать психологические особенности команд – коллективов профессиональных специалистов***.

## СПИСОК ЛИТЕРАТУРЫ


1. **Армстронг М.** Стратегическое управление человеческими ресурсами. Пер. с англ. М.: ИНФРА-М, 2002.

2. **Боэм Б.У.** Инженерное проектирование программного обеспечения. Пер. с англ. / под ред. А.А. Красилова. М.: Радио и связь, 1985.

3. **Дмитриева М.А., Крылов А.А.** Психология труда и инженерная психология. М.: Дельфа, 2005.

4. **Липаев В.В.** Человеческие факторы в программной инженерии: рекомендации и требования к профессиональной квалификации специалистов. Учебник. М.: СИНТЕГ, 2009.

5. **Организация** производства и управления предприятием. Учебник. Под ред. О.Г. Туровца. М.: ИНФРА-М, 2006.

6. **Стрелков Ю. К.** Инженерная и профессиональная психология. М.: Академия, 2001.

7. **Торингтон Д., Холл Л., Темлер С.** Управление человеческими ресурсами. Учебник. Пер. с англ. М.: Дело и сервис, 2004.

УДК 004.054

**В.Е. Гвоздев,** д-р техн. наук, проф., зав. каф.,
**Б.Г. Ильясов,** член-корр. АН Республики Башкортостан,
д-р техн. наук, проф., зав. каф.,
Уфимский государственный авиационный технический университет

E-mail: wega55@mail.ru

# Пирамида программного проекта

*Предлагается системная модель программного проекта, обобщающая известные системные модели – "треугольники проектов" и создающая методологическую основу построения иерархической системы моделей – структурных, когнитивных, математических. В качестве примера использования системной модели формируются модели внешней и внутренней среды программного проекта, когнитивная модель программного проекта.*

***Ключевые слова:*** *программный проект, программный продукт, пирамида программного проекта, треугольник проекта, внешняя и внутренняя среда, когнитивная модель*

## Введение

В многочисленных литературных источниках, посвященных управлению программными проектами, подчеркивается важность построения моделей программных проектов и программных продуктов как одного из основных инструментов поддержки принятия решений. Наличие целостной модели программного проекта как платформы взаимодействия правообладателей и разработчиков является необходимым условием сбалансированного устойчивого развития программного продукта.

Программные проекты обладают всеми признаками сложной системы. Для них характерны:

- иерархичность организации, позволяющая соподчинять друг другу различные объекты, входящие в состав проекта;
- типизация элементов и подходов к построению отдельных программных модулей и баз данных и уникальность программной системы в целом;
- слабая переносимость опыта, накопленного разными исполнителями при разработке программных продуктов, в том числе аналогичного назначения;
- недостаточная изученность протекающих в системах процессов; слабая структурированность теоретических и фактических знаний о системе (экспертных заключений), что часто является причиной контринтуитивного хода программных проектов;
- сложность реализации удовлетворительного прогноза хода программного проекта на достаточно большом промежутке времени;
- полиморфизм, означающий, что одному проекту в зависимости от целей исследования может быть поставлено в соответствие множество системных моделей (функциональная, информационная, событийная, объектная, динамическая и т.д.);
- неоднозначность подходов к декомпозиции проекта (даже в рамках одного из направлений исследования), гетерогенность элементов, входящих в состав проекта (это, в частности, приводит к необходимости построения ансамблей моделей для описания проектов);
- неоднозначность подходов к построению, развитию и обеспечению функционирования программной системы (наличие эквивалентных путей достижения цели);
- случайность и неопределенность факторов, воздействующих на проект (нерасчетные изменения окружающей среды, отказы оборудования, разрушения баз данных, неожиданные изменения организационной структуры и т.п.);
- омнипотентность факторов, выражающаяся, например, в том, что переход на следующую фазу жизненного цикла меняет цели управления и, соответственно, фазовое пространство проекта;
- многокритериальность оценок процессов, связанных с реализацией проектов. Это диктуется следующи-

ми обстоятельствами: наличием множества подсистем, каждая из которых оценивается своими критериями; множественностью показателей (иногда противоречивых), характеризующих результаты функционирования всей системы; наличием неформальных критериев, используемых при принятии решений (например, в случае, когда решения основаны на практическом опыте лиц, принимающих решения) и т.д.

В настоящее время разработано достаточно много моделей, предназначенных для информационной поддержки стратегического и операционного уровней управления программными проектами и программными продуктами разной сложности. В качестве примеров упомянутых моделей отметим следующие:

- трехуровневая архитектура программной системы [3];
- модели жизненного цикла (*code-and-fix*; "водопад"; "спираль", V-модель; инкрементальная и др.);
- модели для описания внешнего облика и поведения программного продукта (вербальная; диаграммы *Use Case*; модели, построенные на основе SADT-методологии и др.);
- структурные (блок-схемы алгоритмов; операторные модели; матрицы проекций (*traceability matrix*) и др.) и т.д.

Тем не менее, проблема разработки формальных моделей программных проектов решена далеко не полностью. Это обусловлено тем, что основное внимание, на наш взгляд, до сих пор уделялось построению моделей в рамках функционального аспекта. В то же время реализация главного принципа управления сложными системами – принципа комплексности – требует многоаспектного подхода к построению моделей программных проектов и программных продуктов, с разных позиций описывающих программные системы и процессы управления ими.

Для обеспечения логической взаимосвязи приводимых ниже материалов дадим следующее определение.

*Программный проект* есть развернутый во времени программный продукт. Различные фазы жизненного цикла программного проекта представляют собой последовательное преобразование ожиданий правообладателей в машинные коды.

Настоящая статья посвящена описанию системной модели программного проекта, с одной стороны включающей в себя в качестве частных случаев описанные в литературе системные модели проектов, с другой стороны, создающая методическую основу для построения системных, структурных и математических моделей, в разных аспектах описывающих программный проект.

## 1. Обоснование выбора точек зрения на программный проект

Методологическим принципом, положенным в основу разработки предлагаемой системной модели является принцип комплексности: адекватный выбор базовых ценностей и стратегических целей; эффективная архитектура ответственности; сбалансированность целей, ресурсов, механизмов управления и синхронизации их изменения. Этот принцип обосновывает выбор совокупности точек зрения на программный проект (рис. 1).


**Рис. 1. Точки зрения на программный проект**

Целесообразность системного моделирования программных проектов в рамках выделенных точек зрения обусловлена следующими соображениями.

**Функциональная точка зрения.** Совокупность задач, которые могут решаться посредством программного продукта, является одной из основных составляющих его ценности в глазах пользователя.

**Информационная точка зрения, знания.** Важным фактором, определяющим качество программного продукта, является качество информационного пространства программного проекта. Информационная система обеспечения жизненного цикла проекта представляет собой комплекс формальных и неформальных каналов обмена информацией между участниками проекта, сбора, систематизации и организации хранения удачных и неудачных решений. Влияние информационной поддержки управления программным проектом возрастает по мере роста сложности проекта.

В известной литературе подчеркивается, что одной из главных причин, препятствующих скоординированной деятельности разработчиков программного продукта, является невозможность получения ими в любой момент времени актуальной информации о состоянии проекта. Способность организации создавать, использовать и воспроизводить знания, генерировать новые идеи, приводящие к созданию нового качества и новых продуктов, в настоящее время является единственным, устойчивым конкурентным преимуществом [15].

**Управленческая точка зрения.** Современная точка зрения на управление сложными системами основана на формировании "умных систем управления", обеспечивающих получение конкурентоспособных про-

дуктов и услуг. В работах [4, 9, 10, 11 и др.] подчеркивается исключительная важность обеспечения соответствия уровней развития систем управления проектами сложности проектов.

**Ресурсная точка зрения.** Необходимым условием успешного выполнения программного проекта является наличие в нужное время в нужном месте требуемых ресурсов в достаточном объеме.

**Точка зрения правообладателей.** Человеческий фактор является системообразующим при создании программного продукта и оценивании качества получаемого результата. Это обусловлено следующим. Во-первых, причиной инициации программного проекта являются субъективные желания заказчика и спонсоров. Во-вторых, требования к потребительским свойствам программного продукта и критерии оценивания его качества также имеют личностную природу. В-третьих, подходы к реализации программного продукта (в том числе подходы к организации информационного пространства проекта) в значительной степени зависят от субъективных предпочтений и пристрастий разработчиков. Например, существует понятие "родовые знания" (*tribal knowledge*), означающее, что группа разработчиков склонна использовать те инструменты и приемы разработки, которые им хорошо знакомы. В-четвертых, состав и объемы ресурсов, а также подходы к их распределению и расходованию также определяются человеческим фактором.

## 2. Пирамида программного проекта

Выделенные точки зрения на программный проект и системообразующий характер человеческого фактора являются основой построения системной модели в виде "пирамиды программного проекта", представленной на рис. 2. Название модели дано по аналогии с известным "треугольником проекта" (*project triangle*). Отметим, что в этой системной модели вершины и ребра представляют собой многокомпонентные конструкции, описываемые множеством атрибутов. Состав атрибутов зависит, во-первых, от фазы проекта, во-вторых, от назначения программного продукта.

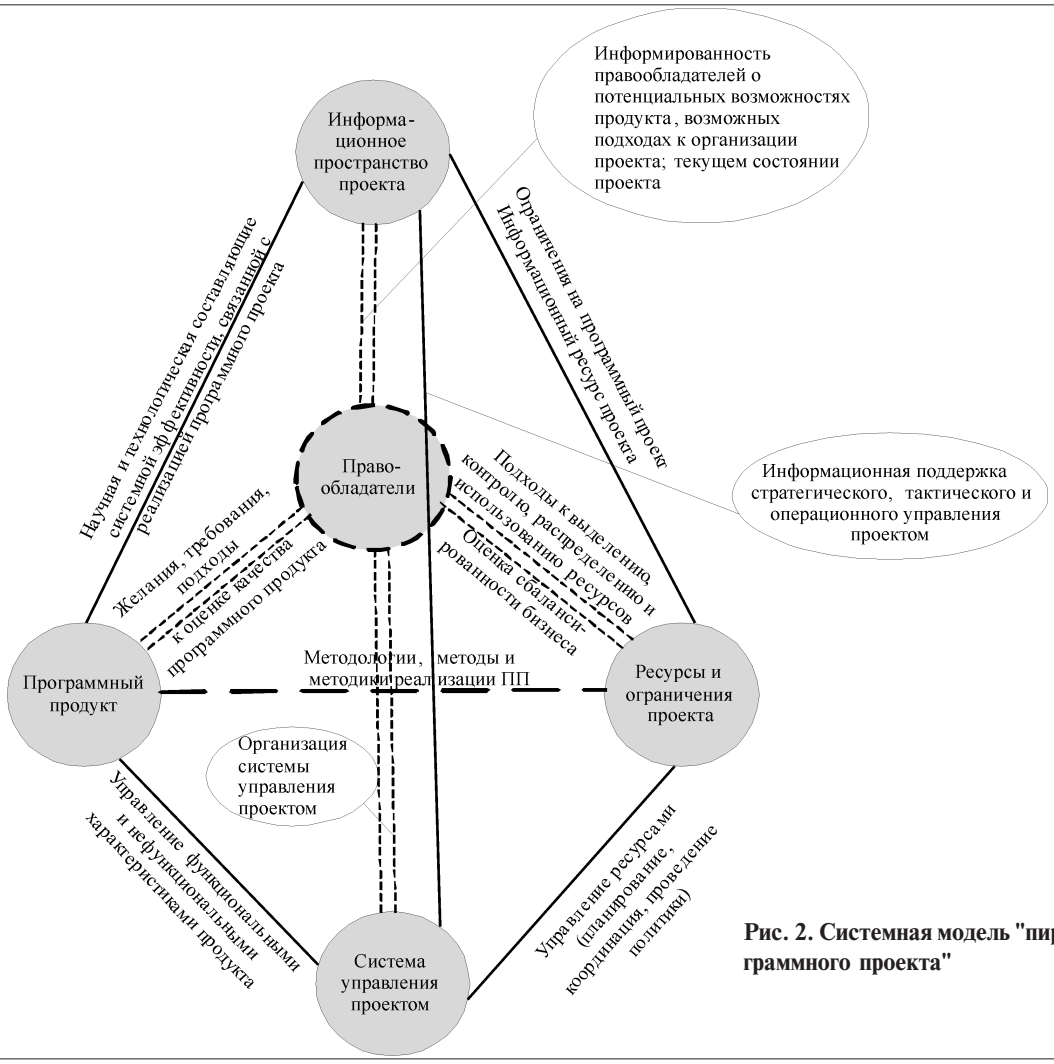


**Рис. 2. Системная модель "пирамида программного проекта"**

Ниже в качестве иллюстрации приведены примеры атрибутов для узлов предлагаемой модели.

***Программный продукт:***

- варианты использования;
- функциональные и нефункциональные требования;
- проектные и конструкторские решения;
- операционные характеристики, планы тестирования, верификации, валидации;
- результаты тестирования, верификации, валидации и т.д.

***Система управления проектом:***

- архитектура и структура административно-профессиональной подсистемы, регламентирующая распределение ролей, правила и способы коммуникации правообладателей и разработчиков;
- архитектура и структура общественно-профессиональной подсистемы (конференции, семинары и т.д.);
- законодательная и нормативно-правовая база;
- модель жизненного цикла программного проекта;
- модель жизненного цикла программного продукта и т.д.

***Ресурсы проекта:***

- бюджет;
- сроки реализации;
- информационные ресурсы;
- инструментальные ресурсы;
- жизненные ценности правообладателей;
- интеллектуальные ресурсы;
- кадровые ресурсы;
- характеристики рабочего пространства проекта и т.д.

***Информационное пространство проекта:***

- репозитории, содержащие информацию о реализованных проектах (например, отчеты *Standish Group*);
- описание полезных практик (например, PMBOK [9], SWEBOK [10]);
- литературные источники;
- системы обмена информацией между правообладателями и т.д.

***Правообладатели:***

- правообладатели, выражающие социально-экономические требования к программному продукту;
- правообладатели, формирующие внешнюю и внутреннюю среду программного продукта для разных условий: экстремальных (инициация проекта, реорганизация продукта, в том числе исключение из эксплуатации и демонтаж и т.д.) и штатных (режим реализации продукта, его модернизация и развитие);
- правообладатели, оценивающие результативность проекта и качество программного продукта;
- структура и роли правообладателей;
- квалификационные характеристики и компетентность правообладателей;
- мотивации правообладателей и т.д.

Выбор атрибутов имеет большое значение с точки зрения создания эффективного управления программным проектом, и эту задачу следует выделить в качестве самостоятельного направления исследования.

## 3. Свойства системной модели

Предлагаемая "пирамида проекта" в качестве частных случаев включает в себя описанные в литературе "треугольники проектов" ("железные треугольники") и архитектуры проектов. Ниже для компактности представления в "пирамиде проекта" использованы следующие сокращения: **ПП** – программный продукт; **СУ** – система управления проектом; **ИП** – информационное пространство проекта; **Р** – ресурсы проекта; **ПР** – правообладатели.

На рис. 3 приведены описанные в литературе варианты "железных треугольников" и архитектур, являющихся частными случаями предлагаемой системной модели – "пирамиды проекта".

## 4. Системная модель среды программного проекта

Как отмечалось во введении, предложенная "пирамида проекта" создает методическую основу для построения системных, структурных и математических моделей, в разных аспектах описывающих программные проекты и продукты. Например, приняв в качестве фокуса вершину "программный продукт", получим модель внешней и внутренних сред программного проекта.

Важность исследования внешних и внутренних сред неоднократно подчеркивалась в известных литературных источниках. Так, в работе [9] отмечается, что одним из факторов, определяющим требования к характеристикам программного продукта, является среда, в которой он будет использоваться. В работе [4] подчеркивается, что среда разработки программного продукта является определяющим фактором при выборе методологии разработки программного продукта. Там же отмечается, что качество информационной составляющей инфраструктуры программного проекта предопределяет не только успех проекта, но и эксплуатационные характеристики программного продукта. Список литературных источников, в которых обсуждается этот вопрос, может быть продолжен.

Приведенные примеры не представляют всего спектра точек зрения на внешнюю и внутреннюю среды. Их упоминание имело цель подчеркнуть необходимость системного моделирования внешних и внутренних сред программного проекта и программного продукта.

Выбрав в качестве модели жизненного цикла программного проекта V-модель и приняв в качестве фокуса вершину "программный продукт", получим модель (рис. 4), характеризующую внешнюю и внутреннюю среды программного проекта.

