МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЁВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

А.В. БЛАГОВ

МЕНЕДЖМЕНТ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский государственный аэрокосмический университет имени академика С.П. Королева (национальный исследовательский университет)» в качестве учебного пособия для студентов, обучающихся по программам высшего образования по направлениям: 010302 «Прикладная математика и информатика» и 030301 «Прикладная математика и физика»

С А М А Р А
Издательство СГАУ
2014

УДК 004(075)
ББК 32.973 – 018.2я7
Б 681

Рецензенты: проректор по информатизации СГАУ,
канд. экон. наук Д. Е. П а ш к о в,
д-р техн. наук, проф. зав. каф. телекоммуникационных
систем УГАТУ А. Х. С у л т а н о в

Благов А.В.

Б 681 **Менеджмент разработки программного обеспечения:** учеб. пособие / *А. В. Благов*. – Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2014. – 84 с.

ISBN 978-5-7883-0987-3

В данном пособии рассмотрены особенности организационной структуры компаний по разработке программного обеспечения, а также основные этапы его разработки. Проведен анализ особенностей успешной работы по созданию программных продуктов в начинающей компании. Исследованы принципы, приёмы и методики разработки коммерческого программного обеспечения.

Пособие предназначено для студентов специальностей и направлений 010302 «Прикладная математика и информатика», 030301 «Прикладная математика и физика».

УДК 004(075)
ББК 32.973 – 018.2я7

ISBN 978-5-7883-0987-3

Оглавление

Введение

В современном мире все большую долю рынка занимают компании по разработке программного обеспечения (ПО). Подавляющее количество успешных инновационных проектов в последние несколько десятков лет относятся именно к IT сфере. Динамично развивающаяся индустрия разработок ПО имеет ряд отличительных особенностей, которые должны учитываться теми, кто связывает свою деятельность именно с этой отраслью.

Выпускникам технических вузов, обучившимся на специальностях по профилю информационных технологий и прикладной математики, необходимо обладать знаниями не только в области разработки ПО, но и в области организации и управления проектами в данной сфере.

Дисциплина «Менеджмент разработки программного обеспечения» уделяет особое внимание следующим моментам:

1. Знакомству с принципами, приёмами и методиками разработки ПО. Изучению основ, позволяющих выпускать качественные программы в срок.

2. Анализу реального опыта успешной разработки ПО в начинающей компании.

3. Получению навыков выбора нужных специалистов и инструментальных средств разработки ПО.

4. Формированию навыков по освоению технологий, планирования и выполнения проекта, своевременного обнаружения и решения возникающих в ходе разработки ПО проблем.

5. Знакомству с основными бизнес-подходами распространения ПО.

Студенты, завершившие изучение данной дисциплины, должны знать:

- основные бизнес-подходы к распространению ПО;
- принципы, приёмы и методики разработки ПО;
- основы методологии планирования разработки ПО;
- основы методологии тестирования разрабатываемого ПО;
- инструментальные средства разработки ПО;

- основы успешного завершения проекта, корпоративной культуры и кадровой политики;

уметь:

- подбирать коллектив разработчиков, организовывать эффективное взаимодействие нужных специалистов и выбирать инструментальные средства разработки ПО;

- создавать реальные планы разработки ПО;

- организовывать эффективное тестирование ПО и успешное завершение проекта в намеченные сроки;

- своевременно обнаруживать и решать возникающие в ходе разработки ПО проблемы.

1. Компании по разработке программного обеспечения и бизнес-планирование

Разработка программного обеспечения (ПО) в большинстве случаев рассматривается как коллективный труд специалистов, направленный на удовлетворение потребности пользователей в автоматизации их деятельности. Как и любой другой коллективный труд, она требует организации, в частности – управления. Это процесс порою длительный, связывающий производственными и иными отношениями тех, кого в той или иной степени можно рассматривать в качестве производителей программы.

1.1. Бизнес-модели развития компаний-разработчиков ПО

В настоящее время общепризнанными являются две модели развития компаний по разработке ПО:

- ***продуктовая*** (или «израильско-скандинавская»),
- ***проектная*** (или «индийская»).

В случае развития по продуктовой модели в разработку продукта и выведение его на рынок компания инвестирует самостоятельно либо с помощью инвестиционных фондов. Прибыль при этом получается от продаж тиражируемого программного продукта. В случае развития по проектной модели компания выполняет заказной проект по разработке ПО, получая прибыль по результатам выполнения заказа.

Потребность рынка в компаниях, построенных в соответствии с проектной моделью, в основном определяется хроническим дефицитом квалифицированных специалистов для исполнения проектов. Потребность определяется уровнем развития информационных технологий в конкретной стране и системой образования, которая готовит специалистов для отрасли. Деньги же из бюджета продаж и маркетинга тратятся в основном на поиск новых заказчиков, у которых по определению существует такая потребность в услугах компании, а не на развитие или создание рынка и формирование потребности в услугах компании. В продуктовой компании бюджет продаж и маркетинга сильно зависит от

того, как продукт «попал в рынок». То есть если продукт занимает нишу, где нет конкурентов и сам продукт достаточно качественный, он продает сам себя.

В табл. 1 приводится сравнение проектной и продуктовой компаний по таким параметрам, как затраты, прибыль и риски.

Таблица 1. Сравнение проектной и продуктовой компании

	Проектная	**Продуктовая**
Затраты	Затраты на разработку ПО сравнимы с затратами на продажи и маркетинг. Это обусловлено потребностью в качественном исполнении проектов. Могут меняться технологии, инструменты, но потребность будет только усиливаться.	Затраты на разработки значительно ниже затрат на продажи и маркетинг. Это обусловлено тем, что приходится завоевывать рынок, на котором уже существуют конкурентные продукты.
Прибыль	Прибыль заранее определена, т.к. проект делается для заказчика. Всё в большей степени зависит от наличия заказов и качества их исполнения.	Продукт делается в основном для будущего заказчика. В момент разработки и выхода на рынок компания является неприбыльной (наоборот требуются значительные вложения). Размер прибыли заранее не определен, однако при успешных продажах не ограничен, как в проектной компании.
Риски	В целом менее рисковая, т.к. прибыль заранее определена. Однако более высоки: - риски управления компанией, - риски роста, - риски, связанные с замыканием на одного заказчика, - страновые риски[1] (политические и экономические)	В целом модель более рисковая, т.к. прибыль заранее не определена.

Основные выводы по бизнес-моделям:

- проектная компания при правильном управлении является прибыльной и может существовать и развиваться на собственные средства;

[1] Производство в проектных компаниях обычно располагается в странах с более низким уровнем жизни.

- продуктовая модель может действительно приносить большие финансовые результаты при меньших управленческих затратах;

- продуктовая модель в подавляющем числе случаев требует значительно больших инвестиций для развития;

- продуктовая модель не может применяться в массовом порядке без присутствия достаточного количества инвесторов;

- продуктовая модель имеет большие риски в своей природе, поэтому инвестор должен быть не стандартным, а венчурным.

Наиболее яркими примерами компаний, развивающихся по продуктовой модели, являются Apple, Google, Microsoft и т.д. К числу проектных компаний можно отнести Net Cracker, Epam Systems, Cloud Castle и т.д.

Необходимо отметить, что возможно также частичное смешение моделей. В ходе работы над проектами могут быть найдены идеи продуктов, которые могут быть разработаны за счет внутренних инвестиций компании, потом опробованы на ряде заказчиков, а потом выделены в отдельную компанию («спин-офф») и «раскручены» с привлечением инвестиций. А для повышения прибыльности компании и использования наработанного опыта внутри продуктовой компании можно сформировать проектное подразделение.

1.2. Бизнес-подходы распространения программного обеспечения

В настоящее время общепризнанными являются несколько подходов к распространению ПО, определяющих следующую классификацию:

- коммерческое (проприетарное);
- «шароварное» (условно-бесплатное), в том числе тестовая версия;
- некоммерческое:
  - свободно-распространяемое,
  - ПО с открытым источником.

Первый вариант распространения ПО предполагает чисто коммерческие отношения с жестким пресечением нелицензионного использования ПО, вплоть до уголовного преследования.

Второй вариант («шароварное» ПО) предполагает свободное скачивание программного продукта и его использование потребителем с

предложением перевести разработчику оговоренную стоимость продукта после истечения срока тестового использования.

Следующие два варианта, условно обозначенные как «некоммерческое» ПО, предполагают его свободное бесплатное применение (и даже модификацию) в случае соблюдения условий лицензии, оговаривающей правила пользования. При этом лицензия оговаривает достаточно широкий спектр условий распространения.

Способы получения прибыли компаний, разрабатывающих коммерческое и условно-бесплатно ПО, очевидны. Рассмотрим варианты заработка средств компаниями, производящими некоммерческое ПО:

- использование ПО с открытым источником для создания рыночной позиции для закрытого ПО;
- использование ПО с открытым источником для создания рыночной позиции для оборудования;
- использование ПО с открытым источником для создания рыночной позиции для услуг;
- использование ПО с открытым источником для создания или поддержания брэнда;
- продажа «официальных аксессуаров» (от сувенирной продукции до документации);
- использование ПО с открытым источником для торговли оперативно обновляемым контентом.

1.3. Разработка бизнес-плана

Ключевым моментом для руководства продуктовой компании (а именно эту бизнес-модель мы примем за основу в силу больших инвестиций и свобод в принятии решений) и потенциальных инвесторов при принятии решения о начале финансирования разработки нового программного продукта является наличие качественного бизнес-плана данной разработки. Бизнес-план – это документ, в котором содержится информация об инвестиционном проекте. В данном документе описываются основные аспекты предпринимательской деятельности компании, анализируются главные проблемы, с которыми может столкнуться компания, и определяются основные способы решения данных про-

блем. Разработка такого документа – это комплексный процесс, который требует объективной оценки всех особенностей создания и продвижения на рынок нового программного продукта.

Бизнес-план может быть использован в качестве внутреннего документа при планировании деятельности продуктовой компании и в качестве коммерческого предложения для внешней стороны – инвестора или кредитора. В последнем случае бизнес-план служит обоснованием того, что инвестор или кредитор сможет вернуть свои деньги с прибылью. В то же время бизнес-план – это не только движение денежных средств, он должен отражать все составляющие инвестиционного проекта. Приведем краткие характеристики основных разделов бизнес-плана.

Титульный лист

Здесь приводятся название и адрес компании, имена и адресы учредителей, название проекта, иногда прописывается и его стоимость.

Резюме (вводная часть)

Резюме – это «концентрированное» описание инвестиционного проекта: его идеи; шагов, осуществляемых для реализации идеи; требуемых затрат; итоговых показателей. Содержание резюме должно очень четко отражать идею проекта, требуемые ресурсы, возможные риски и результаты реализации проекта.

Данный раздел должен показать потенциальным инвесторам привлекательность проекта. Главная его задача – «зацепить» инвестора. Инвесторы начинают знакомство с проектом с резюме, и нередко рассмотрение проекта на этом разделе и заканчивается. Поэтому необходимо отнестись к написанию резюме с особым вниманием: его четкости, понятности и содержательности.

Общепринятая практика представления бизнес-плана – сопровождение бумажного варианта презентацией для инвесторов, подготовленной в *Power Point, Prezi* или ином презентационном пакете. Вероятность благосклонной оценки инвестора после красивой презентации существенно увеличивается.

Информация о компании

В этом разделе необходимо четко и структурированно отразить информацию о компании (предприятии).

Первую часть раздела необходимо отвести для описания истории создания, области деятельности, динамики развития.

Вторая часть должна содержать анализ деятельности компании за предшествующий период: отражаются финансово-хозяйственные показатели работы компании за последний отчетный год, производится анализ финансового состояния компании. Если компания ранее привлекала кредиты, необходимо указать условия и график возврата средств.

Желательно представить информацию о финансовом состоянии компании не в целом за год, а в разбивке по кварталам или месяцам. Это позволит более четко представить динамику изменения финансового положения компании в течение года. Графики и диаграммы сделают представление информации более наглядным, а сравнение со среднеотраслевыми значениями позволят сделать вывод о конкурентоспособности компании.

Информация об отрасли

В этом разделе приводятся результаты анализа отрасли (развивающаяся, стабильная, стагнирующая), на которую ориентирован разрабатываемый программный продукт. Можно привести прогноз развития отрасли. Информацию можно получить в различных источниках: отраслевых министерствах, отраслевых изданиях, интернете, таможне, Госкомстате и т.п. Прогнозы развития рынка готовят также консультационные компании, когда проводят комплексные проекты.

Информация о продукции

В данном разделе приводятся характеристики разрабатываемого программного продукта, приводится описание продуктов-аналогов и проводится сравнительный анализ разрабатываемого продукта и существующих аналогов (заменителей). Акцент следует сделать на преимуществах продукта по отношению к конкурентам: лучшие технические характеристики, новые функции, лучший дизайн, надежность, простота

эксплуатации, более дружественный и «прозрачный» интерфейс и т.п. Для этого можно определить и проанализировать ключевые факторы принятия решения о покупке – путем проведения опросов потребителей, экспертов, целевых фокусных групп и т.д.

Производственный план

В разделе приводится краткое описание производственного процесса и технологии разработки, которое будет понятно неспециалисту, приводятся перечень работ, стоимость, условия поставки и условия оплаты необходимого оборудования, аренды помещения и каналов связи, коммунальных услуг, расходных материалов, служебных поездок и пр.

Потребители. Политика сбыта

Доказать, что реализация разработанного программного продукта (или сервиса) не вызовет серьезных проблем, – основная цель данного раздела.

В разделе должен быть приведен прогноз спроса на продукцию, география его распределения, структура, прогноз динамики развития различных сегментов, определены спрособразующие факторы и построен прогноз их динамики. Приводится характеристика основных потребителей продукции.