Содержание связей компонентов внутри блока "внутренняя среда программного проекта" указано на рис. 2. Каждая из компонент представленной архитектуры в свою очередь является сложной конструкцией. Учитывая диалектическое единство внешней и внутренней сред программного проекта, их взаимосвязи

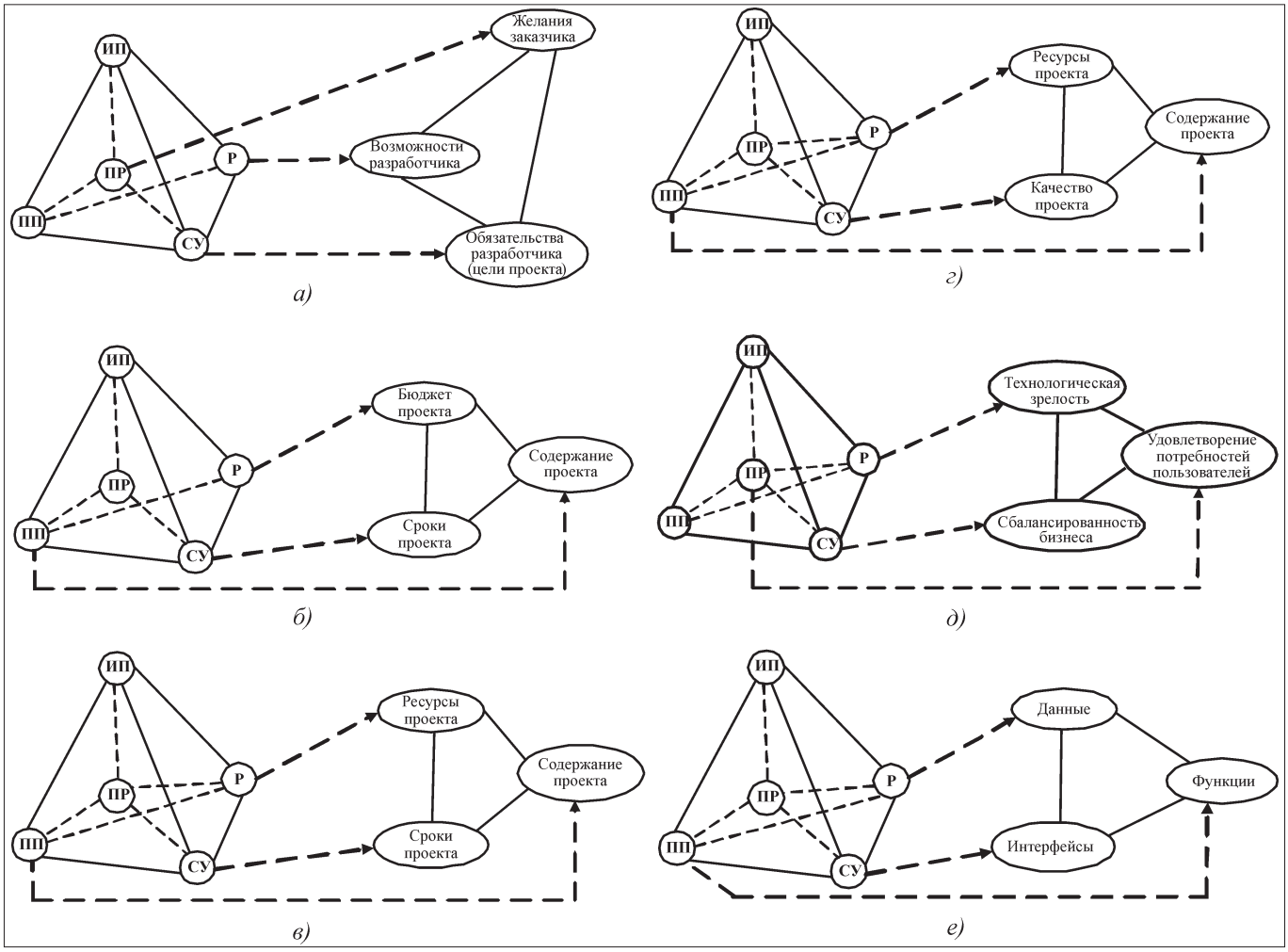


**Рис. 3. "Пирамида проекта" как обобщение описанных в литературе системных моделей программных проектов:**
*а* – пример архитектуры, описанной в работе [1]; *б–г* – примеры архитектур, описанных в работе [2]; *д* – пример архитектуры, предложенной в блоге Романа Кузьмина 29.06.08; *е* – пример "трехуровневой архитектуры", описанной в работе [3]

может быть поставлена в соответствие модель, представленная на рис. 5.

Следуя принципу декомпозиции, представим (в качестве примера) на следующем шаге детализации системную модель внутренней среды программного продукта, соответствующую вершине "правообладатели" (см. рис. 4). Для него внутренняя среда есть результат взаимодействия двух триад (рис. 6): Руководитель организации-Заказчика – представитель организации-Заказчика (консультант) – пользователь с одной стороны. Руководитель организации-Исполнителя – менеджер программного проекта – разработчики с другой. Более детальному описанию выделенных триад посвящено достаточно много работ [4, 12, 13 и многие другие], поэтому в рамках настоящей статьи они затрагиваться не будут. Мы стремились показать, что "пирамида программного проекта" создает методическую основу для построения системных, структурных и математических моделей, в разных аспектах описывающих программный проект.

## 5. Когнитивная модель программного проекта

### *5.1. Семантическая сеть программного проекта*

В работе [14] подчеркивается взаимосвязь онтологии, описывающей бизнес-процесс, и параметров ресурсов, необходимых для реализации бизнес-процессов. Отмеченные обстоятельства создают основу построения когнитивной модели программного проекта, основу которой составляет грань "пирамиды проекта", выделенная на рис. 7 штриховкой.

Приведенная ниже модель соответствует V-модели жизненного цикла программного продукта и предполагает использование "тяжелых" методологий реализации программных систем.

В составе программного проекта выделяются следующие концепты:

А. *Требования правообладателя*, которые определяют внешний облик программного продукта. В многочисленных литературных источниках подчеркивается, что

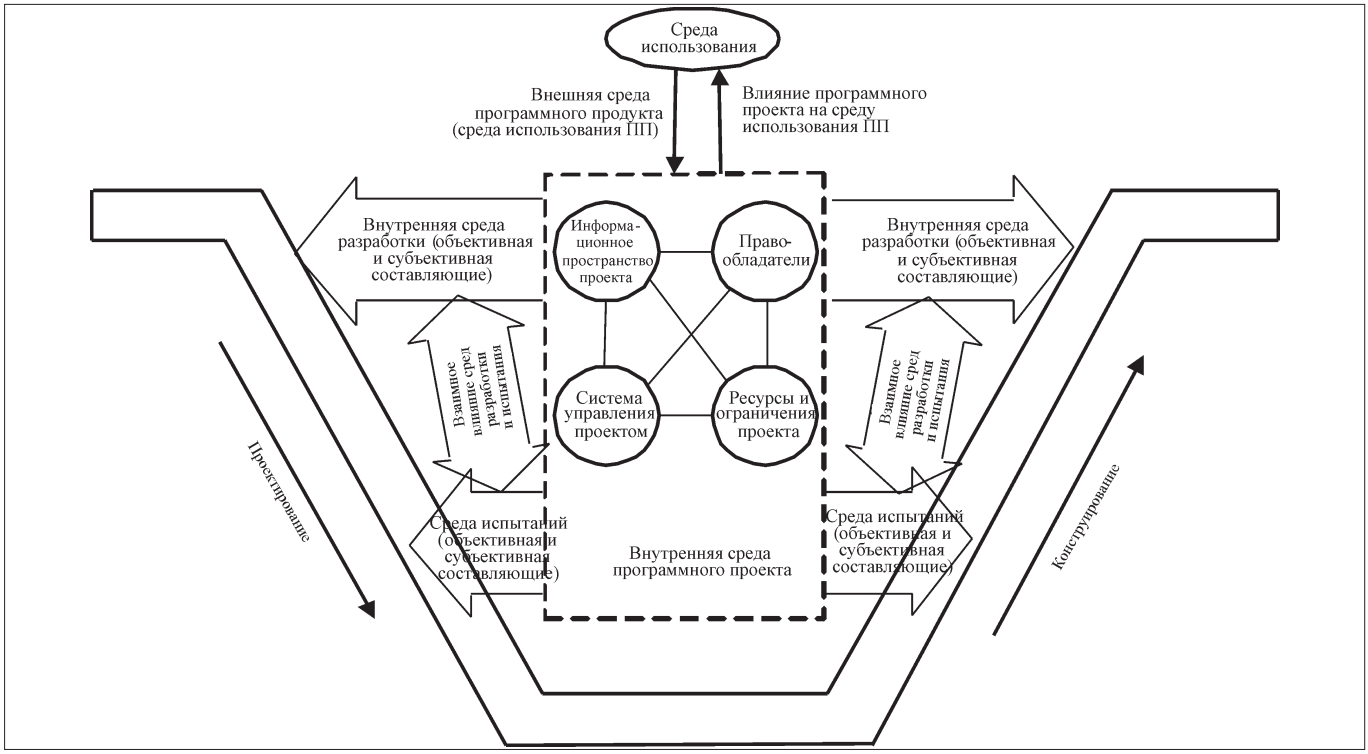


**Рис. 4. Внешняя и внутренняя среда программного проекта**

требования заказчика/пользователя, представленные в виде спецификации требований, являются фундаментом реализации программного продукта. Подчеркивается важность формирования полной, сбалансированной, непротиворечивой системы функциональных и нефункциональных требований, что в свою очередь является основой формирования технического задания на программный продукт, а также обоснованного выделения ресурсов.


**Рис. 5. Взаимосвязь внешней и внутренней сред разработки программного продукта**

***B. Техническое* задание *на программный продукт*.** В отличие от требований правообладателей, техническое задание (ТЗ) дает целостное описание программного продукта с точки зрения разработчика. При формировании ТЗ важно обеспечить баланс между функциональными и нефункциональными требованиями [5], а также заложить основы для обеспечения разумного соотношения между функциональностью программного продукта, его ценой и временем создания.

***C. Ресурсы, выделяемые на реализацию программного проекта.*** Согласно работе [6] ресурсы могут включать в себя такие разнообразные объекты, как персонал, оборудование, основные средства, инструменты, а также коммунальные услуги – энергию, топливо и инфраструктуру средств связи.

В литературе [7] подчеркивается, что типичной ситуацией является несоответствие выделенных ресурсов реальному объему работ по проекту. Предлагается так называемая двухэтапная схема финансирования, которая предполагает на первом этапе выделение незначительных ресурсов на выполнение работ по оценке реалистичности проекта, а также оценки ожидаемого объема работ по проекту. На втором этапе, в зависимости от результатов первого этапа, либо принимается решение об отклонении проекта, либо происходит выделение ресурсов, соответствующих ожидаемому объему работ.


**Рис. 6. Модель среды разработки программного продукта, определяемая взаимоотношениями Заказчика и Исполнителя**

***D. Технологии и практики проектирования, конструирования и испытания программных продуктов.*** Для современных программных проектов характерны сложность создаваемых программных продуктов, ограниченное время, выделяемое на их создание, высокие требования к качеству создаваемых ими информационных продуктов (оказываемых информационных услуг). Разработка современных программных продуктов невозможна без использования *CASE*-инструментов, позволяющих автоматизировать работы на разных стадиях реализации программного продукта. При этом требования к возможностям *CASE*-инструментов тем выше, чем сложнее создаваемый программный продукт. В свою очередь, затраты на приобретение и освоение *CASE*-инструментов тем больше, чем больше предоставляемые ими возможности.

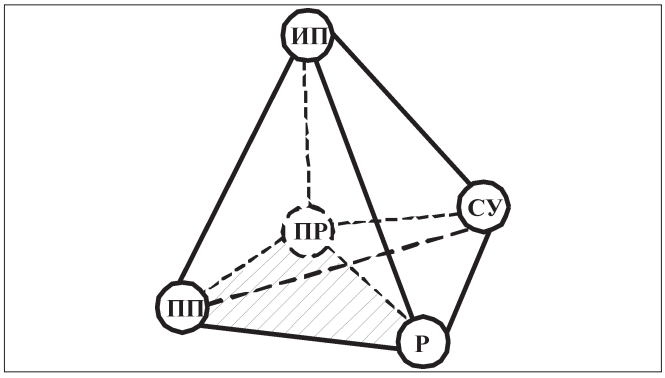


**Рис. 7. Модель "онтология бизнес-процессов – ресурсы проекта – программный продукт"**

***E. Внедрение и опытная эксплуатация программных продуктов.*** Затраты на внедрение программного продукта тем больше, чем он сложнее. В ряде случаев затраты на внедрение, в том числе обучение пользователей, могут превысить затраты на создание программного продукта.

Некачественная реализация программного продукта приводит к необходимости его доработки в процессе внедрения и опытной эксплуатации. Чем ниже качество, тем больше ресурсов приходится привлекать для внесения изменений в программный продукт.

Семантическая сеть, построенная на основе выделенных концептов и связей между ними, представлена на рис. 8.

***5.2. Знаковый ориентированный орграф***

Приведенной семантической сети можно поставить в соответствие знаковый ориентированный граф, показанный на рис. 9. Ниже приводится описание характера влияния пар разных концептов друг на друга.

***AB*** – влияние требований правообладателя на сложность разработки ТЗ. Чем больше требований выдвигает заказчик, тем сложнее разработка ТЗ. Это в значительной степени обусловлено проблемами перевода слабо структурированных, зачастую противоречивых требований заказчика в систему не противоречивых, полных, сбалансированных требований и ограничений.

***BA*** – влияние сложности ТЗ на объем работ по проекту. Чем сложнее ТЗ, тем больше ожидаемый объем работ.

***AC*** – инвестирование проекта. Чем больше ожидаемый объем работ, тем больше объем привлекаемых ресурсов.

***BD*** – влияние сложности технического задания на техники и технологии реализации программного продукта. Чем более сложным является ТЗ, и чем более жесткие ограничения налагаются на время реализации проекта, тем больше значимость формальных приемов разработки и документирования результатов, тем более необходимо применение *CASE*-инструментов.

***DB*** – результаты сопоставления плановых характеристик качества программного продукта с фактическими характеристиками качества. Чем выше качество программного продукта, тем меньше расхождение характеристик.

***CD*** – влияние размера доступных ресурсов на техники и технологии проектирования и конструирования программного продукта. Чем больше ресурсов доступно, тем более квалифицированные специалисты, владеющие приемами разработки сложных программных продуктов, могут быть привлечены. Чем больше доступных ресурсов, тем более развитые *CASE*-инструменты могут быть приобретены и освоены разработчиками программного продукта.

***DE*** – влияние качества программного продукта на затраты, связанные с его внедрением. Чем выше каче-

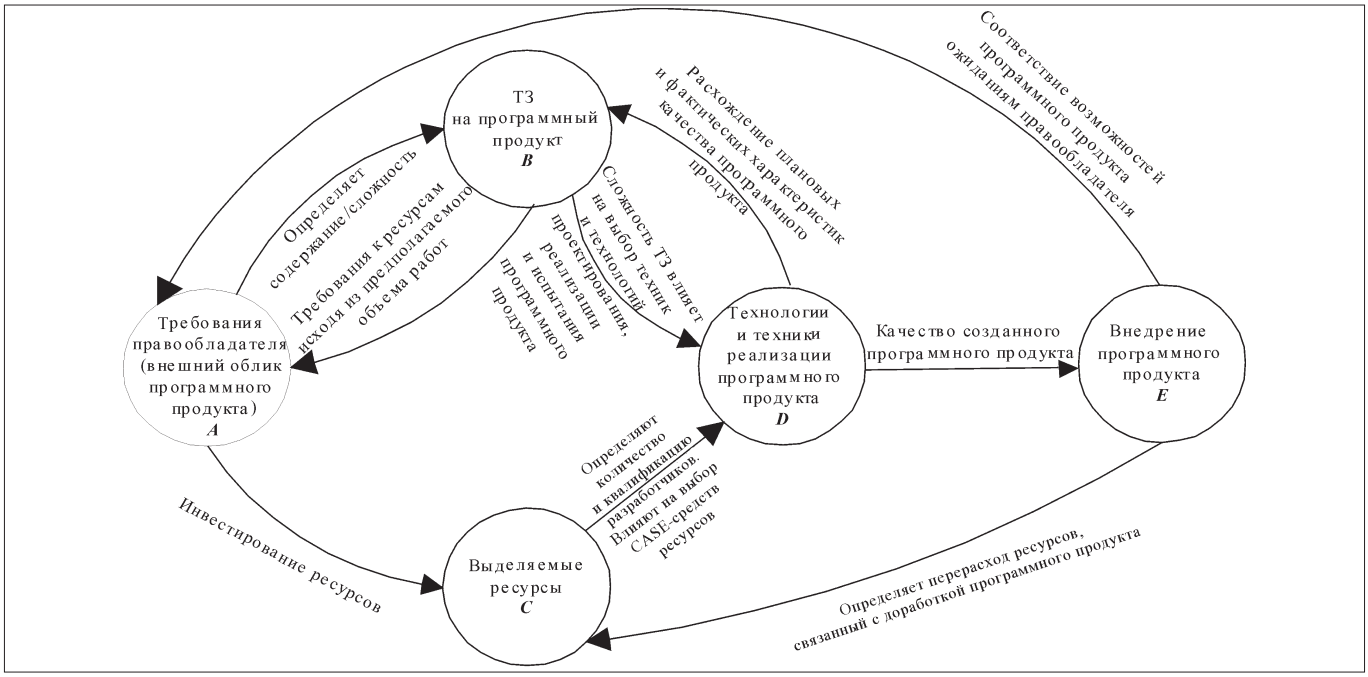


**Рис. 8. Семантическая сеть программного проекта**

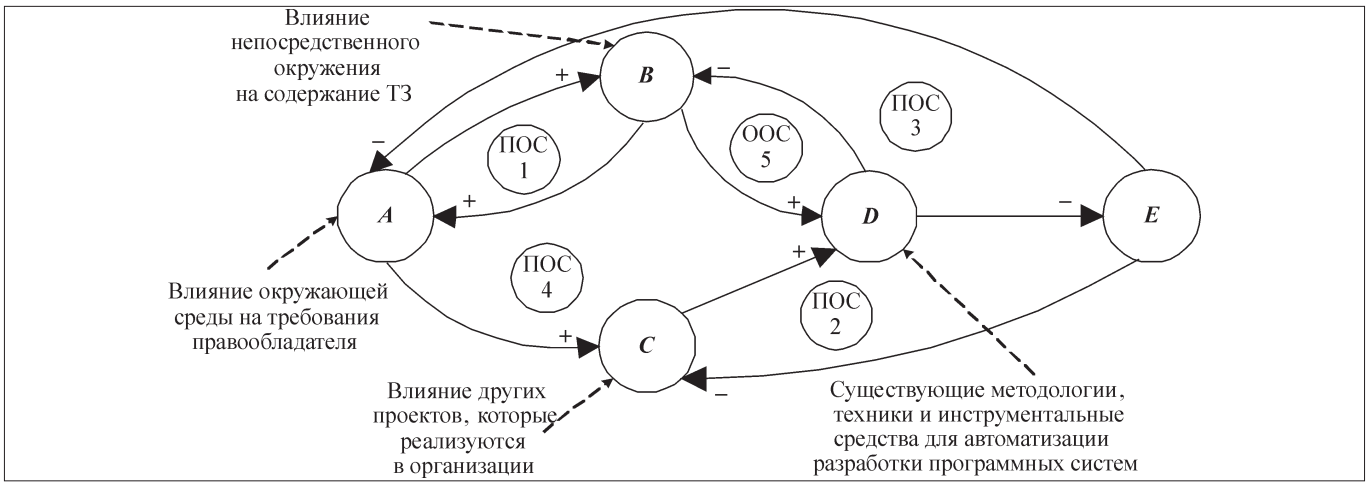


**Рис. 9. Знаковый ориентированный орграф, соответствующий семантической сети программного проекта**

ство, тем меньше требуется доработок на стадии внедрения.