В разделе необходимо отразить ситуацию с патентной защитой продукции, а также ее названия. Особенно это важно при выводе на рынок нового бренда. Использование наименования продукции, которое юридически не защищено, может привести к печальным последствиям: кто-то раньше выпустил продукт или сервис с таким же именем. Рекламный бюджет проекта может быть потрачен впустую.

Дается характеристика планируемым методам продвижения и каналам сбыта продукции. Например, создается собственная торговая сеть или предполагается реализация по существующим каналам продаж.

Большим плюсом бизнес-плана является наличие протоколов о намерениях и предварительно заключенных договоров на поставку продукции.

Ценообразование

В этом разделе дается характеристика ценообразования – методика установления цены на продукцию, рассматриваемую в проекте.

Анализируется себестоимость продукции, конкурентоспособность товара по цене, приводится прогноз продаж продукции, какова вероятность демпинга со стороны конкурентов и т.п. Часто применятся метод: «издержки + прибыль» с обязательным учетом цен конкурентов, если таковые имеются.

Конкуренция

Оценка конкурентоспособности предприятия и разрабатываемой продукции – задача этого раздела.

В разделе приводится характеристика и анализ потенциальных конкурентов: численность персонала, структура затрат на производство, планируемые капиталовложения, объёмы продаж, система организации сбыта и производства, организационная структура и другое. Степень анализа зависит от доступности информации. Производится анализ слабых и сильных сторон конкурентов.

Определяются основные факторы конкурентоспособности предприятия (внутренние и внешние).

Оценка сильных и слабых сторон продукции и услуг компании, предложения по повышению ее конкурентоспособности, по корректировке продуктовой стратегии, системы сбыта и продвижения и повышению квалификации сбытового персонала – все это должен видеть в данном разделе потенциальный инвестор.

Информация о поставщиках

Приводится краткое описание поставщиков оборудования, необходимых материалов и комплектующих, аренды производственных помещений и коммуникаций.

При формировании плана реализации проекта желательно предусматривать «запасной выход» – альтернативные варианты снабжения производства необходимыми ресурсами. Такой подход является одним

из рычагов минимизации рисков проекта. Ориентация только на одного поставщика значительно увеличивает риски реализации проекта.

Организационный план

В этом разделе должна быть приведена характеристика системы управления проектом и описание организационной структуры предприятия.

Дается характеристика формы собственности (ООО, ОАО, др.). Определяется численность персонала, планируемые затраты на оплату труда. Приводится штатное расписание.

Обязательно следует предоставить схематическое изображение организационной структуры предприятия и проекта.

Рабочий график реализации проекта

В этом разделе следует описать временной график реализации проекта и проведение запланированных мероприятий. Приводится график осуществления капитальных вложений и выполнения работ, производственный график: начало производства продукции, выход на проектную мощность, объёмы производства в периоды сезонных колебаний.

Обычно в данном разделе принято представлять данные в следующей форме (табл. 2).

Таблица 2. Рабочий график реализации проекта

Период (месяц)	**0**	**1**	**...**	**12**
Мероприятие 1				
Мероприятие 2				
Мероприятие 3				
Капитальные затраты				

Финансовый план

Раздел предназначен для определения эффективности и финансовой состоятельности проекта. Он является ключевым разделом бизнес-плана.

Составление этого раздела – один из самых ответственных моментов. На основании данных финансового плана производится анализ коммерческой привлекательности проекта.

Здесь должна быть отражена информация о планируемых доходах проекта (объёмы реализации), текущих затратах проекта, об инвестиционных затратах (капитальные вложения, прирост оборотного капитала), планируемых источниках финансирования, их структуре (собственные, заемные), условиях привлечения и возврата заемных источников финансирования. Приводятся диаграммы и графики (рис. 1).

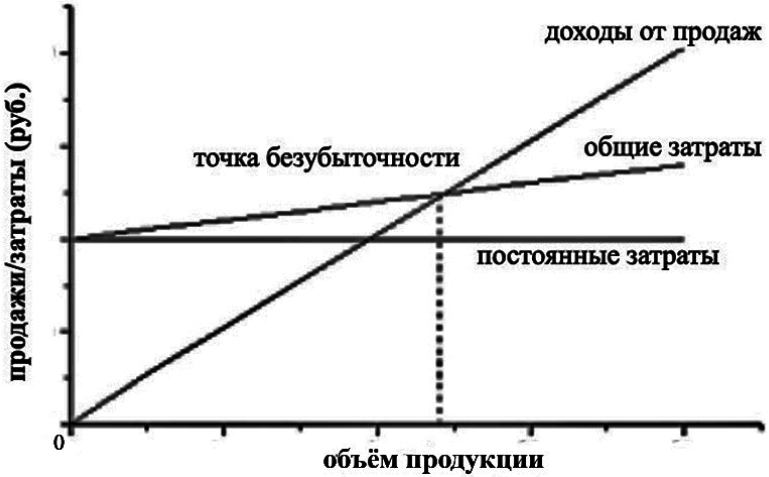

Рис. 1. График соотношения доходов и затрат (анализ безубыточности)

Обязательно должны быть приведены прогнозные формы финансовой отчетности: отчет о прибыли и убытках, отчет о движении денежных средств, баланс.

Приводится сводная характеристика эффективности проекта. Дается краткая интерпретация показателей.

Классически выделяются следующие показатели.

Чистая приведенная стоимость проекта (*NPV*). Показывает накопленный дисконтированный доход (или убыток) проекта. Рассчитывается как сумма дисконтированного чистого потока денежных средств и остаточной стоимости проекта.

Показатель *NPV* представляет собой разницу между всеми денежными притоками и оттоками, приведёнными к определенному моменту времени (моменту оценки инвестиционного проекта).

Для потока платежей *CF* (Cash Flow), где CF_t – платёж через t лет/месяцев $(t = 1,...,\mathrm{N})$, и начальной инвестиции *IC* (Invested Capital) в размере $IC = -CF_0$ чистая приведённая стоимость *NPV* рассчитывается по формуле:

$$NPV = \sum_{t=0}^{N} \frac{CF_t}{(1+i)^t} = -IC + \sum_{t=1}^{N} \frac{CF_t}{(1+i)^t},$$

где *i* — ставка дисконтирования.

При этом важным является такой показатель как ***внутренняя норма доходности*** *IRR,* обозначающий при какой ставке сравнения чистая текущая стоимость проекта становится равной нулю. Другими словами, *IRR* равен *i*, при котором *NPV=0.*

Срок окупаемости инвестиций (простой и дисконтированный) рассчитывается на основании чистого денежного дохода (простого и дисконтированного). Чистый денежный поток – сумма притоков и оттоков денежных средств. Дисконтирование – приведение денежных потоков разных интервалов планирования к начальному моменту времени (первому интервалу планирования). Суть дисконтирования заключается в том, что мы принимаем разновеликую стоимость одной условной денежной единицы в разные периоды времени.

Дисконтированный срок окупаемости инвестиций равен *n*, при котором:

$$\sum_{t=0}^{n} \frac{CF_t}{(1+i)^t} > IC\,.$$

Помимо этих показателей, часто берут в расчет остаточную стоимость и показатели финансовой состоятельности проекта.

К основным разделам бизнес-плана как правило, добавляются приложения с копиями основных уставных документов компании. Иногда в отдельный раздел выносят ***оценки рисков***, но их можно учесть и обозначать в описанных выше разделах.

2. Формирование команды разработчиков ПО и кадровые ресурсы

Главным активом компании по разработке ПО является команда. Она должна быть грамотной, квалифицированной и эффективной. Хотя команда создаётся из отдельных людей, нужно работать с коллективом, причем обдуманно и осторожно. К основам формирования коллектива можно отнести: анализ резюме, собеседование с кандидатами и создание необходимых для работы условий.

2.1. Резюме и его анализ

Умение отличить плохих кандидатов от хороших на основе их резюме – очень важное качество в работе любой компании.

Приведём основные аспекты, на которые следует прежде всего обращать внимание.

Опыт работы

Важным для оценки является величина срока работы кандидата в интересующей области, в каких компаниях он работал и насколько сложны проекты, с которыми имел дело, его роль в них. Всё это необходимо для определения уровня его квалификации. Предыдущий опыт скажется на способности специалиста разбираться с передовыми технологиями и применять их в работе. Хорошей компании нужны люди, справившиеся со сложной работой хотя бы в одной из интересующих её областей.

Преданность работе

Один из лучших способов оценить это качество – посмотреть в резюме: если за последние три года кандидат сменил четыре места, его преданность работе можно поставить под вопрос. Причины смены работы могут быть вполне обоснованными, но такое поведение заслуживает пристального внимания.

Уровень образования

Можно относиться к этому аспекту по разному главное, чтобы кандидат обладал необходимыми навыками и знаниями в нужной области. Однако хорошее специальное образование, полученное в ведущем высшем учебном заведении, будет сильной стороной кандидата. Кроме этого важно учитывать, окончил кандидат вуз либо продолжает учиться. От этого как минимум зависит его возможность уделять необходимое время работе.

Форма описания

В резюме стоит обращать внимание на форму его описания, к примеру на использование, «сильных» и «слабых» глаголов. Как правило, хорошие работники демонстрируют свою приверженность делу и ответственность за решаемые задачи и гордятся этим.

Показателем такого положительного качества обычно являются «сильные» глаголы, например:

- произвёл;
- решил;
- управлял;
- определил;
- написал;
- интегрировал;
- создал.

«Слабые» глаголы в целом означают меньшую приверженность делу и ответственность за него и часто характеризуют нерешительность. Применение таких слов может указывать, что кандидат не в полной мере овладел предметной областью. Примеры таких глаголов:

- принимал участие;
- ознакомился;
- следовал;
- помогал;
- способствовал;
- комментировал.

Важна способность письменно излагать свои мысли, письменная речь хорошего кандидата грамотна и лаконична.

Профессиональный и общий кругозор

Определенной мерой профессионального кругозора кандидата является оценка его способности работать на разных уровнях абстрагирования. Может ли он работать как с высокоуровневым кодом (например, с пользовательским интерфейсом и общей логикой программы), так и с низкоуровневыми технологиями (потоки, управление памятью, внутренняя организация ОС и т.п.). Важным является и то, какими технологиями и предметными областями знаний интересуется кандидат, следит ли за техническими и программными новинками. Из увлечений (хобби), даже порой не профессиональных, можно также сделать важные выводы. Например, занимается спортом – значит ведет здоровый образ жизни и будет мало болеть, плюс, скорее всего, есть воля к победам.

Несмотря на всё вышеперечисленное, резюме не дает полной картины о возможном будущем работнике. Оно лишь помогает отобрать кандидатов для собеседования.

2.2. Собеседование

В собеседовании должны принимать участие все сотрудники, которые будут непосредственно связаны с новичком. Не нужно забывать, что создаётся команда и важно принятие нового сотрудника всеми её членами. Им будет легче это сделать, если они примут непосредственное участие в выборе кандидатов.

На собеседовании, прежде всего, необходимо определить следующее (табл. 3).

Таблица 3. Что необходимо выяснить на собеседовании

Что?	Как?
Уровень знаний	Опрос в тех областях знаний, которые требуются для выполнения данной работы. Особенно важен опрос в областях, указанных кандидатом в резюме. Уместным будет небольшое тестирование.
Квалификация	Нужно узнать, какие проекты вел кандидат, его место в проекте. Можно спросить о какой-нибудь сложной проблеме, которую пришлось кандидату выявить и (или) устранить. Как это было?
Умение работать в команде	Можно задать вопросы относительно случаев, когда для решения задач кандидату требовалась помощь других специалистов. Долго ли он ждал помощи? Как взаимодействовали? Как решили? Какие для кандидата наиболее важные принципы плодотворной работы с людьми? Почему?
Отношение к делу	Попросить кандидата рассказать о случае на его бывшей работе (деятельности), когда команда не уложились по проекту (задачам) в сроки? Почему это произошло? Как была реакция? И как решили проблему? Можно уточнить, был ли случай, когда кандидат отвлекся от своих дел, чтобы помочь кому-то другому. Почему это произошло? Каков был результат?
Преданность	Если кандидат менял несколько раз места работы, необходимо узнать, по каким причинам.
Жажда знаний	Можно уточнить, чем кандидат интересуется, как поддерживает знания, что изучает, что читает.

Типичные ошибки при собеседовании, которые необходимо избегать:

а) вакансия плохо описана;

б) интервьюер один, да к тому же новичок;

в) много времени затрачивается на неквалифицированных кандидатов;

г) чрезмерная или недостаточная реклама возможностей;

д) чрезмерная медлительность в принятии решения.

Решение по кандидату необходимо принимать совместно со всеми, кто участвовал в собеседовании, взвешивая все «за» и «против». Очень важен анализ разработок кандидата, если таковые имеются. При положительном прохождении собеседования необходимо сформировать у кандидата понимание следующих аспектов относительно работы:

- с какими технологиями он будет работать, чем ему предстоит овладеть;

- над какими проектами он будет работать и в какой команде;

- чем интересна и уникальна компания, в которой он будет работать;

- какие есть преимущества у данной работы.

При принятии положительного решения по кандидату необходимо незамедлительно его об этом оповестить. Если кандидат к этому моменту отказывается или сомневается (как правило, люди, находящиеся в поисках работы, рассматривают сразу несколько предложений), необходимо узнать причины и, если возможно, устранить их.

2.3. Кадровое обеспечение проекта

Специалиста, принятого в компанию по разработке ПО, обычно сразу же «ставят» на тот или иной проект, а порой даже на несколько. Рассмотрим подробнее, каким образом происходит кадровое обеспечение проекта.

Во-первых, менеджер проекта (главное лицо в проекте, в следующей главе будет подробно рассмотрена его роль) может заранее знать возможных кандидатов – хорошо себя зарекомендовавших сотрудников фирмы, с которыми он уже имел дело либо знает по чужим работам.