***EC*** – влияние результатов внедрения на дополнительное (по сравнению с плановым) расходование ресурсов. Чем больше доработок возникает на стадии внедрения, тем больше внеплановый перерасход ресурсов.

***EA*** – влияние результатов внедрения программного продукта на признание его ценности правообладателем программного продукта. Чем меньше расхождение свойств разработанного программного продукта по сравнению с ожиданиями пользователей, тем выше его ценность с точки зрения пользователя.

### *5.3. Анализ взаимного влияния концептов*

Контур ***ABA*** (положительная обратная связь (ПОС 1)) соответствует анализу целесообразности инвестирования проекта. Основу анализа составляет ожидаемая системная эффективность, имеющая экономическую, социальную, техническую и научную составляющие [11]. Предполагается, что бóльшая системная эффективность предполагает бóльший объем работ, что соответственно требует бóльших объемов инвестирования.

Контур ***CDEC*** (ПОС 2) означает, что ресурсная поддержка приобретения и освоения новых технологий и техник способствует повышению квалификации разработчиков. Повышение квалификации разработчиков, с одной стороны, стимулируют их настаивать на приобретении и освоении все более совершенных инструментальных средств. С другой стороны, наличие развитых инструментальных средств и высокая квалификация разработчиков приводят к повышению качества программных продуктов при одновременном уменьшении внеплановых ресурсных затрат. Это, в свою очередь, подтверждает целесообразность увеличения ресурсной поддержки приобретения и освоения новых технологий и техник.

Контур ***ABDEA*** (ПОС 3) означает, что сложные для реализации и освоения требования стимулируют развитие и освоение новых, более совершенных технологий создания программных продуктов. Использование новых технологий приводит к появлению более полезных, с точки зрения правообладателя, программных продуктов. Освоение новых программных продуктов пользователями способствует повышению их квалификации, и появлению у них новых потребностей [4].

Контур ***ACDEA*** (ПОС 4) означает то, что реализация более сложных требований вынуждает правообладателя программного продукта увеличивать объем инвестирования. В свою очередь наличие достаточных ресурсов позволяет привлекать более квалифицированных специалистов, осваивать современные технологии и техники, приобретать более совершенные *CASE*-средства.

Контур ***ABC*** – реализация программного продукта в соответствии с согласованным ТЗ. Отрицательная обратная связь (ООС 5) означает, что при неизменном ТЗ и правильных технологиях и техниках реализации будет создан качественный программный продукт. Фактически этот контур соответствует "водопадной" модели жизненного цикла программного продукта.

В совокупности ООС 5 и ПОС 3 представляют собой сочетание "спиральной" (ПОС 3) и "водопадной" (ООС 5) моделей. Если считать, что технологии и техники реализации программного продукта предполагают формирование команды и управление им, то в совокупности ООС 5 и ПОС 3 образуют то, что в литературе именуется *MFS*-методологией.

## Заключение

Предложенная системная модель программного проекта является обобщением описанных в литературе системных моделей – "треугольников проекта". Ее методологическую основу составляет основной принцип исследования сложных систем – принцип комплексности. Содержание предлагаемой модели подчеркивает тот факт, что успех проекта определяется не только ресурсами проекта и организацией системы управления, но и человеческим фактором, а также информационным пространством проекта.

"Пирамида программного проекта" позволяет с системных позиций подходить к построению комплексов системных, структурных и математических моделей, описывающих процессы реализации программного проекта и управления ими, что дает возможность повышать управляемость проектов и предсказуемость их результатов.

## СПИСОК ЛИТЕРАТУРЫ


1. **Брю Г.** Шесть сигм для инженеров. М.: ФАИР-ПРЕСС, 2004. 272 с.

2. **Васкевич Д.** Стратегии клиент/сервер. Руководство по выживанию для специалистов по реорганизации бизнеса. Киев.: "Диалектика", 1994. 384 с.

3. **Виттих В.А.** Парадигма ограниченной рациональности принятия решений: препринт. Самара: Учреждение Российской академии наук "Институт проблем управления сложными системами РАН", 2009. 28 с.

4. **Виттих В.А., Скобелев П.О.** Метод сопряженных взаимодействий для управления распределением ресурсов в реальном масштабе времени: препринт. Самара: Институт проблем управления сложными системами РАН, 2008. 16 с.

5. **Грекул В.И., Денищенко Г.Н., Коровкина Н.Л.** Проектирование информационных систем: курс лекций: учеб. пособие для студентов вузов, обучающихся по специальностям в области информ. технологий. М.: Интернет-Ун-т Информ. Технологий, 2005. 304 с.

6. **Демарко Т., Листер Т.** Человеческий фактор: успешные проекты и команды. 2-е издание. Пер. с англ. СПб.: Символ-Плюс, 2005. 256 с.

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

8. **Макконнелл С.** Остаться в живых. Руководство для менеджера программных проектов. СПб.: Питер, 2006. 240 с.

9. **Макконнелл С.** Сколько стоит программный проект. М.: "Русская Редакция", СПб.: Питер, 2007. 297 с.

10. **Введение** в программную инженерию и управление жизненным циклом ПО. [Блог Сергея Орлика]. 2005. URL: http://sorlik.blogspot.com

11. **Халл Э., Джексон К., Дик Д.** Разработка и управление требованиями. Telelogic, 2005. 230 с.

12. **IEEE** Guide to the Software Engineering Body of Knowledge – SWEBOK, 2004.

13. **Project Management** Institute 2000 "A Guide to the Project Managemnt Body of Knowledge" Newton Square. Pa: Project Management Institute.

14. **ГОСТ Р ИСО МЭК 15288–2005.** Информационная технология. Системная инженерия. Процессы жизненного цикла систем. Введ. 2007–01–01. М.: Госстандарт России, 2007.

15. **Ефимов В.В.** Улучшение качества проектов и процессов: учеб. пос. Ульяновск: УлГТУ, 2004. 185 с.

УДК: 004.4'24

**С.В. Гусс,** асс., ОмГУ им. Ф.М. Достоевского

E-mail: InfoGuss@gmail.com

# Пакет вспомогательных средств для построения семейства программных систем в определенной предметной области

*Рассматриваются общие вопросы разработки семейства программных систем в рамках линейки программных продуктов на основе систематического подхода к повторному использованию производственных активов. Проводится обзор оснований, лежащих в корне данной практики. Обращается внимание на профессиональные характеристики разработчиков конечных продуктов и требуемый уровень знаний для их эффективной работы.*

***Ключевые слова:*** *семейство программных систем, линейка программных продуктов, повторное использование программных активов, фабрика разработки программ, моделирование предметной области*

## Введение

Согласно работам [1–4], специально созданные или отобранные, адаптированные и настроенные на определенное семейство программных систем средства разработки, в сочетании с определенной профессиональной подготовкой исполнителей проекта, способны увеличить производительность процесса создания компьютерных приложений и обеспечить высокое качество разрабатываемого продукта.

К таким средствам относятся модели, каркасы, шаблоны и предметно-ориентированные языки моделирования/программирования, способные связать составные части приложения в рамках общей архитектуры семейства программных систем [1, 5, 6]. В моделях отображаются общие очертания программных единиц. Каркасы определяют характерные для программных единиц повторно используемые абстракции. Предметно-ориентированные языки делают процесс работы с этими абстракциями производительным, а шаблоны повышают эффективность процесса, ограничивая возможность совершения некорректных действий, в то же время, указывая верный путь работы с абстракциями. Добавив к этому автоматические программные инструкции, оказывающие помощь разработчикам непосредственно во время создания приложений, можно объединить все вместе в специальный пакет, расширяющий возможности интегрированной среды разработки. К примеру, среда *Microsoft Visual Studio* 2010 имеет в своем арсенале все необходимые для расширения средства, включая специальные шаблоны проектов, есть нечто подобное и в средах *IDE Eclipse* и *Embarcadero RAD Studio*. Особое выделение продуктов компании *Microsoft* вызвано тем, что в рамках одной из ее исследовательских инициатив проводятся поиск и развитие методов автоматизации, а в конечном счете и индустриализации разработки программных продуктов, что должно повысить уровень зрелости разработки программного обеспечения. Эта инициатива, названная фабриками программного обеспечения [1], рассматривается автором статьи в теоретическом и практическом аспекте.

Наряду с общими рекомендациями по составлению пакета вспомогательных средств, описанными в работах [1, 2, 7], необходимы также рекомендации по созданию пакетов для конкретной предметной области [4]. В качестве предметной области в одном из представленных в статье проектов программного обеспечения выступают игровые обучающие программные системы с лингвистической составляющей [8–10], оптимально подходящие с точки зрения автора (подкрепленной исследованиями ученых в области педагогики и психологии [12]), для учебного процесса. Корот-

кое описание проекта в конце статьи представлено, чтобы познакомить читателей с направлением программных разработок, ведущихся в рамках научного исследования автора представленной работы и с предметной областью обучающих программных систем.

О явлении компьютерных игр в целом можно узнать из исследований в области культурологии, представленных в работе [11]. Вопрос влияния игр на психологию играющих и обучающихся с их помощью с разных сторон рассмотрен в работе [12]. Что же касается внутренней (не только технической) составляющей компьютерных игр, то довольно полно информация по данному вопросу представлена в работе [13].

О необходимости и практичности конкретизации современных методов разработки компьютерных приложений для отдельных предметных областей говорится во множестве работ, часть из которых будет упомянута в тексте статьи.

Между делом, также рассматриваются и этические вопросы, касающиеся профессиональной разработки приложений для специальной предметной области. Автор упомянет и об исторических моментах, а также основах, лежащих в затронутых темах. Не упущен из виду и такой аспект профессиональной разработки, как человеческий фактор [3], связанный с психологическими особенностями разработчиков, их профессиональными качествами и набором необходимых им знаний [14]. Данные вопросы рассматриваются в общих чертах, они основываются на существующих сегодня знаниях, тем не менее, для тех, кто хочет узнать об этом подробней, приводятся литературные источники.

## Профессиональная разработка приложений

В наше время от специалистов в области программной инженерии – непосредственно от профессиональных разработчиков – требуются определенные знания о производстве программных систем "в определенной тематической области их применения" [3]. В работе [4] говорится, что для эффективного внедрения инноваций инженерии программного обеспечения, необходимо выразить их в терминах, "понятных для каждой конкретной подотрасли". Это подразумевает необходимость конкретизации знаний в области разработки приложений для ***конкретной прикладной области*** [15].

Согласно работе [14], от сегодняшних разработчиков требуется больше, чем просто написание программного кода. Необходимо иметь представления обо всем ***жизненном цикле*** программного обеспечения [16, 17], и в соответствии со своей специализацией уметь пользоваться особыми знаниями. Что самое главное, там также упоминается о том, что "эффективное использование специфических для предметной области методов, средств и компонент в большинстве случаев обеспечивает успешность разработок с использованием программной инженерии". Таким образом, аналитики, архитекторы, проектировщики и разработчики помимо знаний из инженерии программного обеспечения должны обладать хотя бы минимальной информацией из специальной предметной области [3]. Минимум информации в данном случае должен определяться уровнем "критичности" проекта программного обеспечения, т.е. тем, какое воздействие реализация данного проекта может оказать на свое окружение (в первую очередь, на исполнителей проекта и пользователей, а впоследствии и на общество в целом или его части). Именно здесь решающее значение имеет человеческий фактор [3].

Все это говорит в пользу подхода к профессиональной разработке в определенной предметной области. В этом случае применяется ***моделирование предметной области*** [18], появляются специальные термины и образуются некоторые закономерности, которые необходимо учитывать, а в определенных случаях требуются повышенная внимательность и способность длительной концентрации внимания [3], тогда оправдана повышенная предусмотрительность исполнителей, ответственных за критические аспекты проекта.

Профессиональная разработка включает [3] ***аспекты качества***, управления и экономики. Вопросы качества и управления имеют существенное значение для проекта, а вопросу экономики, особенно в случае больших проектов уделяется особое внимание, в частности, согласно [19], формируется новая научная дисциплина – "экономика жизненного цикла программных средств". В рамках аспекта качества должны рассматриваться такие вопросы, имеющие отношения к нефункциональным требованиям программного обеспечения, как удобство, надежность, производительность и поддерживаемость (сопровождаемость) [1]. Как видно, нефункциональные требования должны распространяться не только на конечный продукт, но и на процесс его производства.

## Средства разработки программных продуктов в определенной предметной области

В наборе сегодняшнего разработчика можно встретить следующие средства:

- ***модели*** [15, 18, 20], подразумевается также наличие инструментов для работы с моделями в определенной среде разработки;
- ***шаблоны*** (анализа, архитектурные, проектирования, тестирования и т.д.) [5, 21], в литературе часто встречаются как образцы или паттерны;
- ***предметно-ориентированные языки*** [1, 2, 22, 23];
- ***компоненты*** [24–28];
- ***рекомендации, макросы и скрипты*** [1, 7], для автоматизации определенных частей процесса разработки программного обеспечения.

Все эти средства стали плодом развития ***дисциплины разработки компьютерных приложений***. С развитием появлялись новые методы, новые инструменты, какие-то отбрасывались, а какие-то укоренялись и со-

вершенствовались. Нельзя сказать, что до появления объектно-ориентированной парадигмы не было представленных выше инструментов, но массово о них заговорили именно с той поры, начиная с конца 1980-х годов. К этому времени уже была определенная теоретическая база, а развитие технологий позволило опробовать это все на практике, внести коррективы и сделать новые предложения.

Одной из фундаментальных работ считается публикация [24], представленная Малькольмом Дугласом Макилроем (*Malcolm Douglas McIlroy*) в конце 60-х годов прошлого века на конференции, впервые посвященной ***инженерии программного обеспечения***. В этой работе делаются предположения о том, как и в какой форме могли бы быть полезны компоненты программного обеспечения. Конкретно предлагались методы каталогизации компонентов, сегодняшний же подход предполагает взамен метод генерации или автоматического подгона из общего представления [2]. Немалое развитие в этом направлении произвел Дэвид Парнас (*David Parnas*), являющийся основоположником множества фундаментальных принципов разработки программного обеспечения. Его главные работы были посвящены процессу проектирования компонентов в рамках семейства родственных программных систем.

Что касается моделей, то большое увлечение ими началось с работ по созданию ***универсального языка моделирования***, с помощью которого, используя единую систему обозначений для выражения своих намерений, могли бы взаимодействовать и профессионально общаться разработчики программных систем разных специализаций, от аналитика до программиста, реализующего отдельный модуль системы. Это вылилось в создание ***унифицированного языка моделирования*** UML, и здесь не обойтись без упоминания коллектива, включающего Гради Буча (*Grady Booch*), Джеймса Рамбо (*James Rumbaugh*) и Айвара Джекобсона (*Ivar Jacobson*), которые объединили и развили всевозможные, в том числе и собственноручно разработанные, подходы к моделированию предметной области программного обеспечения [18]. Язык UML до сих пор не утратил своей популярности в кругу сторонников объектно-ориентированной парадигмы создания программного обеспечения. И сегодня проводят различные конференции и семинары, на которых предлагаются эффективные пути его использования в повседневной разработке. Например, совсем недавно прошел вводный вебинар на тему "Теория и практика разработки объектно-ориентированных программных систем". В его рамках авторы Денис Иванов и Фёдор Новиков предлагали свой взгляд на разработку, главным образом на специфицирование программных систем, представляя модели, созданные в нотации языка UML средством общения в виде задания вопросов одними ответственными лицами (термин из предлагаемой ими методологии) и ответа на них другими лицами.

Фундаментальный труд по ***шаблонам*** – работа [21], появившаяся в середине 1990-х годов. В рамках этого проекта по каталогизации шаблонов проектирования принимали участие Эрих Гамма (*Erich Gamma*), Ричард Хелм (*Richard Helm*), Ральф Джонсон (*Ralph Johnson*), Джон Влиссидес (*John Vlissides*). Однако вдохновителем идеи является Кристофер Александер (*Christopher Alexander*), не имеющий прямого отношения к компьютерным наукам, он одним из первых заговорил о языке шаблонов, правда применительно к архитектурным сооружениям. По некоторым сведениям, например [2], он не был удовлетворен своей теорией, обосновывая это тем, что архитектурные сооружения, построенные с использованием шаблонов, получаются слишком однообразными, что, к счастью, не является проблемой для внутреннего строения проекта программного обеспечения, а наоборот, идет ему на пользу.

***Предметно-ориентированные языки*** не являются чем-то очень новым, они были раньше и широко используются сегодня [23]. Например, SQL, язык для работы с запросами к базе данных, регулярные выражения для работы с текстом и др. Однако SQL – язык самостоятельный, а разработка подобных языков для других функций в рамках единственного проекта оправдана в случаях существенной сложности и масштаба проекта. Поэтому до сегодняшнего дня, когда платформенные технологии развиты настолько, что позволяют создавать языки уже на имеющейся технологической базе, без создания компилятора и не требующие от разработчиков детальных знаний процесса их создания, практика разработки предметно-ориентированных языков для отдельного проекта не имела широкого применения. Однако сейчас она завоевывает популярность. Выпускаются необходимые инструменты, выходят специальные статьи, проводятся всевозможные конференции. Сегодня имеются труды, из которых можно узнать об этом явлении (в частности [1, 22]) и даже ощутить его на практике (например, благодаря работам [22, 23]). В работе [23] все внимание уделяется разработке ***текстовых языков***, а в работе [22] – ***графических***, где описание сущностей происходит с помощью графических пиктограмм и линий с нанесением поясняющей информации. Первые хороши для решения крупных проблем. Вторые подходят для решения средних и небольших по масштабу задач, но требующих от разработчика определенной внимательности.

В рамках конкретной парадигмы разработки могут использоваться различные ***модели***, ***статические*** и ***динамические***, описывающие структуру системы или отдельной ее части и взаимодействие элементов внутри нее или с внешним миром. Для эффективного повторного использования модели, основной проблемой, препятствующей такому использованию, являются ***границы*** модели. Границы мешают использовать ее в другом контексте без добавления к ней определенных изменений [15]. Эта проблема частично решается применением шаблонов проектирования.