Однако следует иметь в виду, что ни в коем случае нельзя подменять знание о человеке как о работнике сведениями личного характера (приятельские отношения, коммуникабельность и др.).

Во-вторых, менеджер может подбирать кандидатов из числа сотрудников фирмы, с которыми он еще не имел дело. В этом случае ему не обойтись без изучения послужного списка сотрудника, собеседований и прочих способов получения сведений о человеке.

Третий вариант, на который стоит рассчитывать, если возможности предыдущих вариантов исчерпываются, – это принять на работу нового сотрудника. Ситуация почти такая же, как в предыдущем случае, но несколько хуже: про сотрудника фирмы довольно просто получить объективную информацию, тогда как нанимая нового человека, приходится довольствоваться сведениями из документов и результатами собеседований.

Подбор кандидатур на *ключевые роли* в проекте полезно проводить в соответствии с ***графиком привлечения сотрудников к проекту***. Такой график необходимо составить к началу периода, когда решение о проекте утверждено. В нем следует указывать сроки начала работ и оценки их продолжительности, а также квалификационные требования. В ходе последующей разработки график может корректироваться, но начальные установки важны как для руководства, так и для сотрудников. Конкретные персоналии в графике устанавливаются для ключевых ролей, но оценка кадровой потребности должна быть выполнена заранее и по другим ролям.

Составляя график в ходе предпроектной деятельности, менеджер проекта не должен предполагать, что все занимаемые вакансии жестко фиксированы (сотрудник может получить более выгодное предложение, заболеть и т.д.) Уже по этой причине целесообразно, формируя график привлечения сотрудников к проекту, предварительно включать в него всех возможных кандидатов на роли. Это позволит выбирать, а впоследствии даже менять исполнителей ролей.

Решение задачи определения кадровых ресурсов проекта схематично можно изобразить следующим образом (табл. 4).

Таблица 4. Решение задачи определения кадровых ресурсов проекта

<table>
<tr><th>До начала выполнения проекта</th><th>Утверждение кадровой политики проекта</th><th>В ходе выполнения проекта (по мере необходимости)</th></tr>
<tr><td>Кадровые потребности проекта</td><td rowspan="2">График привлечения сотрудников к проекту</td><td rowspan="3">Заполнение вакансий</td></tr>
<tr><td rowspan="2">Оценка распределения кадровых потребностей по времени</td></tr>
<tr><td>Критические ролевые позиции проекта</td></tr>
<tr><td>Возможности подбора кадров на проект</td><td></td><td></td></tr>
</table>

2.4. Удерживание сотрудников и сохранение команды

Важным является не только набор ценных сотрудников, но также их удержание с целью сохранения команды. Учитывая большое количество существующих в настоящее время компаний по разработке ПО, необходимо понимать, что предложение на квалифицированных разработчиков на рынке труда очень велико.

Есть три основные группы причин, по которым люди остаются у своего работодателя:

- профессиональные,
- финансовые,
- социальные.

Рассмотрим поподробнее каждую из групп.

Профессиональные

Хорошие работники любят работать. Они очень гордятся своим делом и хотели бы видеть в нём что-то значительное. Люди должны знать, что их работа важна и по достоинству оценивается. Им нужно знать: их компания выделяется среди других, и они внесли свою лепту в её успех. При этом важно понимать, что с увеличением опыта

и квалификации должны ставиться более ответственные задачи и предлагаться более значимые роли и позиции. Кроме этого надо предоставлять сотрудникам возможность заниматься новыми вещами. Новые интересные задачи и технологии, с которыми они столкнутся на новом месте, не позволят им потерять интерес к работе. Важны проводимые компанией занятия и курсы по повышению уровня профессиональных навыков (по освоению новых технологий, языков и т.д.).

Таким образом, чтобы сотрудники не покинули компанию по причине неудовлетворения профессиональных интересов, руководству нужно:

- убедиться, что люди получают новые навыки и пробуют что-то новое;
- дать понять работникам, что они отвечают за свою работу;
- сделать так, чтобы работники осознавали важность своих продуктов и проектов;
- хвалить людей как лично, так и публично;
- чаще переводить людей из одной группы в другую, чтобы обеспечить их рост и взаимозаменяемость;
- узнавать о целях карьеры своих сотрудников и обеспечивать их карьерный рост.

Финансовые

Проблемы с низким финансированием очень часто приводят к тому, что сотрудники меняют работу. Оплата должна включать базовую зарплату, премии за особые достижения и периодические выплаты, стимулирующие заинтересованность в долговременном успехе компании. Такая схема оплаты не даёт сотрудникам расслабляться и удерживает их, если акции компании растут.

Талантливые люди редко испытывают трудности с поиском высокооплачиваемой работы. Но если вы предлагаете больше среднего, люди считают, что вы их цените высоко, и даже более солидная зарплата в других местах становится для них не такой привлекательной. Риск потерять человека из-за денег будет меньше, если вы и другими способами демонстрируете, что вы его цените.

Поэтому руководству компании следует:

- выдавать зарплату и другие выплаты талантливым сотрудникам, которые должны быть выше средних по отрасли;

- выдавать премии за достижения;

- премировать ведущих сотрудников акциями компании (если таковые имеются).

Но если же сотрудника интересуют только деньги, это, вероятно, не тот работник, который воле нужен.

Социальные

Рабочее место, являющееся частью социальной среды, может влиять на удержание сотрудника. Если люди общаются со своими коллегами и довольны рабочей обстановкой, очень маловероятно, что они захотят поменять работу (если при этом удовлетворены их профессиональные и материальные потребности). Нельзя недооценивать удобство офиса, мощность компьютеров и другой техники, находящихся в распоряжении каждого члена коллектива. А если помимо этого компания заботится и о здоровье, отдыхе и досуге сотрудника, то это может хорошо сплотить коллектив, из которого сложно будет уйти.

Поэтому руководству компании в части создания благоприятной социальной среды следует:

- прикреплять к новичкам старых сотрудников, чтобы они по-дружески помогали им прижиться в коллективе;

- организовывать внеурочные мероприятия (спортивные игры, совместный отдых);

- заботиться о социальных контактах между членами коллектива;

- поощрять социальную активность как на рабочем месте, так и вне его.

Говоря об удержании сотрудников, необходимо на регулярной основе обеспечивать баланс между профессиональной удовлетворённостью, денежным стимулом и социальной поддержкой. Не ждать, когда начнут возникать проблемы. При этом надо понимать, что текучка кадров существует всегда. Меняются личные обстоятельства и приоритеты. Люди уходят по причинам, которые компания не может контро-

лировать: рождение ребёнка, переезд и т.д. С другой стороны, текучка может указывать на серьёзные проблемы в коллективе или организации. Чтобы оставаться в курсе причин текучести кадров, руководству необходимо беседовать с людьми перед их увольнением, прислушиваться к их замечаниям. И если обнаруживается внутренняя проблема, нужно спрашивать других сотрудников, разделяют ли они такую точку зрения. В больших организациях неплохо вести список причин увольнения сотрудников. Эти сведения помогут отслеживать тенденции и принимать соответствующие меры.

При этом необходимо всегда помнить, что главным активом компании по разработке ПО является команда!

3. Организационная структура компании по разработке ПО. Роли, обязанности, функции

Любая компания по разработке ПО, состоящая из высококвалифицированных специалистов, будет работать с максимальной эффективностью лишь в том случае, если она правильно организована. Бывает так, что проекты часто страдают от недостатка организованности и неясностей в распределении ролей и обязанностей, что приводит к значительным затратам ресурсов. Каждый должен знать свои роли и обязанности в общем контексте проекта.

3.1. Модель организационной структуры компании

Команда разработчиков – это группа людей с различными техническими навыками, работающих над реализацией общего проекта. Поскольку разработать ПО довольно сложно, в команду требуются специалисты с самыми разными навыками и способностями, необходимыми для создания продукта.

В компании по разработке ПО принято делить команду на две группы: ***основную*** и ***вспомогательную***. В ***основную*** группу входят специалисты, непосредственно занятые созданием нового программного продукта, а именно:

- менеджеры проекта,
- программисты (разработчики),
- тестировщики,
- разработчики технических документаций (технические писатели),
- специалисты по графическому дизайну и т.д.

Во ***вспомогательную*** (не менее важную) группу принято включать:

- специалистов по менеджменту и маркетингу продукта,
- специалистов по технической поддержке,
- системных администраторов и т.д.

Очень важно, чтобы перечисленные выше функциональные подразделения участвовали в работе над проектом с самого начала. Чем раньше люди смогут понять суть требований к продукту и принять участие в их критическом анализе, тем лучше подготовятся к исполнению собственной миссии и ощутят свой вклад в успех проекта. Кроме того, чтобы завершить создание продукта в срок, все перечисленные подразделения должны работать параллельно на протяжении всего цикла разработки.

3.2. Роли и обязанности в коллективе разработчиков

Рассмотрим основные роли и обязанности лидеров функциональных подразделений, участвующих в создании продукта.

Ключевую роль в проекте играет ***менеджер проекта***.

Менеджер проекта (Project Manager) – отвечает за развитие проекта в целом, гарантирует, что распределение заданий и ресурсов позволяет выполнить проект, работы и предъявление результатов идут по графику, результаты соответствуют требованиям. В рамках этих функций менеджер проекта взаимодействует с заказчиком. Обязанности менеджера проекта:

- подбор кадров и управление ими,
- формулирование и исполнение плана проекта (графика),
- руководство командой,
- обеспечение связей между подразделениями,
- обеспечение готовности продукта.

Руководитель команды (Team Leader) – производит техническое руководство командой в процессе выполнения проекта. Для больших проектов возможно привлечение нескольких руководителей подкоманд, отвечающих за решение частных задач.

Архитектор (Architect) – отвечает за проектирование архитектуры системы, согласовывает развитие работ, связанных с проектом.

Проектировщик подсистемы (Designer) – отвечает за проектирование подсистемы или категории классов, определяет реализацию и интерфейсы с другими подсистемами.

Эксперт предметной области (Domain Expert) – отвечает за изучение сферы приложения, поддерживает направленность проекта на решение задач данной области.

Разработчик (Developer) – реализует проектируемые компоненты, владеет и создает специфичные классы и методы, осуществляет кодирование и автономное тестирование, строит продукт. Это широкое понятие, которое может подразделяться на специальные роли (например, разработчик классов). В зависимости от сложности проекта команда может включать различное число разработчиков.

Разработчик информационной поддержки (Information Developer) – создает документацию, сопровождающую продукт, когда выпускается версия. Включаемые в нее инсталляционные материалы, равно как ссылочные и учебные, а также материалы помощи предоставляются на бумажных и машинных носителях. Для сложных проектов возможно распределение этих задач между несколькими разработчиками информационной поддержки.

Специалист по пользовательскому интерфейсу (Human Factors Engineer) – отвечает за удобство применения системы. Работает с *заказчиком*, чтобы удостовериться, что пользовательский интерфейс удовлетворяет требованиям.

Тестировщик (Tester) – проверяет функциональность, качество и эффективность продукта. Строит и исполняет тесты для каждой фазы развития проекта.

В зависимости от проекта и условий его выполнения роли участников проекта могут совмещаться. Понятно, что совмещение ролей не

может быть произвольным. Одни совмещения нежелательны (в разной степени), другие нейтральны, третьи полезны для проекта. Общий регламент для совмещения определяют следующие принципы:

1. Не следует допускать совмещения ролей, которые имеют конфликтные или противоречивые интересы. Если такое совмещение все-таки используется, то необходимо предусматривать механизмы, которые будут демпфировать противоречия.
2. Представление ролей с конфликтными интересами различным людям обеспечивает равновесие противоречащих точек зрения.
3. Прибегать к совмещению ролей для участников проекта, основной ролью которых является разработка, означает заведомое увеличение сроков выполнения соответствующих работ. По этой причине для них допустимы лишь такие совмещения, которые действительно необходимы в данный момент развития проекта (например, в некоторых авральных ситуациях) не требуют постоянной деятельности в течение длительного срока.
4. Если работнику поручается несколько ролей, то он всегда должен знать, какую из них он выполняет в данный момент.

К примеру, совмещение ролей менеджера проекта (Project Manager) и руководителя команды (Team Leader) допустимо, но лишь тогда, когда осознается и учитывается противоречивость целевых установок этих ролей: руководитель команды действует в условиях, которые формируются менеджером.

Нежелательно совмещение ролей руководителя команды и проектировщика какой-либо подсистемы. И это обусловлено противоречивостью их ролевых интересов.

В табл. 5 приводятся краткие характеристики совмещения ролей, которые можно рассматривать как рекомендации для менеджмента. Разумеется, нельзя абсолютизировать эти характеристики для любых проектов. Не стоит жестко фиксировать распределение ролей и в одном проекте. Более того, распределение ролей можно рассматривать в качестве одного из инструментов, с помощью которого менеджер может руководить коллективом.

Таблица 5. Роли и характеристики их совмещения

Роли	Характеристика совмещения ролей
Менеджер проекта и архитектор	Желательно
Менеджер проекта и руководитель команды	Противоречиво
Руководитель команды и архитектор	Возможно
Руководитель команды и проектировщик подсистемы	Нежелательно
Менеджер проекта и разработчик	Не допускается
Менеджер проекта и специалист по интерфейсу	Возможно
Менеджер проекта и эксперт предметной области	Возможно
Эксперт предметной области и специалист по интерфейсу	Обычно нежелательно
Эксперт предметной области и разработчик	Желательно
Специалист по интерфейсу и разработчик	Желательно
Эксперт предметной области и тестировщик	Возможно, порой желательно

На практике часто придерживаются следующей классификации уровней разработчиков:

- ведущий разработчик (Lead Developer), как правило, исполняет роль Team Leader’a,
- старший разработчик (Senior Developer),
- разработчик (Developer).

В некоторых компаниях практикуют и роль младшего разработчика (Junior Developer). Как правило, ими становятся новички, в том числе проходящие испытательный срок.

В обязанности данных групп разработчиков обычно входит следующее.

Ведущий разработчик (Lead Developer):

- архитектура продукта,
- подбор ключевых инструментов и стандартов,
- диагностика и решение технических проблем,
- технический инструктаж и консультации по продукту,
- наблюдение за работой групп разработчиков документов и тестировщиков,
- мониторинг состояния систем,
- программирование.

Старший разработчик (Senior developer):

- реализация отдельных функций, конкретной технологии,
- проектирование функций,
- согласование архитектуры,
- формулировка требований к функциям,
- снабжение тестировщиков и технических писателей материалами,
- программирование.

Разработчик (Developer):

- реализация функций,
- помощь тестировщикам и исправление ошибок,
- помощь писателям в документировании.

Связи между разработчиками схематично представлены на рис. 2:

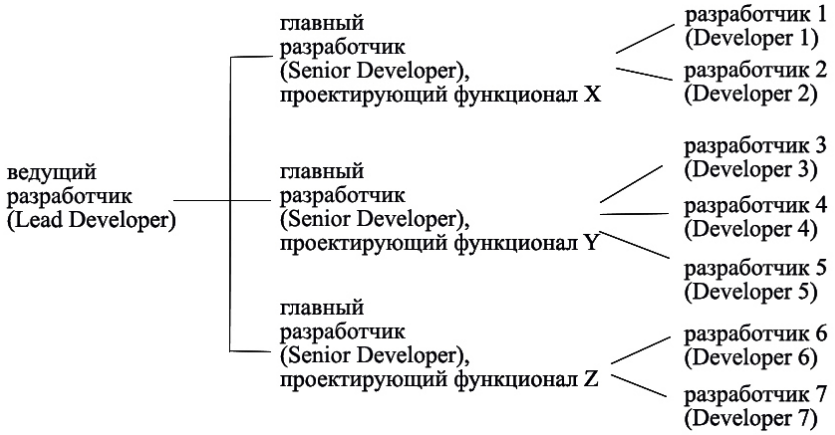

Рис. 2. Иерархия разработчиков и связи между ними

Тестировщики, в свою очередь, также могут иметь свою классификацию:

- ведущий тестировщик,
- инженер по автоматизации,
- тестировщик.

В обязанности данных групп тестировщиков обычно входит следующее.

Ведущий тестировщик:

- составление и исполнение плана тестирования,
- выбор инструментов,
- автоматизация испытаний.

Инженер по автоматизации:

- планирование испытаний,
- автоматизация испытаний,
- оценка и выбор инструментальных средств.

Тестировщик:

- тестирование программы по плану,
- проведение испытаний,
- окончательное подтверждение устранения ошибок,
- подготовка среды для испытаний.

Менеджер проекта при таких схемах ставит задачи ведущим специалистам в тех или иных группах. При этом под его руководством распределяются роли среди других участников проекта. Возможна ситуация когда конкретный разработчик может получить одновременно несколько ролей, и наоборот, роль может быть распределена между несколькими исполнителями. Когда менеджеру в конкретных условиях руководства коллективом приходится распределять роли, он неизбежно сталкивается с тем, что эта задача зависит и от специфики проекта, и от контингента исполнителей. В этой связи уместно упомянуть об одном из решений, которое предлагается специалистами Microsoft в качестве универсального подхода. Речь пойдет о модели проектной группы концепции Microsoft Solution Framework (MSF).

3.3. MSF. Ролевые кластеры

В модели проектной группы концепции Microsoft Solution Framework (MSF) предлагается образовывать небольшие мобильные коллективы (производственные единицы) с общей ответственностью за выполняемые задания – так называемые проектные группы. Такие группы строятся как многопрофильные команды, члены которых распределяют между собой ответственность и дополняют области компетентности друг друга. Группа состоит не более чем из 10 человек. Все они считаются обладающими сходным уровнем профессиональной подготовки, хотя и в разных областях индивидуальной специализации. Провозглашается равноправие членов группы и коллективная ответственность за выполняемые задания: проектная группа – команда равных. Все это позволяет сохранять внутри группы неформальные отношения.

Вместо понятия роли для группы в целом определяются ***ролевые кластеры***. В то время как за успех проекта ответственна вся команда, каждый из ее ролевых кластеров, определяемых моделью, ассоциирован с одной из проектных целей и работает над ее достижением. В данной модели именно эти цели задают роли разработчиков, которые определяются кластерами. В терминологии MSF используется понятие ***области компетенции***, или ***области функциональной специализации*** (*functional area*), обозначающее ту или иную роль, которую выполняет *кластер* группы в проекте. Принципиальное отличие распределения исполнителей по *ролевым кластерам* от распределения ролей заключается лишь в том, что ответственность за это несет не менеджер проекта, а сама группа. Менеджер проекта выдает задания и контролирует их выполнение лишь в целом для группы, не вмешиваясь в ее работу.

Определено шесть ролевых кластеров, которые соответствующим образом структурируют проектные функции разработчиков (рис. 3).

Управление продуктом (*product management*). Ключевая цель кластера – обеспечивать удовлетворение интересов заказчика. Для ее достижения кластер должен содержать следующие области компетенции:

- планирование продукта,

- планирование доходов,
- представление интересов заказчика,
- маркетинг.

Рис. 3. Ролевые кластеры и модели проектной группы MSF

Управление программой (*program management*). Задача – обеспечить реализацию решения в рамках ограничений проекта, что может рассматриваться как удовлетворение требований к бюджету проекта и к его результату. Области компетенции кластера:

- управление проектом;
- выработка архитектуры решения;
- контроль производственного процесса;
- административные службы.

Разработка (*development*). Первостепенной задачей кластера является построение решения в соответствии со спецификацией. Области компетенции кластера:

- технологическое консультирование;
- проектирование и осуществление реализации;
- разработка приложений;
- разработка инфраструктуры.

Тестирование (*test*). Задача кластера – одобрение выпуска продукта только после того, как все дефекты выявлены и устранены. Области компетенции кластера:

- разработка тестов;
- отчетность о тестах;
- планирование тестов.

Удовлетворение потребителя (*user experience*). Цель кластера – повышение эффективности использования продукта. Области компетенции кластера:

- общедоступность (возможности работы для людей с недостатками зрения, слуха и др.);
- интернационализация (эксплуатация в иноязычных средах);
- обеспечение технической поддержки;
- обучение пользователей;
- удобство эксплуатации (эргономика);
- графический дизайн.

Управление выпуском (*release management*). Задача кластера – беспрепятственное внедрение и сопровождение продукта. *Области компетенции* кластера:

- инфраструктура (infrastructure);
- сопровождение (support);
- бизнес-процессы (operations);
- управление выпуском готового продукта (commercial release management).

Видно, что в MSF многие задачи менеджмента перенесены на уровень компетенции групп. В части управления проектом менеджер оказывается в состоянии оперировать крупными заданиями стратегического характера. Эта ситуация во многом похожа на схему экстремального программирования, которая дополняется явным предписанием выделять ролевые кластеры со своими областями компетенции. Если в конкретной ситуации не требуется обеспечивать какую-либо из областей компетенции, то соответствующий кластер не образуется. Организация проектных групп позволяет существенно разгрузить менеджера, что фактически и провозглашается как цель схемы MSF.

3.4. Типичные проблемы в организационной структуре компании и их решение

Принято выделять три основные проблемы, которые могут возникнуть в организационной структуре компании, занимающейся разработкой ПО.

Слишком расплывчатый либо жестко определенный круг обязательств

Формулировки обязанностей должны быть подробными и ориентированными на решение тех или иных задач. Но не следует бросаться из крайности в крайность, иначе эти формулировки могут стать слишком формальными и жёсткими. Вряд ли нужно, чтобы участники группы лишь цитировали описание их задания, когда пора уже выпускать продукт. Главное – избежать основных просчётов при исполнении проекта, а не пытаться управлять всем и всеми, даже в мелочах.

Дисбаланс подразделений

Наличие модели, которая поощряет равновесие навыков и опыта специалистов различных подразделений, не означает, что обеспечение проекта кадрами осуществлялось как надо. Необходимо на протяжении всего выполнения проекта следить за балансом ресурсов функциональных подразделений проекта. Например, хватает ли в группе тестировщиков для реализации проекта. Отношение числа разработчиков к числу тестировщиков критично для проекта и должно быть сбалансировано. Значение этого отношения зависит от потребностей проекта, но большинство организаций стремится поддерживать его между 2:1 и 4:1. Даже если не соблюдать такое отношение, тестирование всё равно придётся проводить, только в этом случае оно займёт больше времени.

Недостаток масштабируемости

Один из недостатков модели с одним менеджером проекта во главе – это слабая масштабируемость. При расширении числа участников проекта, скажем с 12 до 80, единственного менеджера уже будет недостаточно для работы проекта. И в этой ситуации два выхода. Первый – увеличение числа ведущих специалистов: разработчиков, тестировщиков, технических писателей и т.д. В отношении вверенных им групп

они берут на себя значительную часть обязанностей, выполняемых менеджером проекта. Обычно это решение даёт неплохие результаты, особенно если проект остаётся однородным, т.е. вся команда работает над созданием одного и того же продукта. Второй подход – создать несколько групп со стандартной структурой организации (с менеджером проекта во главе), небольшие или средние по размеру. Это особенно хорошо работает, когда в рамках проекта ведётся разработка независимых программ, например, двух продуктов, редакций или независимых компонентов одного продукта. Сформировав несколько небольших групп, можно решить проблему с масштабируемостью, при этом правда потребуется решение дополнительных проблем по взаимодействию.

4. Ранжирование сотрудников, качество работы и вознаграждение. Корпоративная культура

В современных динамично развивающихся компаниях по разработке ПО бывает важно оценить эффективность каждого отдельного сотрудника по размеру вклада и его значению для организации, не отрицая важности работы и личного вклада других участников группы. В основе оценки значения индивидуума — его вклад, а не выполняемые им обязанности или положение в иерархии компании.

4.1. Ранжирование

Ранжирование представляет собой методику выделения тех или иных лиц по определенным принципам, к примеру, в спорте самые результативные спортсмены постепенно входят в ранг «привилегированных». Их значение для команды настолько велико, что с ними заключаются контракты на особых условиях. В компаниях же, в том числе и по разработке ПО, используют такие приставки к названию должности, как «старший» или «главный» специалист. Во всех примерах ранг служит для решения важных кадровых вопросов.

Нужно помнить, что правила ранжирования должны быть просты, не стоит чрезмерно усложнять их, чтобы не посвящать расчёту ранга большую часть своего времени. Часто проще всего вести ранжирование, приписывая сотрудников к одному из кругов: внутреннему, среднему или внешнему.

Внутренний круг составляют сотрудники, наиболее значимые для компании. Любой хороший руководитель или ведущий специалист должен знать, кто из участников команды вносит наибольший вклад и на кого можно всегда положиться. Эти люди – движущая сила проекта (а часто и всей компании). Их участие имеет стратегическое значение для успеха работы команды, поэтому их нужно соответствующим образом выделять и поощрять.

Как правило, внутренний круг составляют самые старшие по должности и наиболее одарённые участники коллектива, обладающие наибольшим доверием. На рабочих собраниях они пользуются большим авторитетом при выборе стратегии, определении направленности продукта и решении других важных для компании вопросов. Возглавляя функциональные подразделения, они олицетворяют собой мастерство и опыт. Эти люди в полной мере ощущают ответственность за создание продукта и делают всё возможное для его успеха.

Средний круг включает перспективных сотрудников. Они очень важны для успеха проекта. Как правило, недостаток опыта по сравнению с людьми внутреннего круга компенсируется у них неуёмным энтузиазмом, заинтересованностью, большим потенциалом роста и амбициями.

Здесь также можно встретить людей, которые работают неплохо, но обычно не демонстрируют исключительных качеств. Они стабильны, надёжны и последовательны. Создание текущей версии продукта во многом зависит от их способностей, однако их нельзя считать «ключевыми игроками».

Внешний круг по большей части состоит из людей, новых для организации, которые ещё не оправдали ожиданий в полной мере. Какими бы многообещающими ни казались новые работники, пока они всё рав-

но остаются неизвестными и неиспытанными. Нужно дать им время, чтобы влиться в работу организации и доказать свои способности делом.

Людей внешнего круга можно без особого труда заменить. Их внезапный уход из организации не будет иметь стратегических последствий для её успеха.

Когда люди перемещаются от внешнего круга по направлению к внутреннему – это хорошо для компании. Однако, если это происходит слишком быстро, то либо компания имеет дело с очень талантливым сотрудником, либо заблуждается в возможностях этого человека. Бывает важно несколько раз проверить, насколько качественно испытуемый справляется с важными и ответственными, но в то же время разноплановыми заданиями.

Если же сотрудник долгое время задерживается во внешнем круге, видимо что-то идет не так. Необходимо разобраться, что ему мешает развиваться и совершенствоваться. Движение же в обратном направлении – от внутреннего круга к внешнему – говорит либо о серьезной проблеме у сотрудника, либо в неправильном управлении и кадровой политике. Нужно не допускать такого развития событий.

Ранжирование помогает позаботиться о людях, внёсших наибольший вклад в успех организации, и избежать увольнения сотрудников, играющих ключевые роли в создании ПО. С помощью ранжирования легче распределить ограниченные ресурсы и решить, перед кем открыть новые возможности.

Области применения ранжирования:

1) Поощрение заслуженных сотрудников. Компания таким образом отмечает вклад сотрудника в успех компании.

2) Распределение привилегий и ограниченных ресурсов. Как правило, ресурсы (не только денежные, например, расположение рабочего места) ранжируются согласно уровню квалификации (кругу) сотрудника. Важно, чтобы все привилегии не доставались лишь одному кругу, вместе с членами внутреннего круга что-то должны получать и люди сред-

него и внешнего кругов. И всё же основной принцип остаётся в силе: достойны заботы лишь те, кто заботится об успехе продукта.