Цель ***шаблонов*** – сохранить обобщенные знания экспертов с тем, чтобы впоследствии их можно было использовать в определенном контексте. В основном,

знания обобщаются в определенной ***горизонтальной предметной области*** (выделенной на основе классов частей систем исходя из функциональных возможностей) с тем, чтобы их можно было использовать в контексте ***вертикальной предметной области*** (выделенной на основе классов систем согласно области применения). Чтобы эффективно применять шаблоны проектирования, нужно, во-первых, иметь о них представление, а во-вторых, желательно иметь опыт их применения, чего может и не быть или он может быть утрачен со временем. Одно из решений проблемы – встроить эти знания в каркас.

***Каркас*** характеризует фиксированную для предметной области часть программного обеспечения, в зависимости от масштаба и способов использования может называться иначе, например, в особых случаях именуется ***программным интерфейсом разработчика***, в других – ***интерпретатором***, в особо крупных и сложных системах может называться ***платформой***. Он облегчает разработку системы в целом или отдельной ее части, благодаря уже имеющейся в наличии общей структуры будущей программной единицы. От разработчика требуется лишь провести детализацию каркаса. Возможная проблема состоит в том, что прежде чем использовать каркас, нужно знать, как проводить его настройку, если каркас большой (например, в случае с платформой), ситуация осложняется. Для облегчения работы с каркасом можно использовать специально созданные для него предметно-ориентированные языки.

Главное преимущество ***предметно-ориентированных языков*** – облегчение процесса детализации каркаса и возможность с их стороны, благодаря встроенным программным рекомендациям и скриптам, объединять компоненты или составляющие в рамках компонента без рутинной работы со стороны разработчика. Одни языки могут применяться для решения мелких проблем в рамках отдельного компонента, другие – для решения крупных проблем в рамках целой системы.

Очевидно, что перечисленные средства взаимосвязаны, и чтобы избавить разработчиков от путаницы в их использовании и неудобств, связанных с совместимостью, целесообразно объединить их в рамках единого ***пакета***. Подробнее этот вопрос будет раскрыт в разделе, посвященном фабрикам разработки программ.

Стоит сказать еще несколько слов о подходах, которые оказали непосредственное влияние на появление этих инструментов. Из работ [1, 2, 22] можно выделить следующие подходы, имеющие отношение к разработке программного обеспечения, учитывая особенности определенной предметной области.

- ***Разработка с использованием моделей*** (*Model-Driven Development*). В основе подхода лежит идея, что из модели можно сгенерировать пригодный для применения код. Одно из перспективных направлений подхода (в рамках инициативы OMG – консорциума, занимающегося разработкой и продвижением объектно-ориентированных технологий и стандартов) – ***архитектура***, управляемая моделью (*Model-Driven Architecture*) [29], с которым связаны такие понятия, как машинно-независимая и платформо-независимая модель.
- ***Языкоориентированное программирование*** (*Language-oriented Programming*). Термин введен компанией *JetBrains*. Предполагает создание специального языка для решения определенных задач программирования.
- ***Предметно-ориентированное моделирование*** (*Domain-Specific Modeling*). Объединяет ряд направлений, связанных идеей создания предметно-ориентированных языков для решения задач определенной предметной области.
- ***Родовое программирование*** (*Generic Programming*). Направлено на решение проблемы структурирования компонентов реализации, из которых строятся конечные программные системы. Связано с поиском наиболее абстрактных представлений эффективных решений для оптимального повторного использования компонентов.
- ***Ментальное программирование*** (*Intentional Programming*). Главная идея, являющаяся трамплином для двух нижеперечисленных подходов, такова: необходима расширяемая среда программирования и метапрограммирования. Такой подход предполагает автоматизированное редактирование и рефакторинг кода. Предусматривает решение проблем спутывания кода, включая тем самым идеи ***аспектно-ориентированного программирования*** (*Aspect-Oriented Programming*).
- ***Порождающее программирование*** (*Generative Programming*). Идея автоматической (в идеале) генерации приложений объединяет такие темы, как ***инженерия предметной области, моделирование характеристик*** и различные техники генерации программ.
- ***Фабрики разработки программ*** (*Software Factories*). Подход включает в общем виде идею, представленную в данной статье. А именно, объединение статического содержимого (шаблонов, моделей, предметно-ориентированных языков, компонентов и инструкций) с динамическим содержимым (поддающимися настройке процессами, мастерами, тестами) в рамках пакета, расширяющего интегрированную среду разработки.

Несмотря на то, что два последних подхода предлагают генерацию приложений или его частей из моделей, они, тем не менее, предусматривают работу со стороны разработчика, связанную с добавлением программного кода. Это основное отличие от ***CASE-технологий*** и архитектуры, управляемой моделью, которые предлагают генерацию предложения "нажатием одной кнопки".

## Инженерия предметной области

Существует множество работ, излагающих идеи инженерии предметной области. Здесь можно выделить два класса направлений. Первый класс составляют работы, описывающие ***идеи для создания методов*** разработки приложений, второй класс составляют работы, предлагающие непосредственно ***методы разработки*** на основе идей первого класса и их практическое применение. В качестве примера первого класса можно привести работу [15], в качестве примера второго клас-

са – [30]. В работе [15] излагается так называемая ***теория предметной области*** (*Domain Theory*) для разработки программного обеспечения, а в работе [30] – применение приемов различных парадигм разработки, в том числе связанных с теорией предметной области на практике, используя язык программирования C++.

В работе [2] проведен обзор различных методов анализа и инженерии предметной области. Наибольшее внимание в ней уделяется методу ***характеристико-ориентированного анализа предметной области*** (*Feature-Oriented Domain Analysis*) и методу ***моделирования предметной области организации*** (*Organization Domain Modeling*).

Инженерия предметной области (*Domain Engineering*) имеет прямое отношение к вопросу ***повторного использования***. Согласно определению, взятому из [2], она имеет дело со сбором, систематизацией и сохранением накопленного опыта создания программных систем в "форме средств многократного применения в рамках определенной предметной области". Кроме того, она обеспечивает методы, способствующие повторному использованию этих средств в процессе разработки новых приложений, облегчает их поиск, классификацию, распространение, адаптацию и сборку.

Главное отличие инженерии предметной области от традиционных методов разработки состоит в следующем. Во-первых, на стадии анализа вместо определения требований к отдельной системе, проводится определение требований к семейству систем. Во-вторых, на стадии проектирования вместо разработки проекта отдельной системы проводится разработка проекта семейства систем, а на этапе реализации, помимо реализации самой системы, производятся ***многократно используемые компоненты***, которые можно будет использовать в будущем.

## Систематическое повторное использование

Тема повторного использования применительно к разработке программных продуктов весьма полно представлена в работе [25]. Примечательно, что в работе, наряду с организационными аспектами внедрения, представлены ***метрики повторного использования***. К ним относятся атрибутные, структурные, функциональные метрики, метрики, характерные для всего приложения в целом, метрики инженерии предметной области и метрики организационного уровня.

Практика введения систематического повторного использования главным образом предполагает создание ***проблемно-*** и ***предметно-ориентированных абстракций***, а также каркасов, в рамках которых эти повторно-используемые абстракции могут быть разработаны путем детализации. Современная тенденция предполагает наличие предметно-ориентированного языка для детализации каркаса, тем самым решается множество проблем, связанных с человеческим фактором, таких как уровень знаний и объем единовременного восприятия информации во время разработки компонента.

Систематическое повторное использование указывает на необходимость разработки не отдельного приложения, а семейства программных систем, что в свою очередь решается введением в производственный процесс такой практики, как линейки программных продуктов. С разработкой семейства связаны такие вопросы, как общность и изменчивость характеристик членов семейства. Существуют различные подходы для выявления этих особенностей.

## Семейства программных систем и линейки программных продуктов

Характерной особенностью ***семейства программных систем*** является наличие обобщенной архитектуры, описывающей все множество членов конкретного семейства, указывая при этом так называемые точки изменчивости, т.е. те места, в которых имеют место различия между членами семейства.

***Линейки программных продуктов*** реализуют семейство программных систем, применяя имеющиеся производственные активы в рамках организационного процесса. К производственным активам относятся архитектурные образцы, шаблоны проектирования, языки, каркасы, компоненты и т.д. ***Организационный процесс*** накладывает определенные ограничения на продукт и его производство. К ограничениям на продукт относятся различные требования к стандартам, интерфейсам, пакетированию, локализации, лицензированию. К ограничениям на производство относятся требования к выбору технологий, платформ, критериям приемки и отчетности, сюда также относятся требования к бюджету и планированию.

Основополагающей работой в области разработки семейства программных систем является [31]. Заслуживающие внимания работы, касающиеся линеек программных продуктов – [32, 33].

## Фабрики разработки программ

Фундаментальной работой по фабрикам разработки программ является работа [1] авторского коллектива во главе с Джеком Гринфилдом (*Jack Greenfield*). Практическому аспекту фабрик программного обеспечения посвящена работа [7] Гунтера Ленца (*Gunther Lenz*).

Фабрика разработки программного обеспечения представляет собой линейку программных продуктов, которая благодаря шаблону, имеющему в своем основании схему, называемую схемой фабрики программного обеспечения, может конфигурировать инструменты и процессы для ***автоматизации разработки***. Фабрика предполагает наличие специальных компонентов, способных к адаптации, сборке и конфигурированию на основе каркаса.

Основные идеи фабрик программного обеспечения: повышение уровня ***индустриализации*** отрасли разработки программного обеспечения, повышение ***экономической эффективности*** последующих приложений из определенного класса, экономия за счет области применения (благодаря уже решенным под-

проблемам предметной области), систематическое повторное использование.

В работе [1] приводится пример построения семейства приложений онлайновой электронной коммерции. Главное, на что здесь стоит обратить внимание – разработка приложения проводится компанией, которая производит и продает разновидности ранее разработанных ею же программных продуктов. Опыт разработки схожих систем наводит производителей на мысль о создании продукта, предназначенного для быстрой разработки приложений из класса, с которым они уже работали, т.е. имеют определенный опыт и набор производственных активов. Их успех зависит от того, насколько эффективно они смогут создавать продукты из имеющихся многократно использующихся компонентов, являющихся частью производственных активов.

***Построение*** семейства приложений происходит в несколько этапов.

Первый этап – ***разработка линейки продуктов***. Включает в себя три подэтапа, состоящих из отдельных фаз.

Первый подэтап – ***анализ линейки продуктов***. Здесь решается вопрос, относительно того, какие продукты будут производиться фабрикой. Первая фаза – ***определение линейки продуктов***. Цель – достичь понимания того, как архитектура, учитывающая изменчивость, будет способствовать построению множества схожих, но в то же время уникальных программных продуктов. Также необходимо идентифицировать и зафиксировать проблемы, в решении которых заинтересованы пользователи, и выявить продукты, которые будут способны решить эти проблемы. Немаловажное значение имеет описание контекста, в котором будут использоваться продукты. Вторая фаза – ***определение границ предметной области***, третья фаза – ***определение границ решений***. На второй фазе рассматривается множество проблем, на третьей – множество решений, т.е. того, как именно будут решаться проблемы, описанные во второй фазе.

Второй подэтап – ***дизайн линейки продуктов***. Здесь решается главный вопрос – как именно фабрика будет производить продукты. Первая фаза – ***разработка архитектуры*** – проводится поиск необходимых архитектурных шаблонов; рассматриваются вопросы о возможности создания специальных инструментов, таких как каркас и предметно-ориентированный язык для использования абстракций, представленных каркасом. Вторая фаза – ***отображение требований***. На этой фазе проводится отображение изменчивости требований на изменчивость производственных активов, анализируется зависимость между изменчивыми элементами. Третья фаза – ***определение и автоматизация процесса***.

Третий подэтап – ***реализация линейки продуктов***. Первая фаза – ***обеспечение активами***. На данной фазе строятся или приобретаются необходимые активы. Вторая фаза – ***пакетирование активов***. Здесь все производственные активы упаковываются в шаблон фабрики программного обеспечения.

Итак, на этапе разработки линейки продуктов получен ***шаблон фабрики программного обеспечения***. Будучи установленным в интегрированную среду разработки, данный шаблон становится фабрикой разработки программ.

Второй этап – ***разработка продуктов***. В роли разработчиков может выступать компания, разработавшая фабрику программного обеспечения. С тем же успехом она может быть использована и сторонними разработчиками, которые приобрели данную фабрику. Разработка продукта включает ***спецификацию продукта***, когда требования выражаются в терминах линейки продуктов с отображением их на производственные активы. В этом случае внимание уделяется не всему приложению, как при традиционном способе разработки, а отдельным частям уже готового приложения. О разработке с использованием фабрики программного обеспечения вкратце будет рассказано в следующем разделе.

## Фабрика разработки Smart Client Software Factory

Чтобы почувствовать, что такое фабрика программного обеспечения, как с ее помощью разрабатывать продукты, здесь приводится пример одной широко используемой благодаря своей универсальности фабрики разработки композитных приложений. Коротко описан процесс разработки с ее помощью приложений.

Композитное приложение – клиентское приложение, состоящее из независимых функциональных элементов. Это могут быть Web-сервисы, другие приложения или определенные их части.

Фабрика *Smart Client Software Factory* содержит в себе опыт создания клиентских приложений на платформе .NET. Она включает различные ***повторно используемые компоненты***, наборы команд, выполняющие рутинные задачи, шаблоны, мастера настроек и другие полезные инструменты разработки.

Разрабатываемые с помощью фабрики приложения получаются легкоразвертываемыми и конфигурируемыми, обладают многофункциональным интерфейсом пользователя. Главное преимущество состоит в том, что в рамках фабрики могут трудиться несколько человек или команд в зависимости от масштаба проекта. Части, разработанные отдельными разработчиками, затем легко интегрируются в единое приложение.

Теперь перейдем к техническим деталям. Во-первых, *Smart Client Software Factory* запускается в рамках интегрированной среды разработки *Microsoft Visual Studio* благодаря технологии GAX / GAT (*Guidance Automation Extensions / Guidance Automation Toolkit*). С помощью технологии GAT фабрики разрабатываются, а с помощью технологии GAX они запускаются в среде разработки. Архитектура *Smart Client* позволяет создавать приложения с графическими интерфейсами по технологии *WinForms*. Основная идея концепции *Smart Client* – возможность продолжения работы клиентского приложения при условии его отсоединения от сети и первичного источника данных, что весьма полезно для композитных приложений. Все это возможно благодаря учету в созданных с помощью фабрики приложени-

ях таких моментов, как кэширование и синхронизация с первичным источником данных. Наряду с автоматическим решением проблем сохранения целостности данных, решаются вопросы безопасности, прежде всего связанные с доступом к хранилищу данных. В фабрику заложен механизм работы со следующими шаблонами: MVP (*Model-View-Presenter*), *Container Hierarchy*, *View Navigation*, *Workspace Hierarchy*, *UI Threading*, *Module Plug-in*, *Module Interface Separation*, *ActionCatalog*, *Service Locator*, *WorkItem Identity Publication*. Полное описание этих шаблонов, включающее такие пункты, как проблема, решение и примеры, можно найти в документации к данной фабрике.

Итак, разработка начинается с создания проекта. Выбирается шаблон проекта *Smart Client Application*. В это время активизируется мастер, позволяющий настроить параметры проекта, такие как имя, расположение на жестком диске самого проекта и библиотеки повторно-используемых элементов фабрики, задание пространства имен приложения, плюс возможность выбора некоторых опциональных возможностей, связанных с предпочтениями разработчика. После выполнения этих операций перед разработчиком имеется предварительно настроенный на детализацию набор проектов единого проектного решения. В его распоряжении есть четыре проекта. Первый – *Infrastructure module interface*. Он представляет собой интерфейс для проекта *Infrastructure module implementation*, содержит набор интерфейсов, доступных другим частям проектного решения. Среди них сигнатуры классов общего назначения и некоторых бизнес-сущностей, передающихся между частями системы, служб, команд и констант. Второй – *Infrastructure module library* – содержит реализации общих компонентов, используемых частями композитного приложения. Третий – *Infrastructure module implenemtation*, который выступает в роли контейнера для воплощения программных элементов. Большая часть работы происходит именно в этом проекте. Четвертый – *Shell project*, он содержит заготовку графического интерфейса клиентского приложения. Разработчик вправе создавать дополнительные проекты в рамках общего проектного решения.

Теперь разработчик может делать остальную работу. Модифицировать графический интерфейс по своему усмотрению, удалять ненужные ему элементы, добавлять новые и адаптировать их к приложению. Создавать рабочие модули (*business modules*), содержащие логику работы приложения. Добавлять к общему решению проекты базовых модулей (*foundation modules*), предоставляющих услуги частям системы. Добавлять ссылки на удаленные Web-сервисы.

Далее приводятся примеры других фабрик программного обеспечения, использующихся на практике.

- *Web Client Software Factory* – применяется для создания композитных Web-приложений.
- *Web Service Software Factory* – позволяет быстро создать Web-сервис.
- *Mobile Client Software Factory* – подходит для разработки композитных приложений для портативных компьютеров.

В следующем разделе представлено описание проекта фабрики разработки обучающих игровых программных систем. Разработка ведется в рамках научно-исследовательской работы автора, посвященной вопросам проектирования подобных систем, где основной акцент ставится на планирование систематического повторного использования.

## Фабрика разработки обучающих игровых программных систем

В этом разделе приводятся ответы на главные вопросы, которые стоят в последнем проекте автора, связанном с разработкой фабрики программного обеспечения.

**Вопрос № 1.** Кто будет использовать фабрику, а кто потреблять продукты, выработанные ею?

**Ответ.** Фабрику разработки обучающих игровых программных систем лингвистической направленности будут использовать небольшие компании-разработчики программного обеспечения, нацеленные на рынок электронного обучения, решившие занять на нем определенное положение. Это также могут быть специалисты в сфере образования, обладающие знаниями разработки компьютерных приложений и желающие создать себе в помощь продукт, способный решать определенные педагогические задачи, стоящие в процессе обучения ими своих подопечных. Продукты, выработанные фабрикой, будут использоваться в учебных заведениях и местах, где есть желающие пройти обучение. Основная задача продукта – способность поддерживать процесс обучения таким образом, чтобы обучающийся не ощущал разницы между обучением, будь он в специальном учебном заведении или дома, т.е. необходима эффективная система контроля и помощи со стороны программной системы.