3) Планирование денежных компенсаций. Рейтинг позволяет спланировать повышение зарплаты, премии и другие материальные компенсации.

4) Планирование кадровой политики. Если компанию покидают люди среднего и внутреннего кругов, то это признак серьёзной проблемы, которую нужно решить. Ранжирование помогает разобраться, где появилась текучка и какой ущерб она наносит группе.

Важно понимать, что ранжирование необходимо использовать очень обдуманно и корректно, и те или иные решения по нему принимать исключительно в соответствии с профессиональным достижениям сотрудника, а не основываясь на личных симпатиях и родственных связях.

4.2. Качество работы и вознаграждение

Вопросы, связанные с поощрением и вознаграждением должны быть увязаны со сложностью выполняемых задач и уровнем квалификации сотрудника. Все эти понятия тесно связаны с эффективностью труда. Рассмотрим ситуации, при которых данная эффективность может быть достигнута.

Модель потока продуктивности

Модель потока продуктивности основывается на соотношении уровня мастерства сотрудника и сложности его работы (см. рис. 4). Если задача слишком простая, то становится скучно и человек не чувствует себя счастливым. Если она становится для него слишком сложной, то он испытывает напряжение и тревогу. Если же возникает умеренное соответствие: не простые, но и не чрезмерно сложные задачи, то мы попадаем в поток, который охватывает достаточно широкий диапазон уровня компетентности и сложности задач.

Необходимо понимать, что из позиции 1 (рис. 4) человек может попасть в зону тревоги, если сложность задач увеличится, а его мастерство не вырастет соответствующим образом (позиция 2). Чтобы вернуться в поток, достаточно любой комбинации роста мастерства и снижения

трудности, но лучше всего двигаться по горизонтали в позицию 3. Аналогично можно повысить мастерство при неизменной сложности работы и переместиться из позиции 1 в позицию 2', где наступает скука. Для входа в поток перед этим специалистом необходимо ставить более сложные нетривиальные задачи, и он будет двигаться в сторону позиции 3. При этом позиция 3 станет новой позицией 1, и весь цикл повторится. Однако очевидно, что в позиции 3 человек приносит гораздо больше пользы компании, нежели в позиции 1. Следует отметить, что нахождение в потоке и постепенное движение вправо вверх ведет к оптимальной продуктивности.

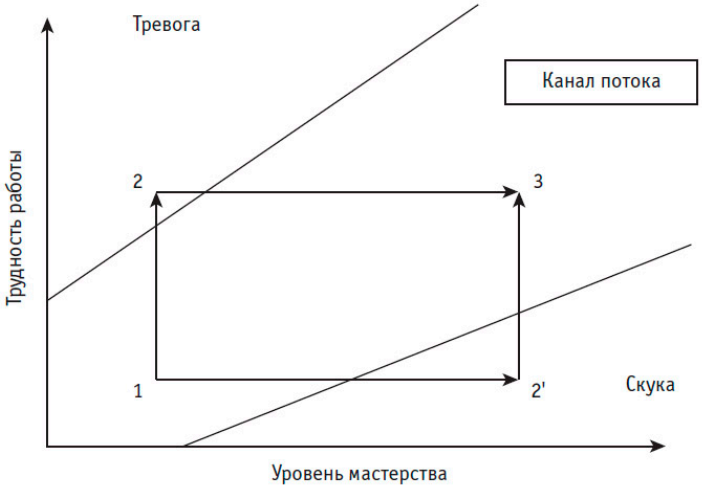

Рис. 4. Модель потока продуктивности

Свяжем понятие вознаграждения с уже рассмотренными понятиями.

Модель, основанная на мастерстве

Часто вознаграждение увязывается с уровнем мастерства, при этом сложность работы не считается доминирующим фактором. В рамках модели *«потока»* этот подход отражен на рис. 5.

Как видно из рисунка, есть зона «адекватного вознаграждения», в которой уровень вознаграждения соответствует объему навыков участ-

ника команды, занятого данной работой. В этой зоне находятся позиции 1 и 3, хотя последней явно соответствует большее вознаграждение, чем первой.

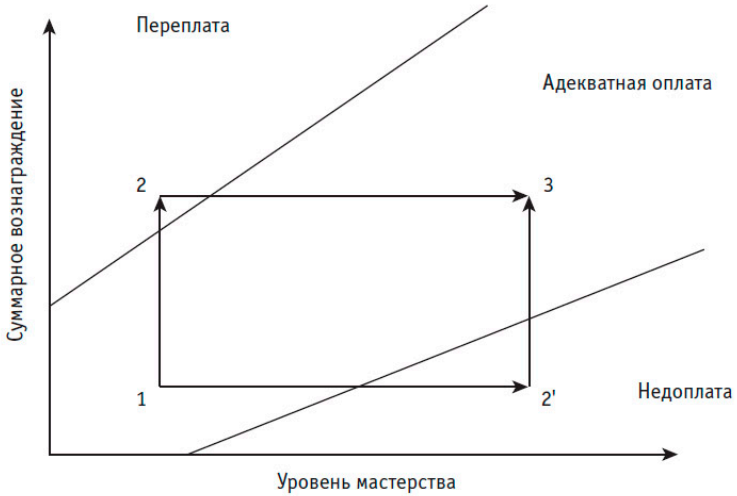

Рис. 5. Модель вознаграждения, основанного на уровне мастерства

В позиции 2 член команды зарабатывает больше, чем следовало бы при его уровне мастерства. Такое положение может свидетельствовать о проблеме, но не обязательно. Например, работник может заниматься более сложной задачей, требующей большего мастерства, чем то, которым он владеет, поэтому он получает соответствующую компенсацию. А может быть, это просто ошибка. В позиции 2' члену команды недоплачивают за его мастерство. Опять же для этого могут быть разные причины. Возможно, работник действительно не полностью загружен работой либо эта работа не требует того уровня мастерства, которым он владеет.

Модель, основанная на задаче

В некоторых компаниях меньше интересуются уровнем мастерства и определяют вознаграждение в соответствии с трудностью задачи (либо в соответствии с мерой ответственности). Такая модель более типична для предпринимательской среды, начинающих компаний (рис. 6).

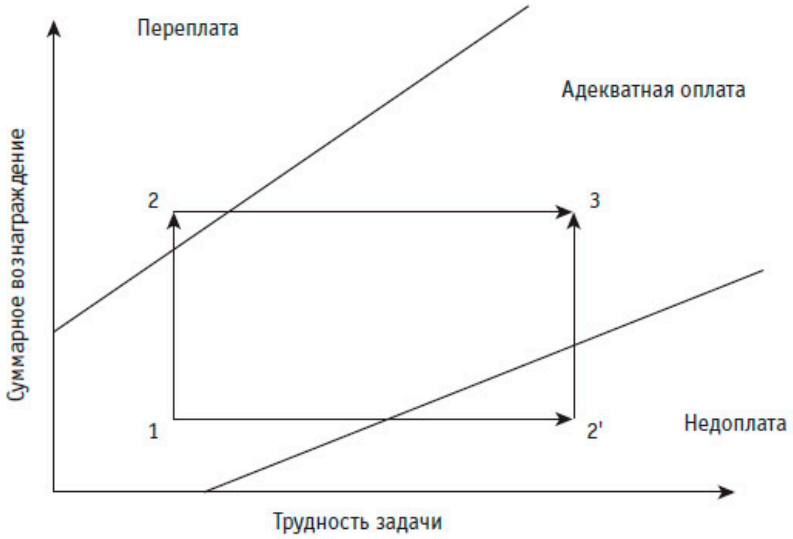

Рис. 6. Модель вознаграждения, основанного на сложности задачи

Позиции 1 и 3 также соответствуют правильному вознаграждению, и в последней вознаграждение выше, чем в первой.

В позиции 2 участник команды получает чрезмерное вознаграждение относительно сложности работы. Это может происходить по ряду причин. Может быть, по ошибке, а может быть, исполнитель этой работы обладает высоким мастерством, не востребованным в данной работе или у него очень хорошие рекомендации.

В позиции 2' участнику команды недоплачивают. Так бывает, когда человек выдвинут на новую и более сложную работу, но соответствующее вознаграждение ему еще не назначено.

В этой модели, как и в предыдущей, попадание работника в положение 2 можно исправить, повысив его квалификацию или поручив более сложное задание при сохранении уровня вознаграждения. Если участники команды находятся в положении 2', то надо увеличить вознаграждение.

В любой модели вход в зону адекватной оплаты за счет снижения сложности задачи, уровня мастерства или величины вознаграждения всегда сопряжен с трудностями. Обычно в компаниях по разработке ПО не приветствуются изменения в сторону понижения.

Конус адекватного вознаграждения

Объединим все рассмотренные ранее модели в одну универсальную. Графически её можно представить с помощью трехмерной диаграммы – «конуса адекватного вознаграждения» (рис. 7).

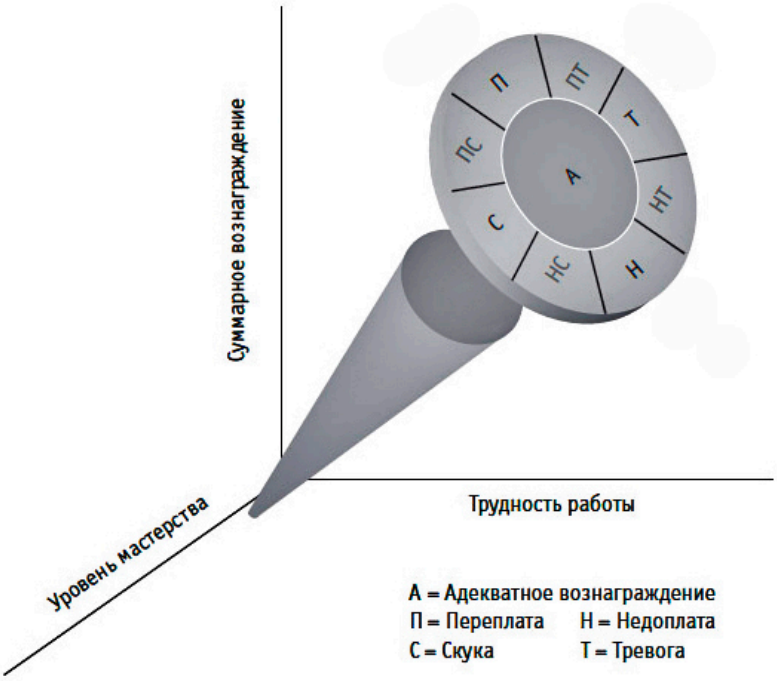

Рис. 7. Конус адекватного вознаграждения

Этот конус иллюстрирует состояния, в которых уровень мастерства, сложность задания и размер вознаграждения находятся во взаимном равновесии. Область внутри этого конуса, помеченная буквой А в центре проекции, соответствует классической ситуации «взаимного удовлетворения». Участник команды не испытывает ни скуки, ни тревоги. Он находится в состоянии потока. Вознаграждение также справедливо:

участнику команды платят не больше, но и не меньше того, что он заслуживает. Он действует с максимально возможной для него продуктивностью, и организация честно расплачивается с ним за это. Следует обратить внимание на расширение конуса вверх: с ростом всех переменных появляется больше свободы для назначения правильного вознаграждения. Это справедливо и на практике: при низких значениях параметров легче допустить ошибку. Все остальные восемь зон соответствуют возможным сценариям «непопадания в конус».

На рис. 8 представлены все девять возможных случаев для нашей конической модели.

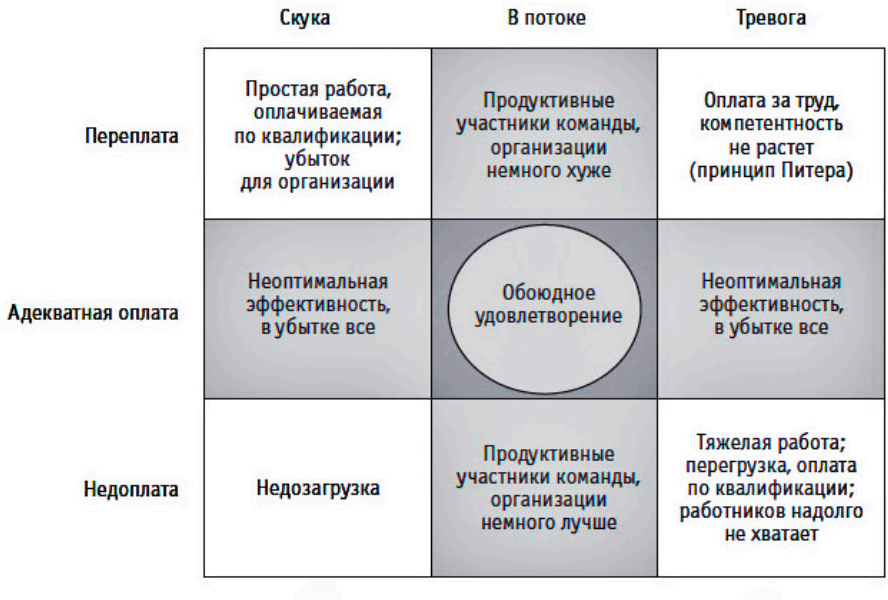

Рис. 8. Девять вариантов в конусе адекватного вознаграждения и рядом с ним

Здесь конус адекватного вознаграждения отображен в центре: достигнуто состояние потока, и нет претензий к вознаграждению. Участник команды доволен и продуктивен, а компания получает от него максимум производительности по справедливой цене.

В среднем ряду слева и справа вознаграждение оказывается справедливым, но участник команды и организация несут потери, потому что продуктивность неоптимальная. В этом случае манипуляции с воз-

награждением не решают проблем, из-за которых участник команды недоволен, а организация не полностью реализует потенциал работников. Это в чистом виде несоответствие сложности/мастерства, и решать проблему надо по этим двум направлениям.

В средней колонке вверху и внизу участник команды находится в потоке, а продуктивность максимальна. Вознаграждение должно быть скорректировано таким образом, чтобы ни работник, ни организация не извлекали из этой ситуации сверхприбыли. Очевидно, что в случае недоплаты это сделать легко. Если же вознаграждение чрезмерно, обычно сохраняют его на том же уровне, пропорционально увеличивая при этом другие параметры (сложность работы и компетенцию).

В левом верхнем квадрате находятся те члены команды, которые получают вознаграждение, соответствующее их квалификации, но выполняют относительно простую работу, которую в организации могли бы за меньшие деньги выполнять менее квалифицированные сотрудники. Это плохой случай: продуктивность по существу не соответствует уровню оплаты, и организация несет убытки упущенных возможностей, поскольку не полностью реализует потенциал своих работников.

Левый нижний квадрат – это те, кому скучно, да и платят им недостаточно. Некоторая квалификация у них имеется, работа относительно простая, а платят немного. Иногда сочетание неудовлетворенности работой и вознаграждением побуждает их искать другую работу; часто они просто прозябают. Этот класс представляет непроизводительную трату человеческого капитала. В эту категорию попадают некоторые государственные служащие.

В правом нижнем углу располагаются самые угнетенные: они выполняют тяжелую работу, для которой их квалификации не вполне достаточно, и при этом им недоплачивают. Такое сочетание нежизнеспособно, поскольку рано или поздно эти люди окончательно измучаются.

Наконец, правый верхний угол представляет тех участников команды, которым платят повышенное вознаграждение за использование их потенциал сверх того, что предполагает квалификация. Какое-то время они могут продержаться, но не долго, если не повысят свое мастерство.

Резюмируя, можно сказать, что четыре угла – это гиблые места, где положение быстро ухудшается или влечет постоянные убытки для участника команды и/или организации. Устойчивы ли промежуточные состояния? Хотя они не столь опасны, как угловые, но с течением времени теряют устойчивость, если не предпринимать никаких действий.

4.3. Корпоративная культура

Понятие «культура» в приложении к созданию программ включает ряд определённых черт и понятий, влияющих на процесс разработки ПО. Культура формируется, прежде всего, принципами и действиями руководящего звена организации.

Как и у отдельного специалиста, так и у команды должно быть представление о предназначении, важности, уровне мастерства и способности решить поставленную перед ней задачу. Если самооценка команды высока, её мотивация и работоспособность достаточны, чтобы с высокой производительностью труда создавать высококачественные продукты. И наоборот, если самооценка низка, команда будет работать ниже своих способностей, даже если она состоит из талантливых людей! В определённом смысле культура может как повышать, так и снижать общую работоспособность коллектива.

Высокая культура разработки ПО, в частности, имеет огромное значение при освоении новых возможностей или для выхода из затруднительной ситуации.

Если в компании развита культура создания качественных продуктов, весьма вероятно, что внутреннее стремление и приверженность к этой практике не иссякнут и в будущем.

Лучший способ создать и воспитать корпоративную культуру – начать с приверженности определённым принципам и культивирования некоторых манер поведения, после чего группа какое-то время должна работать по новым правилам. Действия группы в соответствии с этими принципами и будут факторами, формирующими её культуру.

Рассмотрим некоторые из ***принципов развития корпоративной культуры***:

1) ***Четко определять цели***. Эти цели, сформированные руководством компании, должен понимать и принимать каждый сотрудник. Хорошим примером является цель компании Apple: «*Мы хотим создавать лучшие продукты и делать наших пользователей самыми счастливыми*».

2) ***Помнить историю и отмечать успехи***. Необходимо обязательно отмечать успех и передавать историю компании новым работникам. Это стимулирует формирование культуры стремления к успеху, создания новых продуктов и заставляет быть экспертом в выбранной области.

3) ***Культивировать элитарность***. Строгий отбор при приёме на работу позволяет многим считать привилегией принадлежность к числу технических специалистов компании. Культивирование окружения, построенного на привилегиях, позволяет поддерживать высокий боевой дух и снижать текучесть кадров. Элитарность позволяет воспитать такие черты культуры, как приверженность общему делу, стремление делать заметные вещи и участвовать в решении нестандартных задач.

4) ***Поддерживать дух соперничества.*** Когда каждый член команды и каждый коллектив компании старается быть лучим в выполнении той или иной задачи, команда делает всё «возможное и невозможное» для достижения успеха.

5) ***Соц. пакет, досуг и отдых.*** Когда компания предоставляет хорошие условия и соц. пакет, каждый работник чувствует свою значимость. Совместные спортивные и культурные мероприятия также повышают корпоративную культуру в коллективе.

5. Инструментальные программные средства. Контроль качества

При создании качественного программного продукта нужна слаженная работа команды, формирование которой, а также роли и взаимодействия внутри неё мы рассмотрели в предыдущих главах. Другой важный момент связан непосредственно с создаваемым программным

средством, а именно с написанием программного кода. Должна быть обеспечена целостность программного кода, а также его работоспособность и отсутствие ошибок.

5.1. Инструментальные средства управления кодом

В течение цикла разработки требуется проверять, обновлять, контролировать и пересматривать изменения в исходном коде. С ростом количества людей, работающих над проектом, требования становятся более критичными. Отсюда возникает необходимость использования инструментария по управлению кодом.

Продукты по управлению исходным кодом, прежде всего, системы управления версиями (например, SVN, VSS, CVS и проч.), выполняют такие функции, как:

- хранение файлов, их прошлых, настоящих и будущих версий,
- отслеживание историй изменения (происходит фиксация даты, времени, имени пользователя, его комментариев),
- группирование файлов (по подпроектам, подсистемам, пакетам и т.д.),
- маркировка файлов (как правило, связанных конкретным выпуском программного продукта),
- блокировка и разблокировка файлов (при одновременном использовании).

К основным документам, хранящимся в системе контроля версий, можно отнести:

- исходные файлы,
- файлы библиотек,
- результаты компиляции,
- сценарии компоновки,
- файлы для установки программ,
- пользовательскую документацию,
- спецификации проекта,
- тесты (задания и сценарии, модули),
- доп. инструменты.

Помещение данных документов в систему управления исходным кодом дает ряд преимуществ. Во-первых, работая над проектом и подключившись к системе, каждый сотрудник всегда может найти практически любой файл: разработчик – необходимы классы, тестировщик – тесты и требуемый для тестирования код, технический писатель – функциональные спецификации и т.д. Во-вторых, каждый из них имеет возможность обращаться к любой из версий, видеть историю изменения, ставить пометки, использовать данные. В-третьих, всегда можно отследить критическую ситуацию, сделать «откат», свести, исправляя, противоречащие друг другу фрагменты кода. И, наконец, в четвертых, централизация файлов в системе управления исходным кодом обеспечивает простое резервное копирование всего проекта.

Одна из главных задач в управлении проектом по разработке ПО – это контроль сложности проекта, в том числе управление исходным кодом и конфигурацией.

Структура исходного кода

Как правило, все файлы, используемые при разработке, бывают строго структурированы и отсортированы по каталогам (пакетам):

- ***Project*** – файлы, непосредственно относящиеся к проекту,
- ***Enviroment*** – файлы среды разработки,
- ***Imports*** – файлы, импортированные из других проектов,
- ***Libs*** – файлы библиотек, используемые в проекте.

В ***Projects*** содержатся файлы, содержащие классы проекта, а также файлы, необходимые для сборки, тестирования и написания документации к продукту. Внутри пакета ***Projects*** также имеется своя иерархия и структура (см. табл. 6).

Таблица 6. Примерная структура основного пакета проекта

Структура Projects	**Описание**
$/ProjectName/	Файлы, относящиеся к проекту
$/ProjectName/Branch/	Различные направления в разработке
$/ProjectName/Parts/	Совместно используемые файлы, входящие в проект

$/ProjectName/Parts/Src/	Исходный код для Parts (при необходимости совместно используется /ProjectName/Src)
$/ProjectName/Parts/Doc/	Исходные файлы документации
$/ProjectName/Parts/Help/	исходные файлы справочной системы
$/ProjectName/Parts/Install/	Исходный код программы установки
$/ProjectName/Parts/Patch/	Исходный код вставок
$/ProjectName/Parts/Setup/	Исходный код установщика
$/ProjectName/Parts/Samples/	Исходный код с примерами
$/ProjectName/Parts/Tests/	Исходный код тестов, тестовые задания и т.д.
$/ProjectName/Project/Output/	Область для программ, созданных в других проектах
$/ProjectName/Project/Src/	Исходный код проекта (при необходимости используется совместно с /Parts/Src)
$/ProjectName/Project/Doc/	Файлы документации к проекту (используется совместно с /Parts/Doc)
$/ProjectName/Project/Help/	Файлы справочной системы проекта (используется совместно с /Parts/Help)
$/ProjectName/Project/Imports/	Импорт (совместно используется с /Imports)
$/ProjectName/Project/Install/	Файлы для установки проекта (используется с /Parts/Install)
$/ProjectName/Project/Samples/	Примеры для проекта (используется совместно с /Parts/Samples)
$/ProjectName/Project/Tests/	Тестовые задания, тестовые сценарии (используется совместно с /Parts/Tests)

В ***Environment*** ($/Env) хранятся файлы, используемые командой разработчиков, но не являющиеся частью данного проекта. Это все, начиная с инструментов и утилит и заканчивая стандартами создания кода и шаблонами для проекта. Папка Environment содержит файлы среды, описывающие её с точки зрения разработчика, а также с точки зрения тестирования и документации. Примерный список подкаталогов, которые могут быть в этом разделе хранилища исходного кода, приведен в табл. 7.

Таблица 7. Примерная структура пакета Environment

$/Env/Dev/	ПО среды разработки и инструментальных средств
$/Env/Dev/Bin	Исполняемые модули (подкаталог для каждого инструмента)
$/Env/Dev/Src	Исходный код для этих инструментов (подкаталог для каждого)
$/Env/Dev/Doc	Документация для этих инструментов (подкаталог для каждого)
$/Env/Dev/Etc	Прочие файлы
$/Env/Test/	Инструментальные средства и файлы для тестирования
$/Env/Test/Bin	Исполняемые модули
$/Env/Test/Src	Исходный код и документация для этих инструментов
$/Env/Test/Doc	Документация по среде тестирования
$/Env/Test/Etc	Прочие файлы
$/Env/Documentation/	Документация по среде проекта
$/Env/Documentation/ Templates	Шаблоны проекта, шаблоны документации и справочники по стилям
$/Env/Documentation/ Plans	Планы и спецификации проекта, тестовых заданий и документации
$/Env/Documentation/ Process	Технологические документы проекта
$/Env/Etc	Прочие файлы, не вошедшие в предыдущие категории

Компоновочная система и сборка

В современных компаниях по разработке ПО сценарий сборки продукта, как правило, выполняется с помощью определенной компоновочной системы. Сценарий должен определять нужные для сборки продукта файлы из системы управления исходным кодом. Эта информация включает как сами средства компоновки, так и исходные файлы, библиотеки, заголовки и другие необходимые компоненты.

Надёжная автоматическая система компоновки – необходимое условие успешной разработки.

Сборка является результатом компиляции всего исходного кода продукта. Для корректного построения ПО интеграция должна быть обеспечена на самом элементарном уровне – на уровне исходного кода. Целостность исходного кода обязана быть совершенной: ошибки компиляции и компоновки недопустимы. В сложных проектах совершенства добиться тяжело из-за массы связей между модулями исходного кода. Однако регулярно собирать свою программу можно и нужно.

Способность собирать ПО является определяющей для поставки программ в срок. Одна из наиболее часто возникающих проблем при создании ПО – заставить все части работать вместе. Если о ней забыть до окончания проекта, то потом решение проблемы займёт недели или месяцы работы. В худшем случае потребуется переопределение каких-то API и функций. А это, естественно, означает появление никем не запланированных задержек.

При регулярном создании сборок разработчики могут проверять интеграцию кода. Интерфейсы API, файлы заголовков, параметры, типы данных и макросы – все должны быть в полностью рабочем состоянии, иначе сборка пройдёт некорректно. Сбой при сборке заставит разработчиков общаться друг с другом и при необходимости изменять программу.

Очень важно очередной сборке присваивать свой уникальный номер. Номер сборки – это монотонно возрастающая целая величина, ни разу не повторяющаяся в истории создания приложения. Номер увеличивается на базовых уровнях, этапах и в каждом последующем выпуске ПО.

Дополнительные инструментальные средства

К дополнительным инструментальным средствам, широко используемым в компаниях по разработке ПО, можно отнести:

- отладчики,
- средства анализа производительности,
- средства написания сценариев и автоматизации тестирования.

Устранение проблем, неисправностей, ошибок

Для оперативного устранения проблем и неисправностей в проекте необходимо знать следующее:

- текущее состояние проблемы – открытая или закрытая;
- дата возникновения, изменения и решения;
- текстовое описание проблемы;
- номер выпуска/сборки программы, в которой обнаружена проблема;
- имя человека, описавшего проблему:
- имя человека, работающего над проблемой в настоящее время;
- состояние проблемы – расследуется, требуется больше информации, ожидается внешнее событие, решена и т.д.;
- приоритет проблемы – низкий, средний, высокий;
- выпуск, в котором присутствует проблема;
- статус процедуры контроля качества;
- количество попыток неуспешного решения проблемы;
- список изменений к отчёту о проблеме или неисправности.

На рис. 9 и 10 приводятся интенсивности возникновения и устранения ошибок на разных этапах проекта.

На рисунках мы наблюдаем результаты своевременного устранения ошибок. Если же такого не происходит – количество ошибок будет интенсивно возрастать, что может быть критичным как для завершения этапов, так и для выполнения всего проекта.