Целесообразность использования фабрики заключается в том, что она позволяет строить приложения, отвечающие требованиям множества заказчиков. Нет необходимости строить излишне перегруженную функциями программную систему обучения, которая может лишь отпугнуть потенциальных заказчиков. А вот легкое в освоении приложение с минимальной функциональностью, но выполняющее именно то, что необходимо конкретному заказчику, как раз то, что нужно. Экономия средств, в данном случае, достигается за счет применения повторно используемых элементов для разработки приложений разным заказчикам со своим набором требований, большая часть которых совпадает, а вариации невелики, но обязательны для учета.

**Вопрос № 2.** Каковы необходимые знания разработчиков, использующих фабрику?

**Ответ.** Во-первых, это знание процессов, протекающих в ходе учебного процесса, а также то, в какой степени продукты фабрики могут их поддерживать. Это нужно для того, чтобы была возможность оценить объем работы на создание элементов поддержки процессов, не предусмотренных фабрикой. Во-вторых, необходимы знания того, как можно реализовать требования заказчиков, понимание, какие инструменты разработки для этого доступны.

Определить возможности продукта можно ознакомившись со ***схемой фабрики программного обеспечения***. Схема – это, прежде всего, способ упорядочивания элементов, использующихся в процессе разработки, иначе говоря, артефактов разработки, таких как модели, файлы исходного кода, манифесты развертывания и т.д. Говоря образно, их упорядочивание происходит в сетке, где каждый элемент, или ячейка, представляет собой определенную точку зрения, или аспект программного обеспечения. Вначале определяются инструменты для построения продуктов, т.е. языки, каркасы, шаблоны, а уже затем каждой ячейке сопоставляются реальные, пригодные для использования артефакты разработки. Один и тот же инструмент разработки может быть сопоставлен с несколькими ячейками, т.е. например, для работы с несколькими аспектами можно использовать один предметно-ориентированный язык. В конечном счете сетка преобразуется в схему, где между ячейками уже существуют определенные связи и наложены ограничения. Схема – это также шаблон для описания продуктов фабрики. В работе [1] приводится удачная аналогия, схема – рецепт, точки зрения – ингредиенты (а также инструменты и процессы приготовления).

**Вопрос № 3.** Каковы существенные составляющие схемы фабрики в общих чертах?

**Ответ.** Первая категория составляющих – требования. Список функциональных требований весьма широк, он делится на категории систем, из которых состоит продукт обучения, получаемый с помощью фабрики. Итак, есть требования к функциональности следующих систем: управления пользователями, обучения, обслуживания, взаимодействия человек-машина, безопасности, коммуникации, учебного материала. В рамках перечисленных систем происходят определенные процессы, которые можно конфигурировать. Нефункциональные требования представлены следующими требованиями. *Первая группа* – требования к удобству, решают вопросы того, насколько быстро обучаемые могут освоить систему, эффективно ли организовано взаимодействие с системой, не препятствуют ли определенные элементы организации интерфейса продуктивной работе и т.д. *Вторая группа* – требования к надежности систем. Решают, какие возможные дефекты системы являются терпимыми, а какие крайне нежелательными, препятствующими продуктивному процессу обучения. Содержит требования к нормальному функционированию, защите от сбоев и безопасности. *Третья группа* – требования к производительности, учитывающие время выполнения определенных операций, пропускную способность и эффективность потребления ресурсов вычислительной машины. *Четвертая группа* – требования к сопровождаемости, включают в себя требования к легкости исправления дефектов, модификации, замены, переносимости на другие платформы.

Вторая категория – общая архитектура продуктов и инфраструктура развертывания. Отвечает на следующие вопросы: из каких элементов состоит, какой стиль выбран, какие шаблоны предполагает, как проводится настройка продукта? Последний вопрос связан с таким аспектом, как особенность конкретной платформы, которая накладывает определенные ограничения на развертывание продуктов.

Третья категория – исполняемые артефакты разработки. Сюда входят компоненты, каркасы, конфигурационные файлы, отдельные классы и т.д., отсортированные по принадлежности к конкретной подсистеме (обучения, обслуживания и т.д.). Подробнее о составляющих подсистем можно узнать из работы [8].

Между составляющими схемы могут существовать различные отношения, которые необходимо учитывать и поддерживать их согласованность, чтобы избежать проблем, связанных с перекрытием информации между артефактами и их переименованием. Для разработки конкретного продукта необходимо сконфигурировать схему, которая должна содержать помимо фиксированных частей части изменяемые. Схема фабрики – это простое описание активов, используемых для построения. Реализацией схемы является шаблон фабрики программного обеспечения.

**Вопрос № 4.** Из каких инструментов состоит шаблон фабрики?

**Ответ.** Далее перечисляются инструменты и их описание.

***Предметно-ориентированные языки.*** В рамках разрабатываемой фабрики их несколько для каждой из подсистем, решающей определенные аспекты, к примеру, язык для установления отношений между составляющими подсистем. Предполагается создание графических и текстовых языков.

***Каркасы.*** Для каждой системы в фабрике представлен свой каркас, содержащий высокоуровневые абстрактные сущности, поддающиеся детализации и переопределению.

***Шаблоны.*** Главным образом это шаблоны, реализующиеся в каркасе. В большинстве своем это поддержка классических шаблонов проектирования, представленных в работе [21].

Упакованные вместе эти активы могут использоваться для разработки продуктов в интегрированной среде разработки. Шаблон, так же как и схему, необходимо сконфигурировать перед использованием.

О побуждающих мотивах построения фабрики обучающих игровых программных систем лингвистической направленности и предшествующих этому проектах можно узнать из работы [9].

## Заключение

В статье была представлена информация о процессе разработки программного обеспечения, сфокусированном на систематическом повторном использовании производственных средств и инструментов для их адаптации под конкретный продукт в рамках семейства программных систем.

Резюмируя содержание, можно выделить следующие положения.

- Сегодня от профессионального разработчика требуются знания в определенной предметной области или областях.
- Некоторые области предъявляют особые требования к психологическим характеристикам разработчика.

• Основными средствами разработки являются: модели, каркасы, предметно-ориентированные языки, шаблоны, компоненты.

• Перечисленные элементы – плод развития дисциплины разработки программного обеспечения.

• Инженерия предметной области имеет прямое отношение к вопросу повторного использования.

• Систематическое повторное использование неразрывно связано с введением практики разработки семейства программных систем.

• Линейки программных продуктов реализуют семейство программных систем, применяя имеющиеся производственные активы в рамках организационного процесса.

• Разработка в рамках фабрики программного обеспечения происходит в два этапа: разработка линейки продуктов, разработка продуктов.

• Сегодня имеются фабрики разработки для общих типов приложений, необходимы фабрики узкоспециализированные.

Автор статьи полагает, что представленной в данной работе информации достаточно для того, чтобы начать знакомство с подходами, способствующими развитию зрелости разработчиков в индустрии создания и производства программного обеспечения. А представленный список литературы и указание на источники в соответствующих разделах помогут получить интересующимся необходимые знания по определенному вопросу.

## СПИСОК ЛИТЕРАТУРЫ


1. **Гринфилд Д., Шорт К., Кент С., Крупи Д.** Фабрики разработки программ: потоковая сборка типовых приложений, моделирование, структуры и инструменты. М.: Вильямс, 2007. 592 с.

2. **Чарнецки К., Айзенекер У.** Порождающее программирование: методы, инструменты, применение. Для профессионалов. СПб.: Питер, 2005. 731 с.

3. **Липаев В.В.** Человеческие факторы в программной инженерии: рекомендации и требования к профессиональной квалификации специалистов. Учебник. М.: СИНТЕГ, 2009. 328 с.

4. **Макконнелл С.** Профессиональная разработка программного обеспечения. СПб.: Символ-Плюс, 2006. 240 с.

5. **Басс Л., Клементс П., Кацман Р.** Архитектура программного обеспечения на практике. 2-е издание. СПб.: Питер, 2006. 575 с.

6. **Брукс Ф.** Мифический человеко-месяц, или как создаются программные системы. СПб.: Символ-Плюс, 2001. 304 с.

7. **Lenz G., Wienands C.** Practical software factories in .NET. USA: Apress, 2006. 240 p.

8. **Гусс С.В.** Элементы повторного использования для программных систем учебного назначения. Концептуальное проектирование и анализ // Математические структуры и моделирование. 2009. Вып. 20. С. 78–92.

9. **Гусс С.В.** Фабрика разработки игровых лингвистических программных систем учебного назначения // Технологии Microsoft в теории и практике программирования. Мат. конф. / под ред. проф. Гергеля В.П. Нижний Новгород: Изд-во Нижегородского госуниверситета, 2010. С. 110–112.

10. **Гусс С.В.** Модель каркаса программных компонентов поддержки занятий лингвистической направленности в игровой форме // Прикладная информатика. 2010. Вып. 3 (27). С. 62–77.

11. **Липков А.И.** Ящик Пандоры: Феномен компьютерных игр в мире и в России. М.: Издательство ЛКИ, 2008. 192 с.

12. **Керделлан К., Грезийон Г.** Дети процессора: Как интернет и видеоигры формируют завтрашних взрослых. Екатеринбург: У-Фактория, 2006. 272 с.

13. **Роллингз Э., Моррис Д.** Проектирование и архитектура игр.: Пер. с англ. М.: Вильямс, 2006. 1040 с.

14. **Рекомендации** по преподаванию программной инженерии и информатики в университетах = Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering; Computing Curricula 2001: Computer Science: пер. с англ. М.: ИНТУИТ.РУ "Интернет-Университет Информационных Технологий", 2007. 462 с.

15. **Sutcliffe A.** The Domain Theory: Patterns for Knowledge and Software Reuse. USA: CRC Press, 2002. 424 p.

16. **Брауде Э.** Технология разработки программного обеспечения. СПб.: Питер, 2004. 655 с.

17. **Липаев В.В.** Программная инженерия. Методологические основы. М.: ТЕИС, 2006. 608 с.

18. **Арлоу Д., Нейштадт А.** UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, второе издание. СПб.: Символ-Плюс, 2007. 624 с.

19. **Липаев В.В.** Технико-экономическое обоснование проектов сложных программных средств. М.: СИНТЕГ, 2004. 284 с.

20. **Duffy D.** Domain Architectures: Models and Architectures for UML Applications. USA: Wiley, 2004. 406 p.

21. **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2008. 366 с.

22. **Cook S., Jones G., Kent S., Wills A.** Domain-Specific Development with Visual Studio DSL Tools. USA: Addison Wesley Longman, Inc., 2007. 563 p.

23. **Rahien A.** DSLs in Boo. Domain-Specific Languages in .NET. USA: Manning Publications Co., 2010. 352 p.

24. **McIlroy M.** Mass Produced Software Components // Rep. on a conference sponsored by the NATO Science Committee, Garmisch, Germany. 1968. 7–11 October. Scientific Affairs Division, NATO. Brussels, 1969. URL: http://www.cs.dartmouth.edu/~doug/components.txt (Дата обращения: 31.08.2010).

25. **Mili H., Mili A., Yacoub S., Addy E.** Reuse-Based Software Engineering. Techniques, Organization, and Controls. USA: John Wiley & Sons, Inc., 2002. 682 p.

26. **Parnas D.L.** On the Criteria to be Used in Decomposing Systems into Modules / Communications of the ACM. 1972. 15 December. URL: www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf (Дата обращения: 31.08.2010).

27. **Parnas D.L.** Active Design Reviews: Principles and Practices / Proc. of the 8th International Conference on Software Engineering, London. 1985. August. URL: http://www.cs.virginia.edu/ ~wh5a/personal/Quals/misc/ParnasPaper%20on%20stanley/ActiveDesign Reviews.pdf (Дата обращения: 31.08.2010).

28. **Wang A., Qian K.** Component-Oriented Programming. USA: Wiley-Interscience, 2005. 336 p.

29. **Mellor S., Scott K., Uhl A., Weise D.** MDA Distilled: Principles of Model-Driven Architecture. USA: Addison Wesley, 2004. 176 p.

30. **Коплиен Дж.** Мультипарадигменное проектирование для C++. Библиотека программиста. СПб.: Питер, 2005. 235 с.

31. **Parnas D.L.** On the Design and Development of Program Families / IEEE Transactions on Software Engineering. 1976. Vol. SE-2. No. 1. URL: http://web.cecs.pdx.edu/~omse532/Parnas_Families.pdf (Дата обращения: 31.08.2010).

32. **Parnas D.L.** Software Product Lines: What To Do When Enumeration Won't Work / Proc. of the First International Workshop on Variability Modelling of Software-Intensive Systems. 2007. 16–18 January. 2007. URL: http://ulir.ul.ie/ bitstream/10344/160/2/09SQRL1045.pdf (Дата обращения: 31.08.2010).

33. **Kang K., Sugumaran V., Park S.** Applied Software Product Line Engineering. USA: CRC Press, 2010. 563 p.

УДК 681.3.06+519.68

**В.М. Баканов,** д-р техн. наук, проф., Московский государственный университет приборостроения и информатики – МГУПИ

E-mail: e881e@mail.ru

# Априорная количественная оценка эффективности параллельных программ на конкретных многопроцессорных системах

*Обсуждается проблема эффективного использования ресурсов многопроцессорных вычислительных систем (МВС) архитектуры MPP (Massively Parallel Processing) для расчетов по конкретным параллельным алгоритмам. Предлагается интегральный показатель эффективности выполнения данной параллельной программы на конкретной МВС, учитывающий как аппаратные показатели системы, так и качество распараллеливания программы.*

***Ключевые слова:*** *параллельное программирование, архитектура MPP, сетевые технологии, латентность, гранулярность параллельной программы, технология MPI, эффективность распараллеливания*

Для решения проблем государственного и прикладного значения (от задач численного моделирования ядерных взрывов, молекулярных и генетических исследований до оптимизации параметров спортивного инвентаря) важным инструментарием является применение суперкомпьютерных систем, подкрепленное соответствующим математическим и программным обеспечением.

Способ повышения вычислительных мощностей путем параллелизации вычислений на многих одновременно выполняющих программу процессорах (*вычислительных узлах*, ВУ), в противовес повышения мощностей единого процессора, в настоящее время считается наиболее перспективным [1, 2]. Однако процесс разработки параллельных программ существенно более сложен, чем традиционно-последовательных программ, особенно в области оценки эффективности (в данном случае – соответствия программы параметрам МВС, прямым следствием чего являются возможность максимального использования ресурсов МВС и снижение времени решения задачи). Проблемы разработки эффективных (максимально использующих архитектурные особенности МВС) параллельных программ относятся к области *программной инженерии*.

МВС физически представляют собой множество (обычно классической фон-Неймановской архитектуры) ВУ, объединенных по возможности максимально быстродействующей и низкоинерционной компьютерной сетью. Межузловая сеть в данном случае служит аналогом общей шины в традиционных архитектурах ЭВМ, однако количественные ограничения на производительность объединяющей ВУ сети требуют особой организации вычислений – логического разбиения алгоритма на ортогональные (независимые) по отношению к обрабатываемым данным блоки (*гранулы, зерна параллелизма*) с минимизацией числа и объемов обменов между ВУ (чего практически никогда не требуется при классической общей шине вследствие ее огромной производительности и малой инерционности).

Большей популярностью пользуются МВС архитектуры MPP вследствие их значительной масштабируемости и относительно несложной реализации в виде *вычислительных кластеров* (ВК).

В ВК узлы физически *близко связаны*, для систем метакомпьютинга допускается расположение ВУ на значительных расстояниях (причем узлы зачастую гетерогенны). Соответственно, приемы параллельного программирования для ВК и метакомпьютерных систем количественно разнятся – относительно большое

время обмена данных в системах метакомпьютинга требует реализации значительно бóльшего размера *гранулы параллелизма* (последовательности инструкций процессора, не требующих при выполнении обменов данными с соседними процессорами) для избежания чрезмерных потерь на простой процессоров во время обмена данными, что катастрофически влияет на время решения задачи.

Первые ВК использовали для обмена данными сети со скоростью 10 и 100 Мбит/с (обычно *Ethernet*), в начале XXI века начался переход на гигабитовую, а сейчас и 10–100 гигабитовые технологии. Значительная величина латентности (инерционности) 100/1000 Мбит/с *Ethernet*-сетей (эта сетевая технология априорно разрабатывалась для некритичных к латентности применений) привела к тому, что в настоящее время значительные по мощности системы используют специализированные быстродействующие низколатентные сетевые технологии.

Для априорной оценки производительности вновь создаваемой МВС необходима информация о трех ее параметрах, причем определение каждого является отнюдь нетривиальной задачей (в качестве постановочного варианта приведем работу [1]):

- производительность единичного ВУ;
- параметры коммуникационной среды (сети, позволяющей обмениваться данными между ВУ);
- класс решаемых задач и конкретные технологии распараллеливания алгоритмов.

Как сказано, гранулой параллелизма обычно называют участок исполняемого кода, выполняющийся на отдельном процессоре и не требующий при исполнении данных от других процессоров (полужирные горизонтальные линии на рис. 1).

Привычно стало употреблять термины "*мелкозернистый*" и "*крупнозернистый*" *параллелизм* (*fine-grained parallelism* и *coarse-grained parallelism* соответственно), однако количественно определить принадлежность конкретного случая выполнения программы на данной МВС к одному из подобных определений не так просто. Пусть среднее время выполнения гранулы равно 0,1 с – это мелко- или крупнозернистый параллелизм? Ответ зависит от параметров *коммуникационной сети* (КС), обеспечивающей обмен данными между ВС.

В большинстве случаев невозможно полностью совместить обмен данными с вычислениями внутри гранулы (даже при использовании асинхронных обменов гранула-приемник принципиально не может начать работу, пока гранула-передатчик не перешлет ей данные и эти данные не будут получены адресатом). Поэтому рационально определить время обмена данными $t_{обм}$ (в общем случае $t_{обм}$ есть функция типа обмена и размера передаваемых данных) и время выполнения (*временная характеристика размера гранулы*) собственно гранулы параллелизма $t_{гран}$ (рис. 1, *б*, *в*); при этом принята парадигма "1 гранула = 1 операция обмена" (что больше соответствует случаю рис. 1, *в*, чем максимально упрощенному варианту рис. 1, *б*).

В качестве *показателя гранулярности параллельной программы* целесообразно использовать отношение $t_{гран}/t_{обм}$, определяющее отношение времени выполнения вычислений по отношению ко времени обмена данными; можно применить и зависимость $(t_{обм} + t_{гран})/t_{обм}$. Так как при выполнении реальной программы число гранул велико (каждую можно характеризовать парой $t_{обм}$ и $t_{гран}$), следует использовать статистическую оценку этой величины, некоторым образом усредненную по всем парам $t_{обм}$ и $t_{гран}$.

Попытки реализовать $t_{обм} \to 0$ представляют собой в основном аппаратную задачу (фактически переход к более быстродействующим и низколатентным сетевым технологиям). В то же время повысить размер гранулы параллелизма $t_{гран}$ – задача чисто алгоритмическая (даже при неизменной производительности аппаратной части). Повышение $t_{гран}$ достигается именно на этапе распараллеливания имеющегося алгоритма решения задачи; например, для каждого из алгоритмов умножения матриц – классического, по Винóграду или Штрассену – можно построить несколько конкретных алгоритмов распараллеливания с сильно различающимися значениями $t_{гран}$ и $t_{обм}$.

Из соображений повышения суммарного быстродействия МВС целесообразно всемерно увеличивать отношение $t_{гран}/t_{обм}$, которое можно назвать *коэффициентом гранулированности параллельной программы* (а фактически – *показателем эффективности выполнения данной параллельной программы на конкретной МВС*). Таким образом, с помощью этого коэффициента можно априори оценить, насколько пригодна (эффективна) данная МВС для решения задачи или же наоборот – определить круг задач, которые могут быть эффективно решенными на данной МВС, а также

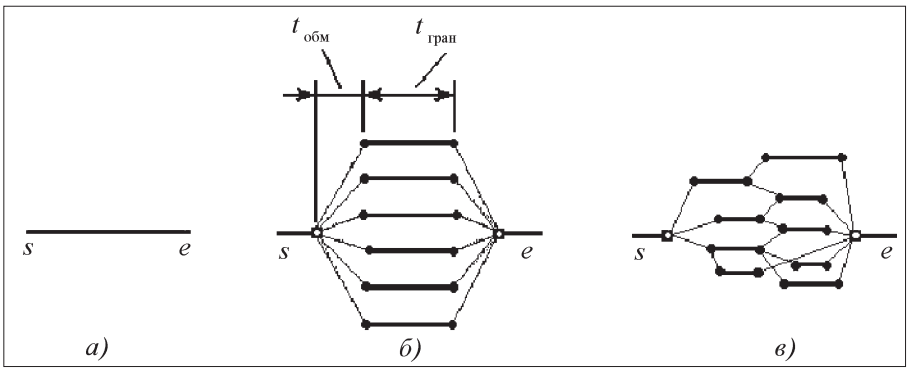


**Рис. 1. Диаграммы выполнения процессов при последовательном вычислении (*а*), при близком к идеальному распараллеливании (*б*) и в общем случае распараллеливания (*в*); *s* и *e* – моменты начала и конца процесса вычислений соответственно**

принять решение о модернизации МВС или параллельной программы.

При $t_{гран}/t_{обм} \le 1$ можно говорить о *мелкозернистом*, а при $t_{гран}/t_{обм} >> 1$ – о *крупнозернистом параллелизме*. Преимуществом данного подхода является потенциальная априорная определяемость величин $t_{обм}$ и $t_{гран}$.

В случае физически близкого расположения ВУ значение $t_{обм}$ обычно составляет $10^{-5}$–$10^{-2}$ с, при этом рациональной величиной $t_{гран}$ будет не менее $10^{-3}$–$10^{-1}$ с; в то же время для GRID-систем с $t_{обм}$ порядка единиц секунд следует осуществлять распараллеливание с $t_{гран} = 10$–$10^2$ с и более. Распараллеливание имеющегося алгоритма при требовании обеспечения максимальной гранулярности затруднено во всех отношениях, поэтому целесообразно ограничить величину $t_{гран}$ значением (10...100) $t_{обм}$ (что соответствует уровню от "отлично" до "хорошо", при этом потери производительности вследствие ожидания данных не превысят 1...10 % соответственно).

Число операций в гранулах и размер сообщений известны (или могут быть несложным образом оценены) для большинства стандартных параллельных алгоритмов, а зависимости $t_{обм}$ от размера сообщения и зависимость $t_{гран}$ от числа операций должны быть определены экспериментально. Некоторые трудности могут возникнуть при использовании итерационных алгоритмов, где число итераций зависит от исходных данных и априорно неизвестно; в этом случае может быть определена длительность одной итерации и оценено разумное их количество. При невозможности априорных оценок приходится использовать технологию профилирования параллельной программы [2].

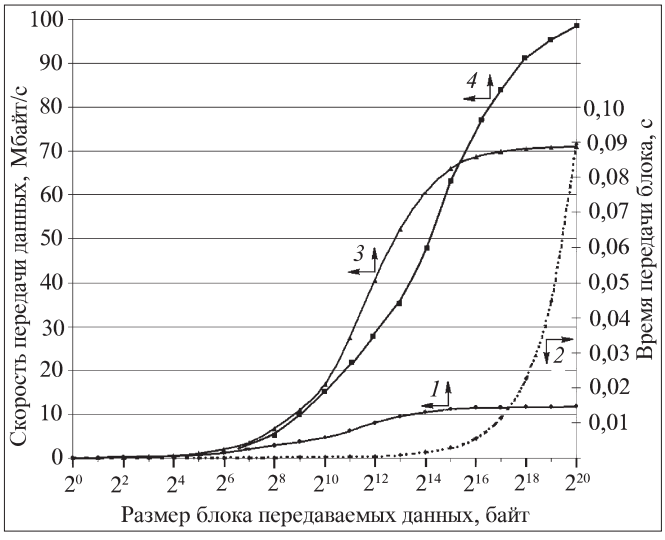


**Рис. 2. Скорость обмена данными по сети 100 Мбит/с (*1*), 1000 Мбит/с, MTU = 1500 байт (*3*), 1000 Мбит/с, MTU = 9000 байт (*4*) и время передачи блока (*2*) для 1000 Мбит/с сети в зависимости от размера блока передаваемых данных**

Количественный анализ влияния отношения $t_{гран}/t_{обм}$ на производительность МВС проводился на вычислительном кластере Лаборатории многопроцессорных вычислительных систем кафедры ИТ-4 МГУПИ. Кластер является МВС инструментального уровня и вследствие возможности динамического изменения технологии коммуникационной сети и числа задействованных ВУ назван *кластерным конструктором*.

Эмпирическая зависимость реальной скорости передачи данных (производительности сети) и времени обмена 100 и 1000 Мбит/с *Ethernet*-сетей от длины сообщений при обменах "точка–точка" приведены на рис. 2 (линейная логическая топология сети). При экспериментах изменялась также величина MTU (*Maximum Transmission Unit*, предельный размер пакета данных, который может быть передан по сети за один физический кадр с использованием протокола TCP/IP). Полученные результаты являются *программно-апаратными*, так как учитывают как время подготовки данных в системе "процессор–сетевой адаптер", так и программную часть (обработка библиотечных вызовов *MPI_Send/MPI_Recv*).

Эмпирическая зависимость (см. рис. 2) скорости обмена от размера сообщений соответствует формуле $G = \dfrac{X}{X/G_{max} + L}$ (где $G$ – скорость обмена; $X$ – длина сообщения; $G_{max}$ – максимальная производительность сети; $L$ – латентность), являющейся следствием известной [3] модели $T = \dfrac{X}{G_{max}} + L$ ($T$ – время передачи сообщения). Замеренное время обмена соответствует закону прямой пропорциональности от размера сообщения (в диапазоне $X = 10^2...10^{20}$ байт). Нет серьезных препятствий считать подобный характер зависимости нетипичным и для иных типов обменов и иных топологий сети (в некоторых случаях придется учитывать распространение сигнала последовательно через несколько ВУ).

Как видно из рис. 2, при переходе от 100 к 1000 Мбит/с производительность для заявленных в случае 1000 Мбит/с сети в 110–120 Мбайт/с достигается только при использовании так называемых *JUMBO FRAMES* (кадров с MTU = 9000 байт).

На сети *Ethernet* 100 Мбит/с аппаратно-программная латентность равна 40 мкс, цена обмена $4\times10^3$ бит, за это время указанный процессор проводит примерно $10^3$ плавающих пар операций "умножение/сложение" с числами двойной точности, для 1000 Мбит/с латентность 30 мкс, цена обмена $30\times10^3$ бит (значения латентности и быстродействия сети замерялись посредством модулей из набора тестов НИВЦ МГУ и Испытательной лаборатории проекта МВС ИПМ им. М.В. Келдыша РАН).

Проводя подобные описанным выше исследования каждой конкретной МВС, количественно определяем время $t_{обм}$. Вопросы, возникающие при определении $t_{обм}$ для более сложных случаев (например, при коллективных обменах или усложненной топологии сети) требуют чуть более тщательного анализа.

Выявление в программе блоков, самодостаточных по данным (блоков, которые могут выполняться независимо и параллельно) является одной из главных задач анализа *тонкой информационной структуры алгоритма* [2]. Для количественного анализа величин $t_{обм}$ и $t_{гран}$ на указанной МВС проводились эксперименты с различными параметрами КС и размерами обрабатываемых данных (при неизменном алгоритме как решения задачи, так и алгоритме собственно распараллеливания). Сравнение проводилось с помощью процедуры традиционного метода умножения матриц при ленточном (практически полностью соответствующем схеме рис. 1, *б*) алгоритме (рис. 3) распараллеливания для квадратных матриц разной размерности (элементы – вещественные числа двойной точности, стандартный GNU-компилятор языка C). При этом размер гранулы параллелизма (в арифметических операциях "умножение/сложение") равен $2N^3/P^2$, общее число гранул суть $N^2/P^2$, длина пересылаемых (неоптимизированная пересылка строк и столбцов матриц-сомножителей) от координирующего узла (КУ) к ВУ данных равна $2N^2/P$ и от ВУ к КУ – $N^2/P^2$, здесь $N$ – порядок матриц, $P$ – число ВУ.

На рис. 4 приведено изменение производительности при переходе коммуникационной *Ethernet*-сети от 100 Мбит/с к 1000 Мбит/с (неизменный алгоритм и параметры компилятора, порядок умножаемых матриц $N_1 = 800$, $N_2 = 1600$, $N_3 = 3200$). В данном случае показатель $t_{гран}/t_{обм}$ растет прямо пропорционально $N$ и обратно пропорционально числу процессоров $P$ (так как $(2N^3/P^2)/(2N^2/P + N^2/P^2) = 2N/(2P + 1) \cong N/P$, перевод во временны́е показатели изменит только постоянный коэффициент). Отношения $t_{гран}/t_{обм}$ для трех указанных размерностей матриц равны 23/46/91 соответственно для сети 1000 Мбит/с и числа процессоров $P = 10$ (для сети 100 Мбит/с при аналогичных условиях отношения $t_{гран}/t_{обм}$ равны 2,3/4,6/9,1).

Рис. 4 иллюстрирует явление *сетевой деградации* вычислений (основная причина снижения масштабируемости), которое увеличивается при уменьшении размера гранулы параллелизма (на рис. 4 *ускорение вычислений* – отношение времени вычислений на единичном процессоре ко времени вычислений на нескольких процессорах). При переходе от 100 к 1000 Мбит/с сети происходит как повышение суммарной производительности МВС, так и снижение величины сетевой деградации. Таким образом, показатель $t_{гран}/t_{обм}$ существенно влияет и на масштабируемость системы "параллельная программа + МВС" – с возрастанием показателя гранулярности кривые на рис. 4 все более приближаются к прямой идеального распараллеливания.

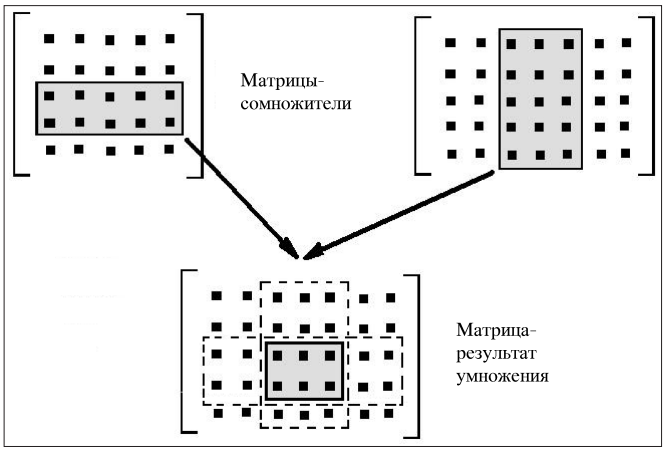


**Рис. 3. Схема параллельного умножения матриц ленточным методом**

На рис. 5 приведены экспериментальные данные *относительного ускорения вычислений* при переходе КС от 100 к 1000 Мбит/с ($k_{1000}/k_{100}$ – отношение ускорений вычислений на КС с параметрами 1000 и 100 Мбит/с соответственно). Рис. 5 доказывает, каким образом производительность повышается именно за счет увеличения пропускной способности сети и снижения латентности (режимы компиляции и параметры вычислительных узлов не изменялись). Максимальное увеличение производительности достигается при малой величине гранулы параллелизма ($t_{гран}$ мало́ и снижение $t_{обм}$ более значимо), при большем размере гранулы ($t_{гран}$ велико́ и снижение $t_{обм}$ менее значимо) повышение производительности незначительно.

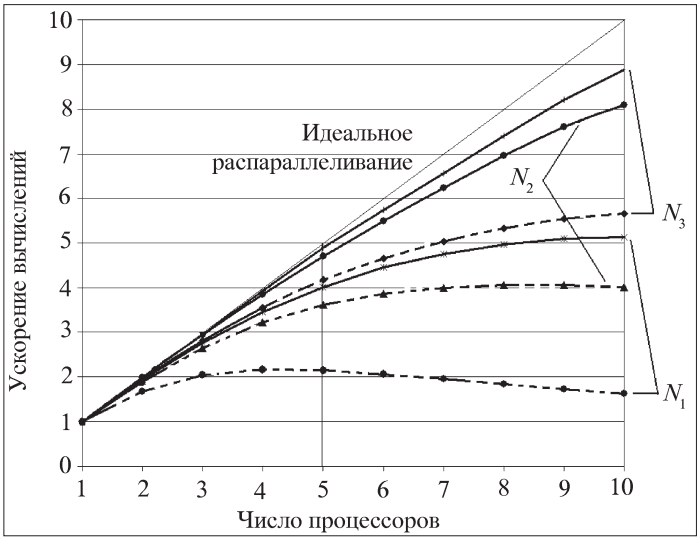


**Рис. 4. Влияние параметров коммуникационной сети на повышение производительности МВС (штриховые линии – КС быстродействием 100 Мбит/с, сплошные линии – КС 1000 Мбит/с *Ethernet*, $N_3 = 2N_2 = 4N_1 = 3200$)**

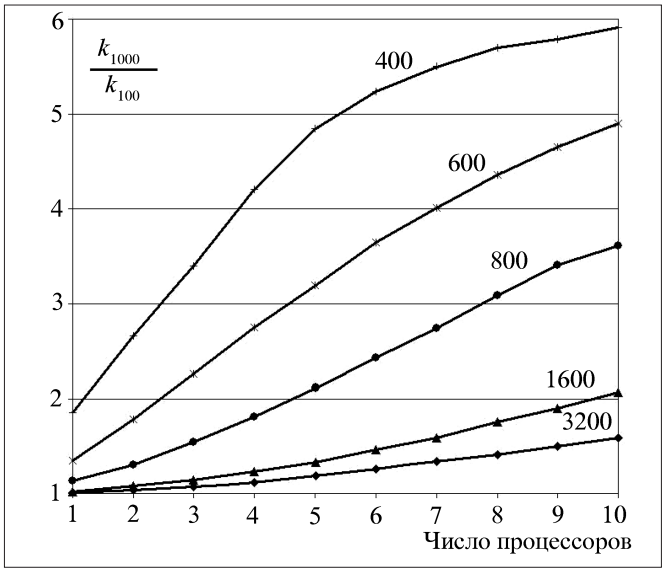


**Рис. 5. Относительное увеличение производительности МВС при переходе от 100 Мбит/с к 1000 Мбит/с *Ethernet*-сети (числа у кривых – размер перемножаемых матриц)**

В целом механизм модернизации МВС путем усовершенствования компьютерной сети не только повышает суммарную производительность даже без замены элементной базы вычислительных узлов, но и расширяет область выполняемых на данной МВС алгоритмов в сторону мелкогранулярных.

Таким образом, получены количественные данные зависимости производительности МВС от параметров коммуникационной среды и размера гранулы параллелизма. Данные являются основой для построения научно обоснованных рекомендаций по разработке новых МВС, модернизации существующих и выбора МВС для решения конкретных вычислительно-трудоемких задач.

## Заключение

1. Предложен и обоснован интегральный количественный показатель эффективности выполнения параллельных программ на конкретной МВС, учитывающий как аппаратные показатели МВС (*скорость обмена данными* между вычислительными узлами МВС и *производительность каждого узла*), так и качество распараллеливания алгоритма (размер *гранулы параллелизма*). Предложена количественная шкала совместимости (эффективности) конкретной параллельной задачи и данной МВС.

2. Для конкретного случая распараллеливания приведены количественные оценки показателя гранулярности и экспериментально подтверждено существенное влияние этого показателя на производительность и масштабируемость МВС.

## СПИСОК ЛИТЕРАТУРЫ

1. **Гергель В.П.** Теория и практика параллельных вычислений (учебное пособие). М: БИНОМ, 2007. 423 с.

2. **Воеводин В.В., Воеводин Вл.В.** Параллельные вычисления. СПб.: БХВ-Петербург, 2004. 608 с.

3. **Лацис А.О.** Как построить и использовать суперкомпьютер. М.: Бестселлер, 2003. 240 с.