Часто проблемы, возникающие при написании программного кода (громоздкость, противоречие одних фрагментов другим и тому подобное), связанны либо со сменой одних инструментальных средств другими на середине пути, либо с игнорированием тех или иных инстру-

ментальных средств как таковых. Поэтому для проекта бывает очень важно правильно определиться с набором инструментальных средств. Более того, даже для небольших проектов просто необходимо использовать систему управления версиями, а также компоновочную систему и т.д. Зачастую бывает лучше дойти до завершения проекта с тем инструментарием, который уже используется, нежели менять его на середине пути.

Рис. 9. Интенсивность возникновения и устранения ошибок в начале проекта

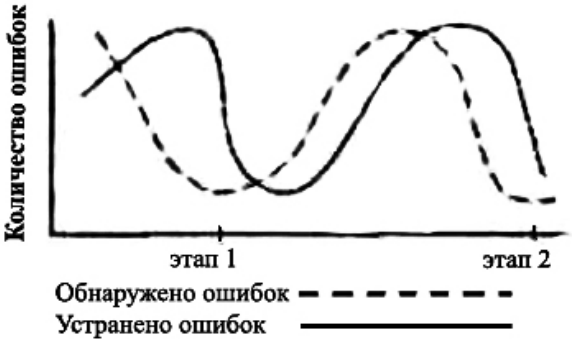

Рис. 10. Интенсивность возникновения и устранения ошибок в конце этапов

Проблемы управления исходным кодом обычно бывают связаны со следующим:

1) Неправильной структурой проекта. Это бывает особенно критично с ростом проекта. Необходимо заранее спланировать всю систему в соответствии с нуждами проекта, предусмотрев возможность выпуска нескольких версий.

2) Неверным или некорректным содержанием. Необходимо помнить, что исходный код требует непрерывного контроля над изменениями.

3) Конфликтами ключевых файлов. Типичным бывает случай, когда одному разработчику выдается файл, а другой не может его взять для внесения важных изменений. Это может существенно замедлить работу над проектом. Поэтому необходимо заранее предусмотреть способ быстрого внесения критичных исправлений и изменений. Можно либо разделять важную и ключевую информацию на несколько файлов, основываясь на логических подсистемах, компонентах и классах, либо применять систему управления исходным кодом, поддерживающую возможность совместного применения и последующего слияния одних и тех же файлов.

4) Долгим распознаванием фрагментов исходного кода. Никогда не стоит пренебрегать ведением комментариев и описательной документации, т.к. один и тот же код обязательно будут использовать несколько членов команды, которые должны быстро понимать его значение.

Одной из самых приоритетных задач является ***обеспечение целостности данных***. Очень важно бывает определить правила обеспечения целостности данных, в большинстве случаев основанных на внутреннем процессе разработки. Необходимо своевременно исправлять ошибки и менять их статус, а также вводить корректные значения в различные поля (информация о выпуске, сборке, этапе и т.д.). Какими бы ни были внутренние методы проверки целостности, не нужно допускать хранения недостоверных или устаревших данных, иначе команда разработчиков будет присваивать данным любые значения, а вся система перестанет внушать доверие и станет бесполезной. Лучший способ избежать проблем с целостностью данных – это убедиться в

том, что вся команда осознает важность этих данных и может обнаруживать и решать проблемы самостоятельно. Собственная мотивация заработает хорошо, если продемонстрировать реальную значимость этих данных для команды. Нужно также не забывать периодически пересматривать данные и обсуждать результаты с командой.

5.2. Основы системы контроля качества

При недостаточном учете контроля качества могут быть сорваны сроки проекта или продукт может получиться таким, что потребители не смогут им пользоваться в полной мере. Любая компания по разработке ПО должна эффективно балансировать между качеством продукта и временем его представления на рынке.

Создание качественного продукта основывается на четырёх простых принципах:

- тестирование продукта осуществляется параллельно с процессом его разработки;
- качество продукта улучшается регулярно при завершении каждого этапа разработки;
- тестирование необходимо максимально автоматизировать;
- качество является частью культуры и технологии.

Проект создается в ограниченное время, поэтому поиск и устранение проблем и ошибок – условие обязательное. При этом просто необходимо тестировать функции сразу же после их создания. Этот процесс и называется параллельным тестированием. Чтобы его правильно и своевременно осуществлять, нужно иметь средства автоматизированного тестирования. Автоматизация предоставляет ряд преимуществ:

- сокращение внутреннего цикла тестирования;
- сокращение потребности в персонале;
- проверка изменений, внесенных в последний момент;
- обеспечение полноты тестирования для новых выпусков.

Стабилизация и интеграция

Примерно через каждые 4-6 недель команда должна отводить определенное время (в зависимости от сложности проекта) на тестирование, стабилизацию и интеграцию функций, завершённых к данному момен-

ту. Не нужно работать над новыми функциями и кодом, пока нет уверенности в стабильности того, что уже построено.

Периоды стабилизации и интеграции также позволяют сопоставить фактическое продвижение проекта с запланированным. Если проект хорошо спланирован, команда будет точно знать, на какой неделе какая функция будет завершена. Укладывается ли все в график? Можно ли использовать определённые функции в намеченный срок? Эта информация необходима для того, чтобы не дать проекту выйти из колеи.

Виды тестирования

Существует множество видов и классификаций тестирования, однако можно выделить наиболее часто встречающиеся виды:

1) Входные тесты – тесты, проверяющие ПО перед подтверждением внесенных изменений. Они позволяют разработчикам проверить функции в локальной сборке перед помещением кода в основную базу. Входные тесты предотвращают внесение критических ошибок в ежедневную сборку и сбой базисного теста.

2) Unit-тесты (модульные тесты) – тесты, проверяющие на корректность отдельные модули программы. Unit-тесты позволяют «изолировать» отдельные части программы и показать, что по отдельности данные части работоспособны.

3) Ежедневные базисные тесты – способ реализации стратегии «тестировать как можно раньше». Базисные тесты – это основной набор автоматизированных регрессивных тестов, проверяющих, что ключевые функции продукта работают. Они обеспечивают создание работоспособной сборки.

4) Тесты для стабилизации и интеграции – тесты, проверяющие интеграцию функций через определенные интервалы времени.

5) Бета-тестирование – тестирование, происходящее на стороне заказчика. Это возможность дать клиентам проверить и оценить разработанное ПО до его выпуска и применения в реальной рабочей среде.

К ключевым областям проекта, подлежащим тестированию, можно также отнести:

- тестирование производительности,
- тестирование совместимости,
- тестирование функциональных возможностей.

Типичные проблемы контроля качества

Нехватка ресурсов является наиболее частой проблемой системы контроля качества. При этом никогда не стоит пренебрегать штатным составом отдела качества (при увеличении оборотов компании и объема работы должен увеличиваться не только отдел разработки, но и отдел качества).

Недостаточная подготовка также является типичной проблемой в отношении контроля качества. Для решения, а лучше упреждения данной проблемы в команде должны быть квалифицированные специалисты, отвечающие за качество, кроме того, у них должно быть современное оборудование и ПО.

Отсутствие автоматизации, безусловно, приводит как к увеличению времени, отводящегося на контроль качества, а соответственно, всей разработки, так и к невозможности успешного завершения проекта.

Неправильная расстановка акцентов может также стать существенной проблемой. Следует расставить приоритеты в тестировании функций: не тратить слишком много времени на мелкие детали, которыми зачастую можно пренебречь, но при этом необходимо не откладывать решение проблем по ключевым функциям.

6. Жизненный цикл ПО. Управление рисками

В реализации любого проекта существует ряд контрольных точек – моментов разработки, когда осуществляется подведение промежуточных итогов, осмысление достигнутого, возможные корректировки и дальнейшее принятие решения. Определение контрольных точек является важным элементом планирования как времени, так и ресурсов. Понятие жизненного цикла ПО связано с систематизацией ресурсов, методов и результатов на основных этапах разработки ПО.

6.1. Жизненный цикл ПО

В самом общем виде жизненный цикл ПО можно изобразить следующим образом (рис. 11):

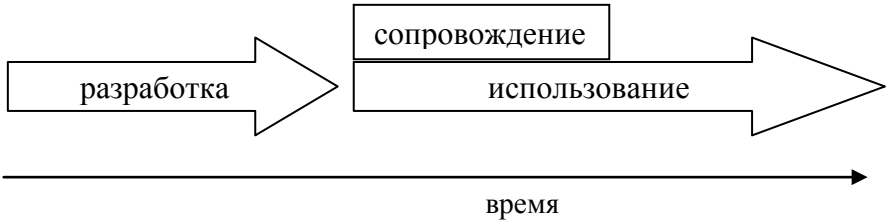

Рис. 11. Жизненный цикл ПО в общем виде

По окончании стадии разработки ПО происходит непосредственное использование данного продукта заказчиком, оно обычно некоторое время сопровождается поддержкой разработчиков.

Существует ряд общепринятых моделей для описания жизненного цикла ПО.

Одной из самых распространенных является модель, состоящая из двух фаз: разработка и сопровождение, и ряда этапов (рис. 12).

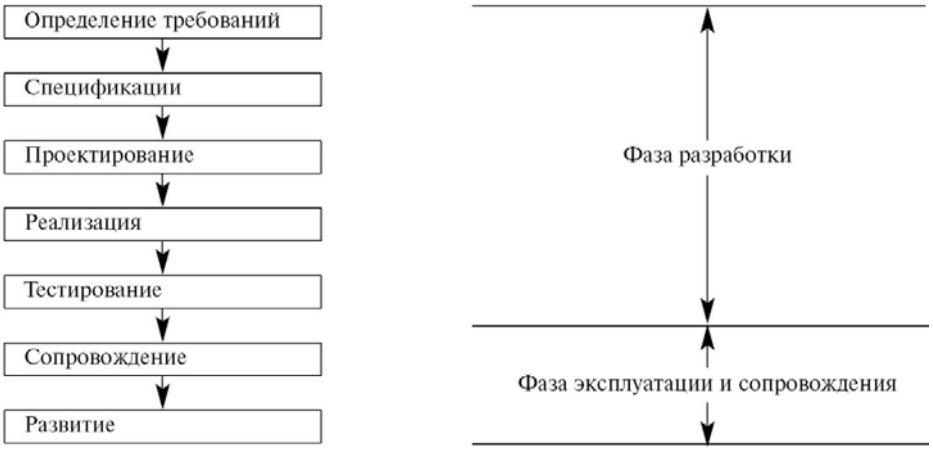

Рис. 12. Общепринятая двухфазовая модель жизненного цикла ПО

По предложенной модели этапы «Определение требований», «Спецификации», «Проектирование», «Реализация и тестирование» отнесены к фазе разработки; этапы «Сопровождение» и «Развитие» – к фазе эксплуатации и сопровождения.

Самым начальным этапом является «Определение требований» – здесь происходит постановка и формулировка задачи, а также описание ожидаемых функций и ограничений системы. На этом этапе менеджер проекта взаимодействует с клиентом, принимается решение о создании системы (подробнее о требованиях к системе в см. в главе 7).

На этапе «Спецификации» происходит фиксация обозначенных требований в виде спецификаций системы – законченного описания поведения системы. На данном этапе активизируются функции архитектора и проектировщика системы. Спецификации должны быть приведены в соответствие целям проекта, обладать полнотой однозначностью и непротиворечивостью.

Разработка проектных решений, отвечающих на вопрос о том, как должна быть реализована система, чтобы она могла удовлетворять специфицированным требованиям, выполняется на этапе «Проектирование». Поскольку сложность системы в целом может быть очень высока, главной задачей этого этапа является последовательная декомпозиция системы до уровня очевидно реализуемых модулей или процедур. В данном этапе принимают участие менеджер проекта, архитектор, проектировщик системы, руководитель команды и ведущие разработчики.

На этапе «Реализации» или кодирования каждый из модулей, выявленных при декомпозиции, программируется на наиболее подходящем для данного приложения языке. С точки зрения автоматизации этот этап традиционно является наиболее развитым. Основные действующие лица этапа – руководитель команды и все разработчики. «Реализация» также включает в себя интеграцию и сборку.

В общепринятой модели этап разработки завершается этапом тестирования, на практике же данные этапы постоянно пересекаются, и в реализации этапа «Тестирования» задействована как вся команда тестировщиков, так и команде разработчиков.

Фаза эксплуатации и сопровождения включает в себя деятельность по обеспечению нормального функционирования программного продукта, в том числе фиксирование скрытых во время исполнения программ ошибок, поиск их причин, исправление, повышение эксплуатационных характеристик системы, адаптацию системы к окружающей

среде, а также, при необходимости, и более существенные работы по совершенствованию системы. Все это дает право говорить об эволюции системы. В связи с этим фаза эксплуатации и сопровождения разбивается на два этапа: «Сопровождение» и «Развитие». В ряде случаев на данную фазу приходится большая часть средств, расходуемых в процессе жизненного цикла программного обеспечения.

Приведенная общепринятая модель является наглядной с точки зрения обозначения основных этапов разработки ПО, однако она не совсем объективна, т.к. лишь очень простые задачи могут проходить все этапы без каких-либо итераций (возвратов на предыдущие шаги производственного процесса). При создании ПО иногда бывает необходимо перепроектирование, переделка спецификаций и т.д. При разработке больших нетрадиционных систем итеративность возникает регулярно на любом этапе жизненного цикла как из-за допущенных на предыдущих шагах ошибок и неточностей, так и из-за изменений внешних требований к условиям эксплуатации системы. Поэтому для описания жизненного цикла ПО часто используют классическую итерационную модель (рис. 13).