УДК 004.4'2, 004.413, 378.14

**Е.Н. Павловский,** канд. физ.-мат. наук, науч. сотр.,
Институт дискретной математики и информатики (Новосибирский государственный университет), бизнес-аналитик, ООО "Новософт"

E-mail: euxsun@gmail.com

# К подготовке специалистов по программной инженерии

*Рассматривается использование методологии и инструментальных средств, применяемых в программной инженерии для организации процессов воспитания и обучения студентов по IT-специальностям. Предложено использовать модель зрелости процессов для образования, системы управления версиями, системы отслеживания ошибок, формальные онтологии, проектную методологию и коллективные формы изучения дисциплин.*

***Ключевые слова:*** *система управления версиями, система отслеживания ошибок, модель зрелости процессов разработки, формальные онтологии, управление проектами, мотивация студентов*

## Введение

Отрасль информационных технологий (IT) является одной из самых быстроразвивающихся. Высокими темпами обновляется не только программное и аппаратное обеспечение, но и технологии, а также методология решения промышленных задач с использованием положений программной инженерии. Соответственно этому должны меняться и подходы к подготовке кадров для отрасли. Программы образования IT-специалиста должны учитывать эту динамику развития и обновляться. В то же время они должны быть устроены так, чтобы студент, получив необходимый набор фундаментальных знаний, мог освоить и новые "веяния", и технологические навыки в индустрии разработки и использования программного обеспечения.

Представляется, что необходимо динамически изменять программу с привлечением работодателей от индустрии и научных учреждений. Работодатели дают технологию — то, что апробировано и используется в текущий момент времени, научные институты — перспективу — то, что нужно в будущем, то, на основе чего можно будет создавать *brainware* (интеллектуальный продукт) [1]. Чтобы правильно организовать процесс постоянного изменения образовательной программы, достаточно провести аналогию с настраиваемыми бизнес-процессами, для которых на рынке IT-продуктов и услуг уже существует множество решений. К сожалению, в настоящее время наше образование в области IT само не пользуется преимуществами IT.

Для построения процесса обновления программ необходимо давать возможность компаниям вести прямую и обратную связь с университетами. В Новосибирском государственном университете (НГУ), как и в некоторых других университетах, сложилась хорошая практика – сотрудники IT-фирм работают преподавателями. Фактически, они самым непосредственным образом влияют на процесс обучения, вносят в читаемые ими курсы самые последние новинки. Это так называемые практикующие преподаватели. И хотя студенты всегда их с интересом слушают, обнаруживается ряд связанных с этим педагогических и методических трудностей.

Во-первых, IT-специалист — это не всегда хороший методист. Он хорошо знает свою предметную область и может зажигательно о ней рассказывать. Это нравится студентам. Однако из этого не следует, что студенты обучаются соответствующему материалу эффективно. Во-вторых, IT-специалист работает в технологиях, которые применимы на текущий момент, и он хорошо знает их достоинства и недостатки. Вместе

с тем способен ли он абстрагироваться и "возвыситься" над текущими обстоятельствами, чтобы описать студентам перспективу? Вероятно, не всегда. Для этого необходимо глубоко понимать суть апробированных технологий. Необходимо осознавать научные идеи и подходы, даже те, которые не нашли широкого применения. Так, например, концепция семантического программирования [2], активно развивавшаяся в 1980-е годы в Институте математики Сибирского отделения Академии наук, только сейчас находит свое практическое применение в виде технологии программирования *Ontobox* [3] с языком запросов *Libretto*[1].

Еще одним достоинством описанной формы сотрудничества университета и IT-компаний является обратная связь. Компании имеют возможность определить еще на стадии обучения, какого студента они возьмут к себе в будущем.

Принимая во внимание отмеченные выше обстоятельства, хотелось бы перевести отношения университета и индустрии на следующий уровень зрелости. Методология зрелости процессов разработки программного продукта (*Capability Maturity Model* – CMM, [4]) уже давно применяется в программной инженерии. Она заключается в выделении уровней зрелости бизнес-процессов его разработки и описании некоторых ключевых практических навыков, помогающих перейти от уровня к уровню. Существующему уровню зрелости процессов образования, скорее, подходит название "повторяемый" (второй уровень в терминологии CMM). Следующий уровень — "определенный", когда четко определены бизнес-процессы, понятно, откуда идет управляющее воздействие, где и какой результат ожидается, и кто отвечает за каждый подпроцесс. Сейчас образовательный процесс разделен по функциональному признаку: здесь студента алгебре подучили, там — объектно-ориентированному программированию, а где-то — компьютерной графике. К сожалению, в этой последовательности нет интегрирующего механизма. При достаточно хорошей внутренней организации определяются зависимости между дисциплинами, что может дать студенту понимание связи между различными курсами. Однако в итоге отсутствует ясность, кто отвечает за формирование таких характеристик специалиста, как математическая культура, привычка к рефлексии, способность работать с источниками, умение принимать обоснованные инженерные решения.

В качестве такого интегрирующего инструмента образовательные стандарты третьего поколения [5] предлагают вместо традиционных "знаний, умений и навыков" использовать понятие "компетенции" и прослеживать наработку этих компетенций у студентов сквозным способом через освоение ими ряда курсов. Следуя методологии процессного подхода, необходимо делать и следующий шаг, а именно — вводить лиц, ответственных за весь бизнес-процесс и за отдельные подпроцессы (для формирования каждой компетенции).

[1] URL: http://ontobox.org/

Теперь рассмотрим применение методологии IT в образовании. Множество применяемых в программной инженерии инструментов можно использовать и в образовательном процессе — это тоже коллективный процесс, имеющий, однако, некоторые отличия. Рассмотрим далее их сходство и различие.

## 1. Система отслеживания ошибок[2] в лекциях

Кто еще помнит, как будучи студентом он сидел на университетской скамье и переписывал лекции с доски в тетрадь, тот знает, насколько этот метод передачи точных формулировок, текстов и прочей формальной информации ненадежен. Лектор иногда совершает неумышленные ошибки, что-то забывает записать, на чем-то не успевает подробно остановиться.

Здесь усматривается знакомый процесс "отлавливания ошибок", только не в коде (хотя записи в тетради с некоторой натяжкой можно считать кодом). Вместе с тем хотелось бы заметить, что сам процесс подачи материала с опечатками очень педагогичен (не случайный, а при специальном навыке лектора) — он заставляет студентов задуматься: "А что-то тут не так, надо исправить". Студент попадает в естественную (для него самого) проблемную ситуацию, выходом из которой является нахождение правильного понимания. Студент приходит к нужному понятию через ошибки и внутреннюю мыслительную работу. На самом деле, этот метод "проблемных ситуаций" (о нем далее) дает хорошие результаты не только в процессе понимания лекционного материала.

Таким образом, предлагается использовать какую-либо систему[3] для фиксации ошибок, опечаток, непонимания (и далее по категориям сообщений, например, "запрос функциональности") в лекциях. Здесь тестировщиками выступают студенты, желательно организованные в малые группы, а разработчиком (программистом) – лектор. Система отслеживания ошибок решает важную задачу коммуникации. Такая связь должна существовать между студентами и лектором в течение семестра постоянно.

Наряду с решением текущих задач усвоения материала студенты осваивают инструмент, который им пригодится в профессиональной деятельности. Более того, умение пользоваться данным инструментом совершенствуется при решении практических задач. Действительно, необходимость использования систе-

[2] **Система отслеживания ошибок** (англ. *bag tracking system*) – прикладная программа, разработанная с целью помочь разработчикам программного обеспечения (программистам, тестировщикам и др.) учитывать и контролировать ошибки (баги), найденные в программах, пожелания пользователей, а также следить за процессом устранения этих ошибок и выполнения или невыполнения пожеланий. (Википедия, URL: http://ru.wikipedia.org/wiki/Система_отслеживания_ошибок).

[3] В IT-индустрии широко распространены такие системы, как Jira, Bagzilla, Trac и др.

мы отслеживания ошибок появляется естественно, как удобный инструмент для решения текущих проблем студентов. Высокая мотивация к изучению и применению систем отслеживания ошибок при этом обеспечена.

## 2. Коллективная работа при разборе лекций

Работы по всем сколько-нибудь сложным IT-проектам организованы в коллективах. По этой причине важно давать студентам практику работы в команде. Хорошо, если на каких-то преподаваемых курсах работа построена на принципах разделения ролей, ротации исполнителей ролей, взаимной (горизонтальной) ответственности. Однако, как правило, наша система образования воспитывает индивидуалистов. Положение может усугубиться внедрением индивидуальных образовательных траекторий.

Часто при поступлении на работу у новоиспеченного специалиста возникают трудности организации своей работы внутри коллектива. Одним из важных факторов является уровень понимания разделения ролей в коллективе. Еще одним — достаточность и правильность коммуникаций для выработки общего понимания задач, а также навык и дисциплина обработки задач, планирования их выполнения и отчетности.

В рамках учебного процесса можно организовать работу в коллективе относительно учебных задач и возникающих при этом вопросов. Например, как было отмечено выше, при разборе лекций, при поиске в них ошибок, работу можно построить в малой группе. В ходе разбора лекций, сравнения конспектов, обсуждения тонких моментов ("а одинаково ли мы понимаем?") вырабатывается общее понимание, в том числе – за счет индивидуальной мыслительной работы каждого. Кроме того, обнаруживаются пробелы, несовпадения конспектов — кто-то пишет со слов лектора, а кто-то — по собственному пониманию.

При организации работы в малых группах перед студентами возникают трудности организации работы этой малой группы. Все как у старших товарищей: надо договориться о времени и месте совместной работы; подготовиться к этой работе; устранить в ходе работы мешающие ее должному выполнению факторы (отвлечения, прерывания и т.п.). Таким образом, у студентов в проблемной ситуации востребуются организаторские навыки, научить которым можно только тогда, когда человек с подобными вопросами уже сталкивался. Такой подход также создает мотивацию к эффективной работе, в отличие от работы в одиночестве, которая при отсутствии должной самодисциплины дает очень низкий эффект. Студент уже не может сказать себе: "Сегодня что-то не хочется разбирать лекции, завтра разберу", так как он договорился с товарищами на определенное время о том, что они поработают вместе. Взаимные обязательства и контроль "сбоку" делают процесс внеаудиторной работы студента более устойчивым к факторам студенческой среды. В то же время, студенты проходят практику работы в коллективе.

Для практики распределения задач и ролей можно предложить студентам определить, кто в малой группе отвечает за формулирование вопросов к лектору по результатам разбора лекций, кто отвечает за получение ответов. При этом каждый из членов малой группы несет ответственность не только перед самой группой, но и перед всеми остальными студентами.

Важной ролью, в которой должны попробовать себя все члены малой группы, является роль организатора. Ему необходимо назначать время и место собраний группы, обеспечивать работу ресурсами, следить за выполнением взятых обязательств. Для будущих управленцев это хороший способ получить необходимые качества: требовательность к участникам; умение планировать время, задачи; способность фиксировать текущее положение дел; принимать и исполнять решения.

## 3. Единый репозиторий всех работ студентов

В учебном процессе уместно использовать единый репозиторий всех работ студентов. Обязать студентов еще на первом курсе все свои работы размещать в какую-либо информационную систему управления версиями (например, SVN)[4] – значит поставить их в ситуацию, когда понять работу подобного рода систем просто необходимо. Причем это не требует от студента особых ментальных усилий. Важно, что навык работы с этой системой развивается в результате, скорее, практической, чем учебной деятельности, так как к обучению она относится опосредовано. Например, студент может выкладывать в систему реферат по какому-то отдельному предмету, и по мере написания добавлять его новые версии.

Еще одно преимущество такого подхода в том, что преподаватель всегда может увидеть – делал студент работу или нет. Можно отследить ход работы, отметить, сколько было версий документа, сколько правок. Кроме того, нерадивые студенты не смогут прибегать к таким аргументам, как "я забыл флешку", "у меня вирус все съел" и т.п. Таким образом, реализуется защита от вирусов, хранение различных версий документов, доступ к работам преподавателей.

Организация коллективной работы с исходными текстами – отдельная задача. Она должна решаться в рамках какого-либо курса в виде проектной работы над одним заданием.

---

[4] Система управления версиями – программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение и многое другое. SVN (сокр. от *SubVersion*) – одна из распространенных систем управления версиями, применяется, как правило, в разработке продуктов с открытым исходным кодом.

## 4. Проблемно-задачный подход

Методологическая основа проблемно-задачного подхода обсуждалась в работе [6]. В данном разделе речь пойдет о вопросах его применения к учебно-воспитательному процессу.

Нельзя рассматривать процесс образования сам по себе, как выпуск каких-то единиц продукции (специалистов). В этом процессе необходимо учитывать профессиональное и личностное самоопределения студентов. В этом смысле образовательный процесс не должен быть какой-то замкнутой внутри себя системой с его учебными заданиями, не совсем понятными студентам, еще не опробовавшим саму проблематику профессиональной деятельности. В советское время были распространены учебно-производственные комбинаты, решавшие задачу профессиональной ориентации абитуриентов. Хочется отметить, что в настоящее время эта традиция понемногу возрождается. Один из проблемных вопросов в процессе образования заключается в том, что многие учебные задания для студентов выглядят совершенно неестественными и непонятными. В таком случае студенты слабо понимают, что им понадобится завтра, а что — нет. Если при этом еще студент верит на слово преподавателю, то шансы, что получится воспитать специалиста с критическим мышлением, значительно уменьшаются.

Гарантию воспитания кадров, принимающих самостоятельные решения, специалистов с критическим мышлением может дать проблемно-задачный подход к обучению. Он заключается в помещении студентов в проблемные ситуации, в которых возникает необходимость осознанного выбора, поиска и принятия решений. Это дает высокую мотивацию студентам к изучению предмета, ставит их в позицию активного познающего. При таком подходе возрастают требования к преподавателям: студенты становятся реально заинтересованы в том, чтобы им доступно и "по делу" объяснили материал, не удаляясь далеко в пространные рассуждения, например, об общей пользе математики в IT.

В настоящее время студенты, как правило, рассматривают образовательный процесс как некоторую формализованную последовательность действий, а именно – нужно сдать такие-то задания, зачеты, экзамены. По этой причине они слабо мотивированы, обучение становится просто серией сдач экзаменов и зачетов. Очевидно, что образовательная проблема должна быть подчинена некоторой мотивирующей постановке проблемы из профессиональной области.

Источником правильных формулировок профессиональных задач может быть наука, инженерия. Как ставить такие задачи — отдельный довольно сложный вопрос, который выходит за рамки тематики данной статьи. В качестве вариантов можно предложить работать над образовательной проблемой как над профессиональной, в виде проектной деятельности (об этом далее). Кроме того, необходимо учитывать, что общение является важной составляющей жизни студентов, поэтому работа в коллективе над некоторой задачей имеет свои внутренние устойчивые самовоспроизводимые мотивы.

Модель проблемных ситуаций построена на исследовательском интересе студентов и развивает его. Когда студент сталкивается с такой ситуацией — он ставит перед собой вопрос: "Что я лично могу сделать для ее разрешения, что мне для этого нужно, какие знания, навыки и умения?" Только при таком подходе мы можем констатировать, что человек научился, буквально "научил себя". Данный подход востребует исследовательские способности студентов. Сталкиваясь с проблемной ситуацией, студент будет воспринимать ее как свою личную проблему. Проявляя исследовательские способности, он будет вынужден научиться формулировать задачу, направленную не ее разрешение.

Постановка задач является опорной точкой в организации самостоятельной работы студентов, которой так много места отводится образовательными стандартами нового поколения.

Наряду с самостоятельной работой, у студентов должна быть возможность задать назревшие вопросы и получить на них ответы. В случае хорошей организации методических комплексов с использованием, например, технологий *Semantic Web* – студент мог бы формулировать вопросы на языке OWL/DL[5], допускающем машинную обработку.

## 5. Онтологический инжиниринг – интегрирование знаний студентов в единую систему

Для стимулирования процессов интеграции знаний изучаемых дисциплин в единую систему можно предложить студентам построить диаграмму связей между изученными понятиями. Диаграмма может отражать порядок изложения, отношение, позволяющее определить одно понятие через другое. Построение таких обобщающих диаграмм полезно для студентов в формировании их целостного понимания изучаемой дисциплины, в определении ее места среди других предметов. Оно заставляет привести в порядок свои представления о предмете и вовремя скорректировать их, предоставив результаты построений преподавателю.

Такие диаграммы можно составлять сразу в формате OWL[6], что дает не только навык написания онтологий, но и возможность преподавателю воспользоваться трудами студентов для формирования формальной

[5] OWL/DL – фрагмент языка OWL, предназначенный для пользователей, которым нужна максимальная выразительность при сохранении полноты вычислений (все логические заключения, подразумеваемые той или иной онтологией, будут гарантированно вычислимыми) и разрешаемости (все вычисления завершатся за определенное время) (Википедия, URL: http://ru.wikipedia.org/wiki/Web_Ontology_Language).

[6] OWL (сокр. от Web Ontology Language) – XML-язык для представления формальных онтологий.

онтологии своей дисциплины. Последнее позволит в будущем согласовывать программы связанных между собой дисциплин (на основе используемых в них понятий).

Следует отметить, что, как правило, онтология изучаемой дисциплины очень хорошо поддается формализации. Наряду с иерархией понятий преподавателю рекомендуется использовать онтологии верхнего уровня, такие как "познание", "учебная дисциплина", "исследование". Данный подход был предложен автором в секционном докладе на конференции "Мальцевские чтения – 2010"[7], изложен в тезисах конференции "Преподавание ИТ в России 2010" [7].

## 6. Уважать – значит требовать

Еще одним важным моментом в применении подхода проблемных ситуаций является необходимость оказания высокой степени доверия студентам. Студенты должны иметь возможность решать практически значимые задачи профессионально. При этом нужно с должной строгостью спрашивать с них именно такой подход к решению. Основой уважения, как известно, является соответствие требованиям. Студенты заслуживают уважения к себе тем, что решают задачи в ходе образовательного процесса теми же методами и средствами, что и их старшие товарищи, работающие в IT-сфере профессионально, используя для этого различные инструменты коллективной работы, современные технологии и т.п.

Отталкиваясь от принципа подобия методов, можно заметить особенность сдачи IT-проектов. Вероятно, причина большого числа затянутых, не сдаваемых в срок проектов, кроется в привычке сдавать экзамен в конце семестра. Эта особенность традиционного образования существенно тормозит развитие IT-индустрии.

Теперь логично перейти к обсуждению еще одного инструмента профессиональной деятельности применительно к проблемно-задачному подходу.

## 7. Использование проектной методологии при решении задачи

Представляется целесообразным в рамках учебной деятельности предлагать студентам решение каждой учебной задачи, требующей существенных ресурсозатрат, оформлять в виде проекта. Ключевыми характеристиками каждого проекта являются бюджет, сроки и объем работы. Бюджетом для студентов может, например, выступает их учебное время. При таком подходе у студентов появится реальная потребность в знаниях из области проектного управления.

В качестве реальных проектов могут рассматриваться: изучение конкретной дисциплины, выполнение задания по курсу, изучение отдельного модуля. При предлагаемом подходе возникает необходимость оценить время на изучение предмета, определить требования заказчика (преподавателя), согласовать сроки сдачи проекта, спланировать работы по проекту. Это чрезвычайно полезная практика, ведь суть проектного подхода отрабатывается на понятной для студентов задаче.

Для студентов старших курсов, можно преподавать основы риск-менеджмента. Однако делать это можно только тогда, когда они уже познакомятся с реальными вопросами проектирования на практике.

Каждое новое знание должно быть обусловлено для обучаемого некоторой внутренней востребованностью, готовностью. Все изложенные выше идеи и подходы опираются на следующий принцип: новые знания должны отвечать на заданные "де-факто" решаемой задачей вопросы. При этом следует заметить, что традиционная система образования, как правило, заставляет отвечать на незаданные вопросы, чем не только демотивирует обучающихся, но и порождает множество бесполезных разрозненных сведений в голове студента.

## 8. Система обратных связей, контроля и управления образовательным процессом

Для реализации описанных выше подходов в Новосибирском государственном университете развернут инициативный проект по созданию "Системы обратных связей, контроля и управления образовательным процессом"[8]. Система состоит из организационных форм, методологий и подходов, где в качестве инфраструктуры используется соответствующая информационная система.

В систему закладываются следующие функции:

1. Управление задачами: возможность ставить себе задачи и отслеживать их выполнение; формы отчетности (для отчетов, прежде всего, перед собой), система напоминаний.

2. Коммуникации: система отслеживания ошибок в лекциях; форум.

3. Средства коллективной работы студентов (система контроля версий, гипертекстовая база знаний, пополняемая студентами и модерируемая преподавателями, полуформализованная с помощью OWL).

4. Аналитика: сбор статистических данных о выполнении заданий, проектов, прогнозирование проблемных ситуаций.

5. Социальная сеть: игровая форма рейтинга студентов; рейтинг преподавателей (по количеству и качеству ответов на вопросы студентов).

6. Доступ к учебным программам для студентов и работодателей.

7. Инструменты планирования учебных программ во времени (диаграммы Гантта).

Планируется организовать доступ работодателей к рейтингу и к результатам образования студента, что позволит им отбирать необходимые им кадры, а также

[7] Сайт конференции. URL: http://www.math.nsc.ru/ conference/ malmeet/10/index.html.

[8] Сайт проекта: http://trac6.assembla.com/education_and_ feedback.

участвовать в формировании индивидуальных или коллективных программ специального обучения (дополнительных курсов) целевым образом, где "нацеливание" проводится с помощью данных системы. Рейтинг, с другой стороны, подстегнет студентов к демонстрации наиболее достойных качеств и внесет дух здорового соревнования при должной объективизации базовых показателей.

## Заключение

При организации процессов обучения IT-специалистов с применением известных, зарекомендовавших себя технологий программной инженерии, предлагается:

– применять для лекций системы отслеживания ошибок, что способствует освоению одного из важнейших инструментов разработчика программного обеспечения, более глубокому пониманию лекций, реализации коммуникаций между преподавателем и студентом;

– использовать системы управления версиями для хранения всех работ студента во время учебы, параллельно отрабатывая навык управления версиями;

– организовывать изучение дисциплин в малых группах, давая студентам практику р47

аботы в коллективе с разделением ролей;

– применять методы постановки проблемных ситуаций для более эффективного вовлечения студентов в профессиональную деятельность и решения вопросов повышения их мотивации к учебной деятельности;

– использовать проектную методологию при организации изучения дисциплин, давая студентам практику работы в реальном проекте, имеющем свой бюджет, сроки и объем работы.

Хотелось бы надеяться, что эти идеи и подходы найдут понимание как среди специалистов программной инженерии, так и среди преподавателей, организаторов курсов, заведующих кафедрами и кураторов специализации.

*Автор выражает благодарность Д.Е. Пальчунову, В.В. Мухортову, В.А. Васенину, О.Е. Грачёву за полезные обсуждения, способствовавшие более точному изложению представленных в данной публикации подходов.*

## СПИСОК ЛИТЕРАТУРЫ

1. **Магистральный интерфейс** — сближение началось (беседа с академиком Ю.Л. Ершовым) // Наука в Сибири. 2002. № 42–43 от 01.11.2002. URL: http://www.sbras.ru/HBC/hbc.phtml?20+224+1.

2. **Гончаров С.С., Свириденко Д.И.** Σ-программирование // Логико-математические основы проблемы МОЗ. 1985. Вып. 107: Вычислительные системы. С. 3–29.

3. **Малых А.А., Манцивода А.В.** Онтобокс: онтологии для объектов // Известия ИГУ. 2009. № 2. С. 94–104.

4. **Паук М., Куртис Б., Хриссис М.Б.** Модель зрелости процессов разработки программного обеспечения / Пер. под ред. В. Рябикина. М.: Интерфейс-Пресс, 2002. URL: http://ryabikin.narod.ru/sw-cmm/index.htm.

5. **Федеральные** государственные образовательные стандарты / Министерство образования и науки РФ. URL: http://mon.gov.ru/pro/fgos/.

6. **Проблемно-ориентированный** подход к науке: Философия математики как концептуальный прагматизм / под. ред. В.В. Целищева, Ю.Л. Ершова и др. Новосибирск: Наука, 2001. 154 с.

7. **Павловский Е.Н.** Онтологический инжиниринг в процессе согласования программ учебных дисциплин // Тезисы доклада на Восьмой открытой всероссийской конференции под эгидой АП КИТ "Преподавание информационных технологий в Российской Федерации"–2010. URL: http://it-education.ru/2010/persons/conferee.php (дата обращения: 24.05.2010).

УДК 378.016+004.413:339.1

**Ю.П. Ехлаков,** д-р техн. наук, проректор,
Томский государственный университет систем управления и радиоэлектроники

E-mail: upe@tusur.ru

# Развитие профессиональных компетенций образовательного стандарта "Программная инженерия"

*Излагаются основные проблемы, возникающие у выпускников вуза при организации бизнеса на промышленном рынке прикладных программных продуктов; описываются и раскрываются дополнительные профессиональные компетенции, освоение которых позволит сформировать у студентов современный взгляд на проблемы и тенденции продвижения прикладных программных продуктов на промышленные рынки, развить практические навыки по организации и управлению бизнесом.*

***Ключевые слова:*** *рынок программных продуктов, сегментирование и позиционирование, нормативно-правовые основы организации малого бизнеса, командообразование, планирование и анализ финансовой деятельности, формы и методы ценообразования*

Значительную долю рынка прикладных программных продуктов (ПП) составляют малые предприятия, возглавляемые вчерашними выпускниками вузов, имеющими базовое образование, как правило, в области информатики и программирования. Специфика малого бизнеса на этом рынке состоит в том, что ПП как конечный интеллектуальный продукт создается небольшими коллективами разработчиков. При этом каждая фирма, создавая программный продукт для конкретного заказчика, стремится вывести на рынок и продать его еще нескольким потребителям, самостоятельно осуществляя изучение рынка и разработку стратегии продвижения. Сложность (проблемность) реализации таких задач заключается в отсутствии необходимых материальных средств для "раскрутки" ПП, низком уровне компетентности разработчиков в вопросах организации продвижения и продаж ПП, наличии высоких рисков при определении ресурсного обеспечения этих процессов.

Кроме того, молодые специалисты-разработчики ПП, желающие начать собственное дело, должны знать, как открыть и зарегистрировать компанию и организовать свой бизнес с учетом российского законодательства в области права и финансов. Становясь руководителями компаний, они сталкиваются с рядом и других проблем: как убедить заказчика не покупать готовые решения, а заказать разработку программного продукта с учетом специфики своего бизнеса; как определить и обосновать трудозатраты на его создание и договорную цену; как грамотно, с учетом зарубежных и отечественных стандартов, организовать процесс разработки; как, учитывая условия сложившегося рынка программных продуктов, обеспечить требуемый уровень рентабельности своего проекта. При выводе тиражного программного продукта на рынок остро встают проблемы разработки стратегии позиционирования и ценовой политики. Особую актуальность в последнее время приобретают вопросы правовой защиты интеллектуальной собственности на разрабатываемые ПП, регулирования прав и обязанностей разработчика и заказчика, защиты интересов каждой из сторон, выявления возможных последствий нарушения законов с учетом предусмотренных мер ответственности.

Анализ государственного образовательного стандарта третьего поколения по направлению подготовки "Программная инженерия" показал, что предлагаемые для освоения профессиональные компетенции (спо-

собность к выполнению начальной оценки степени трудности, рисков, затрат и формированию рабочего графика; способность к подготовке коммерческих предложений с вариантами решения; способность к оцениванию временной и емкостной сложности программного обеспечения; понимание стандартов и моделей жизненного цикла; понимание классических концепций и моделей менеджмента в управлении проектами; понимание методов управления процессами разработки требований, оценки рисков, приобретения, проектирования, конструирования, тестирования, эволюции и сопровождения) лишь частично покрывают спектр вышеперечисленных проблем, с которыми придется столкнуться выпускнику, желающему создать свой бизнес. В связи с этим представляется целесообразным в вариативной части примерного учебного плана по данному направлению подготовки ввести ряд дополнительных профессиональных компетенций, раскрывающих особенности промышленного рынка ПП, организации бизнеса, управления финансами и нормативно-правового регулирования деятельности на этом рынке.

В основу выбора дополнительных профессиональных компетенций и наполнения их соответствующим содержанием положены накопленные знания и опыт возглавляемого автором коллектива разработчиков прикладных программных продуктов и "шишки", набитые в процессе продвижения продуктов на рынок [1–3], а также материалы, изложенные в работах [4–8].

**1. Профессиональные компетенции по вопросам особенностей промышленного рынка ПП:**

- способность к проведению исследования и анализа промышленного рынка ПП;
- понимание проблем сегментирования, позиционирования и продвижения ПП на рынок.

Освоение данных компетенций будет направлено на понимание особенностей промышленного рынка ПП: понятий товара и услуги на рынке; роли и задач каждого из участников рынка (государства, разработчика, заказчика, посредника, партнера, конкурента); достоинств и недостатков двух возможных стратегий заказчика — приобретение готового (тиражного) программного продукта либо заказ новой разработки.

В развитие классической теории маркетинга следует раскрыть подходы к сегментированию рынка потенциальных пользователей, привести набор специфических для этого рынка переменных сегментирования, описать процедуру выделения целевых рынков с учетом реальных возможностей разработчика по тиражированию и сопровождению ПП; подробно описать стратегии продвижения ПП в среде Интернет, как в наиболее приемлемом канале продвижения для малого бизнеса.

Задачу позиционирования ПП следует представить с точки зрения всех сотрудников компании-заказчика, принимающих решение о приобретении продукта (первых руководителей, специалистов IT-служб, непосредственных пользователей).

**2. Профессиональные компетенции по вопросам организации бизнеса:**

- способность организовать компанию, умение выбрать наиболее приемлемую форму организации и структуру управления;
- понимание проблем командообразования с учетом специфики профессиональной деятельности и психологических особенностей программистов.

В этой группе компетенций следует раскрыть основы организации бизнеса: существующую законодательную базу организации малого бизнеса, возможные организационно-правовые формы деятельности (акционерное общество, индивидуальное предпринимательство), процедуры регистрации нового предприятия или приобретения уже существующей компании. При выборе и обосновании организационной структуры управления компанией необходимо описать содержание, преимущества и недостатки классических структур управления: функциональной, линейной (проектной) и матричной. В качестве основных моделей управления компанией предлагается раскрыть содержание бюрократической модели, основанной на жесткой регламентации деятельности, и модели участия, предусматривающей творческий подход исполнителей к работе.

Вопросы командообразования следует рассматривать как с точки зрения формирования команды, так и с точки зрения создания условий для ее эффективной работы: описать профессиональные и психологические особенности программиста, коммуникации и возможные конфликты, этапы формирования команды и роли каждого из участников, основные правила поиска и найма специалистов, мотивации, материальные и моральные стимулы к труду.

В качестве основной модели создания и продвижения ПП следует рассмотреть проектный метод: раскрыть основные идеи управления проектом по созданию и выводу на рынок ПП (выбор модели жизненного цикла, постановка целей, формирование календарного плана работ и управление ресурсами проекта, оценка рисков, мониторинг реализации проекта).

**3. Профессиональные компетенции в области экономики и финансов компании:**

- понимание основ финансовой деятельности компании;
- способность определить договорную и рыночную цены на ПП, рыночную стоимость продукта при полной либо частичной переуступке прав на ПП;
- способность к разработке бизнес-планов по созданию и продвижению на рынок новых ПП.

Профессиональные компетенции по основам управления экономикой и финансами компании должны быть направлены на понимание содержания ее финансовой деятельности. При этом необходимо раскрыть вопросы формирования бюджета и анализа финансовой деятельности, основные положения формирования политики ценообразования: возможные цели ценовой политики, формы и методы ценообразования. Вопросы экономической эффективности ПП

следует рассматривать как с точки зрения экономического эффекта от внедрения ПП в сфере производства и управления, так и оценки эффективности вложений в разработку ПП как инвестиционного проекта. При обосновании договорной цены на заказные программные продукты необходимо рассматривать два альтернативных метода:

- прямой метод, основанный на экспертном оценивании размерности программной системы в строках исходного кода;
- метод функциональных точек, позволяющий оценивать трудозатраты на разработку программных компонентов, реализующих конкретный функционал.

Определение рыночной цены при выводе и продвижении на рынок ПП целесообразно проводить с использованием концепции безубыточности.

Оценку рыночной стоимости ПП как продукта интеллектуальной деятельности предлагается рассматривать с точки зрения затратного (метод исходных затрат, метод замещения стоимости, метод восстановления стоимости); доходного (метод стоимости роялти, правило 25 %) и рыночного (метод сравнения продаж) подходов.

Вопросы создания бизнес-планов по разработке и продвижению на рынок программных продуктов должны раскрывать содержание и методики формирования как отдельных разделов плана (научно-технического, официально-экономического и коммерческого), так и бизнес-плана в целом.

**4. Профессиональные компетенции в области нормативно-правовых основ ведения бизнеса:**

- понимание и умение использовать отечественные и зарубежные стандарты на документирование, стадии жизненного цикла и оценку качества ПП;
- понимание особенностей ПП как интеллектуального продукта;
- понимание юридических и административных основ по защите прав и ответственности за правонарушения в области информационно-коммуникационных технологий.

Профессиональные компетенции в области нормативно-правовых основ ведения бизнеса должны быть направлены на получение знаний и умения использовать отечественные и зарубежные стандарты, регламентирующие процессы жизненного цикла разработки, документирования и оценки качества программных систем и информационных технологий. Необходимо раскрыть основы законодательной базы в области правовой защиты баз данных и программ для ЭВМ, представить особенности ПП как объектов интеллектуальной собственности, изучить вопросы авторского права (в том числе содержание авторского договора), регистрации и сертификации программ для ЭВМ, уголовной и административной ответственности за правонарушения.

На практических занятиях, направленных на закрепление и конкретизацию отдельных профессиональных компетенций, предлагается рассматривать следующие темы: системный анализ деятельности IT-компании; методы сегментации промышленного рынка ПП; алгоритмы формирования потребительских предпочтений пользователей; выбор и обоснование организационной структуры управления IT-компанией; выбор и обоснование юридической формы создания и регистрации компании; управление персоналом в IT-компании; разработка бюджета на создание и продвижение на рынок ПП; методы формирования договорной цены на разработку ПП; методы оценки рыночной стоимости ПП; защита авторских прав на ПП как объект интеллектуальной собственности.

Освоение предлагаемых дополнительных профессиональных компетенций позволит сформировать у студентов современный взгляд на проблемы и тенденции продвижения на промышленные рынки прикладных программных продуктов, развить практические навыки по организации и управлению бизнесом и тем самым сократить сроки адаптации при работе на этом рынке.

## СПИСОК ЛИТЕРАТУРЫ


1. **Ехлаков Ю.П.** Особенности развития рынка прикладного программного обеспечения // Промышленные контроллеры АСУ. 2002. № 6. С. 47–50.

2. **Ехлаков Ю.П.** Вывод прикладного программного обеспечения на рынок корпоративных продаж: взгляд разработчика // Маркетинг в России и за рубежом. 2009. № 4 (72). С. 45–50.

3. **Ехлаков Ю.П., Ефимов А.А.** Функциональные модели бизнес-процессов фирмы-посредника на рынке программных продуктов // Бизнес-информатика. 2010. № 1. С. 22–30.

4. **Архипенков С.** Руководство командой разработчиков программного обеспечения. Прикладные мысли [Электронный ресурс]: [персональный сайт]. URL: http://www.arkhipenkov.ru/index.files/publications.htm.

5. **Фатрелл Р.Т., Шафер Д.Ф., Шафер Л.И.** Управление программными проектами. Достижение оптимального качества при минимуме затрат. М.: Вильямс, 2004. 1136 с.

6. **Синк Э.** Бизнес для программистов. Как начать свое дело. СПб.: Питер, 2008. 256 с.

7. **Уэбстер Ф.** Основы промышленного маркетинга. М.: Издательский дом Гребенникова, 2005. 416 с.

8. **Шив Ч.Д., Хайэм А.** Курс MBA по маркетингу / пер. с англ. М.: Альпина Бизнес Букс, 2007. 718 с.

# CONTENTS



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

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

Сдано в набор 13.12.10 г. Подписано в печать 08.02.11 г. Формат 60×88 1/8. Бумага офсетная. Печать офсетная. Усл. печ. л. 5,88. Уч.-изд. л. 7,16. Цена свободная.

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