Стрелки, идущие вверх, обозначают возвраты к предыдущим этапам, квалифицируемые как требование повторить этап для исправления обнаруженной ошибки. Может показаться странным переход от этапа «Эксплуатация и сопровождение» к этапу «Тестирование и отладка». Дело, однако, в том, что претензии заказчика, предъявляемые в ходе эксплуатации системы, часто представляются в такой форме, что нуждаются в перепроверке. Чтобы понять, о каких ошибках идет речь в претензии, разработчикам полезно предварительно воспроизвести пользовательскую ситуацию у себя, т.е. выполнить действия, которые обычно относят к тестированию.

Классическая итерационная модель абсолютизирует возможность возвратов на предыдущие этапы. Итеративное наращивание имеет целью повышение гибкости системы, обеспечение возможности ее адаптации к меняющимся требованиям к программе и условиям развития проекта.

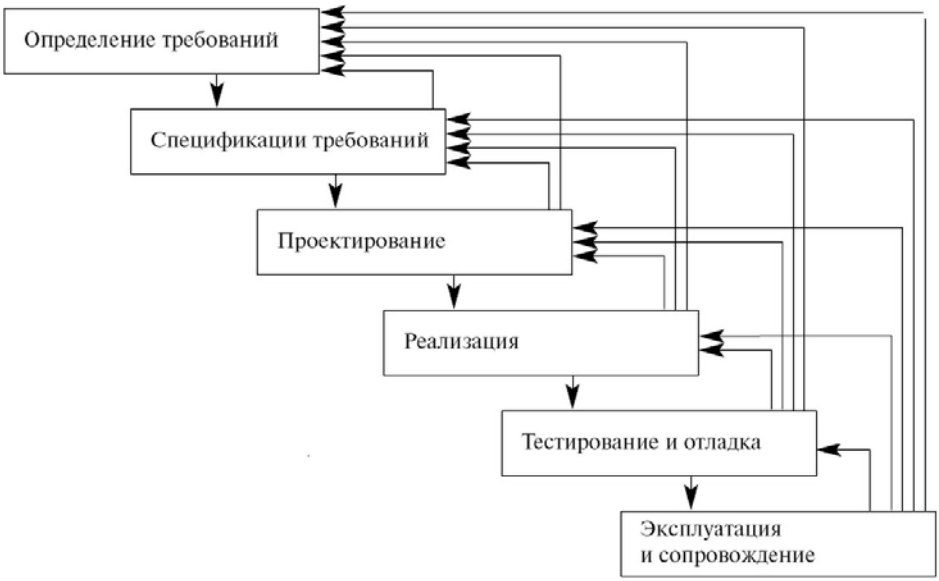

Рис. 13. Итерационная модель жизненного цикла ПО

Рассмотрим ещё одну модель жизненного цикла ПО – каскадную (рис. 14). В каскадной модели в отличие от итерационной минимизированы возвраты.

Характерными чертами каскадной модели являются:

- завершение каждого этапа проверкой полученных результатов, с целью устранить как можно большее количество проблем, связанных с разработкой изделия;

- циклическое повторение пройденных этапов (как в классической итерационной модели).

В каскадной модели уточняются понятия этапов, некоторые из них структурируются (спецификация требований и реализация).

В соответствии с каскадной моделью завершение этапа «Определения требований» включает фиксацию их в виде специальных документов, называемых обзорами того, что требуется от системы (описание функций), а спецификация требований к программам – подтверждением выполнения зафиксированных в обзорах функций в планируемых к реализации программах. Кроме того, подтверждение предполагается и

на первом этапе после определения требований. Это отражает тот факт, что полученные требования необходимо согласовывать с заказчиком.

Результат этапа «Проектирование» верифицируется, т.е. проверяется, обеспечивают ли принятая структура системы и её механизмы выполнимость специфицированных функций.

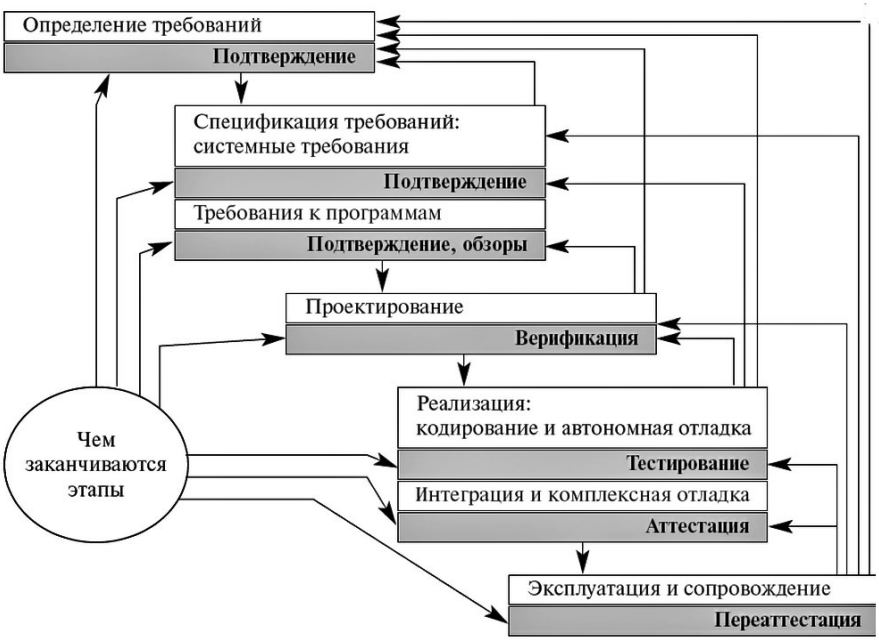

Рис. 14. Каскадная модель жизненного цикла ПО

Этап «Реализация» контролируется путем тестирования компонентов, а после этапа «Интеграция и комплексная отладка» в некоторых случаях проводится аттестация, т.е. проверка-фиксация фактически реализованных функций системы.

Сходные понятия «Верификация» и «Аттестация», приписанные к различным этапам, следует по-разному характеризовать. Верификация отвечает на вопрос, правильно ли создана программная система, а аттестация – правильно ли работает программная система.

Верификация проверяет соответствие программного обеспечения *спецификациям*. На уровне проверки продукта появляется дополнительный критерий правильности: соответствие ожиданиям заинтересо-

ванных в проекте лиц. Отсюда следует, что если верификация может осуществляться силами команды разработчиков (непременно задействуются роли менеджера, архитектора, проектировщиков подсистем и т.д.), то для проведения аттестации обязательно привлекаются внешние специалисты и именно они дают мотивированное заключение о пригодности или непригодности предлагаемого изделия для пользователей.

6.2. Управление рисками

Понятие риска обычно связывается с негативным влиянием на какую-либо деятельность. Применительно к программным проектам рисками называют неопределенные события, негативно, позитивно влияющие или не влияющие (нейтральные) на ход развития проекта. Важно знать, что к ним нужно готовиться заранее. Управление рисками проекта (Project Risk Management) включает в себя процессы, обеспечивающие планирование рисков, их идентификацию, анализ, разработку откликов и контроль в течение жизненного цикла проекта.

Риск невыполнения проекта может быть связан, например, с изменением рыночной конъюнктуры. Ненадежность подрядчиков – еще один пример проектного риска. Кроме того, есть чисто внутренние причины рисков: сбои в используемом окружении (программном и техническом), неточность предъявляемых требований, ненадежность кадров (принятый на ключевую роль сотрудник может отказаться от контракта в самый неподходящий момент).

Чтобы снизить влияние рисков на развитие проекта, должен быть разработан специальный план, называемый далее планом *управления рисками*. Содержание этого плана – *идентификация* рисков данного проекта и мероприятия, снижающие зависимость его от рисков. При составлении плана *управления рисками* необходимо быть совершенно уверенным в том, что рассматриваются только подлинные риски, а не известные проблемы или условия, т.е. причины беспокойства за проект, которые не влияют на достижение целей проекта. Таким образом, нужно различать:

1) ***причины риска*** – определенные события или обстоятельства в проекте или в его окружении, которые вызывают неопределенность;

2) ***риски*** – неопределенные события или обстоятельства, возникновение которых может привести к воздействию на процесс достижения целей проекта:

а) к негативному воздействию, если в результате траектория проектной деятельности выходит из области допустимости;

б) нейтральному воздействию, если траектория не выходит из этой области;

в) к позитивному воздействию, когда новая траектория не только остается допустимой, но и по разным причинам оказывается предпочтительнее других траекторий, которые могли бы реализоваться, если бы рисковая ситуация не возникла;

3) ***последствия*** — незапланированные отклонения траектории выполнения проекта, возникшие в результате того, что событие или обстоятельство, квалифицированное как риск, имело место быть.

Первая стадия составления плана управления рисками при любой методологии – идентификация рисков.

Вторая стадия – выставление приоритетов рискам. В результате выставления приоритетов определяются риски, которые будут отслеживаться в проекте. Остальные риски не отслеживаются, но игнорируются обоснованно, так как реально в проекте можно отследить лишь ограниченное количество рисков, а потому нужно выбрать наиболее существенные из них.

На третьей стадии продолжается работа с полученным списком. Устанавливается возможность влияния на риски, т.е. на причины рисков и на оказываемое ими воздействие на проект.

Влияние на риски классифицируется следующим образом:

1) Исключение риска (avoid). Некоторые рискованные ситуации можно исключить полностью. Например, чтобы увольнение работника с ключевой ролью не сильно сказалось на продолжении развития проекта, целесообразно с самого начала предложить для исполнения этой роли двух человек сравнимой квалификации. В начале проекта их дис-

куссии полезны для выработки объективных решений, а если один из них откажется от контракта, второй все еще сможет продолжать дело. К сожалению, дублирование не может быть рекомендовано для исключения всех рискованных ситуаций.

2) Переключение риска (transfer). Это частный случай исключения, когда риск переносится из сферы влияния проекта на окружение. Например, если рыночная ценность создаваемого изделия кажется сомнительной, для его разработки комфортнее договориться с заказчиком об авансовых платежах, тем самым заставив его самого решать задачу преодоления риска и освободив от этого разработчиков (возможно, за счет снижения оплаты проекта). К этому уровню относятся все варианты контрактных соглашений, но не только они. Оперируя рисками, всегда нужно стараться предусмотреть способы переключения.

3) Уменьшение риска (mitigate). Если риск не может быть исключен, можно постараться уменьшить вероятность его появления на практике (оперирование причинами). Например, для снижения вероятности увольнения сотрудника следует предугадать причины поступка и постараться создать для сотрудника более комфортные условия (повысить заработную плату, создать льготы и т.п.) Нужно, чтобы подобные решения делались не в ответ на заявление об увольнении, а заранее. Это сохранит определенную стабильность в коллективе, которая сама по себе является методом уменьшения риска.

4) Предупреждение ущерба от риска (accept). Когда не получается удовлетворительно уменьшить риск, следует подготовиться к встрече с ним так, чтобы минимизировать потери, т.е. снизить вероятность негативного воздействия и уменьшить само воздействие (оперирование последствиями). Если это удается, то продолжение проекта во многих случаях оказывается успешным. В примере с увольнением следует как можно скорее найти замену данному сотруднику. Естественно, время выполнения проекта увеличится (в частности, потому что нового человека придется вводить в курс дела), но работа все-таки будет продолжена.

Для всех рискованных ситуаций планом управления рисками предусматриваются мероприятия на указанных уровнях в различной комбинации, возможно, дополненные более тонкой реакцией на возникновение риска.

Проект с большой долей новаций – всегда рискованное предприятие. По этой причине для него план управления рисками является обязательным и должен быть максимально детализированным в части учета неверных решений, ошибок в оценке ситуаций и т.п.

7. Определение, формулирование и трассировка требований

Каждый член команды разработчиков должен чётко представлять, какую программу нужно создать, для чего она предназначена и каковы её возможности. Проще всего добиться этого понимания с помощью чётко определённого и строго контролируемого набора требований.

Не менее важна возможность улучшения программного продукта и переработки некоторых его фрагментов. Проект должен допускать постепенное улучшение программы вплоть до добавления одних функций и удаления других. Эти две потребности: строгий контроль и свобода развития – часто выглядят взаимоисключающими, однако важен именно их баланс.

7.1. Требования к программному изделию

Требования к программному изделию – основа любого программного проекта. Они формируют как проектное задание, так и развитие проекта в целом. Качество создаваемого программного обеспечения во многом определяется тем, насколько оно удовлетворяет требованиям.

Требованиями к ПО можно считать совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации. Понятие ***требований*** в наиболее общем виде сводится к двум следующим важным аспектам:

- средства программного изделия, в которых нуждается пользователь для решения своих проблем или достижения определенных целей;

- характеристики программного изделия, которыми должна обладать система в целом или ее компонент, чтобы удовлетворять соглашениям, спецификациям, стандартам или другой формально установленной документации.

Рассмотрим процесс управления требованиями, который позволяет их сбалансировать.

В начале работы над программным продуктом необходимо добиться простого и ясного видения проблемы, при котором задачи и приоритеты проекта стали бы очевидными для всех его участников; важно объединить их усилия и гарантировать, что проектная команда будет работать сообща. Залогом хорошего видения проблемы является центральная идея проекта, которая сплотит группу и даже всю компанию воедино. Наиболее яркими подобными примерами являются центральные идеи таких компаний, как например, Microsoft или Apple при разработке наиболее успешных своих продуктов (ОС Windows, iPod или iPhone).

Когда установлено общее видение проекта и достигнуто понимание пользовательских проблем, необходимо переходить к определению требований.

Существует множество различных типов требований и различных уровней их детализации.

Один из лучших способов дать чёткое описание набора требований к проекту – представить его в виде схемы. Самый высокий уровень схемы занимают общие требования. Они объединяют совокупности частных требований, которые можно обсуждать, оценивать, сравнивать и утверждать как единое целое.

Говоря о типах требований, в общем случае их можно классифицировать по трем уровням: ***бизнес-требования, требования пользователей и функциональные требования.*** Кроме того, каждая система имеет свои ***нефункциональные требования***. Модель на рис. 15 иллюстрирует способ представления этих типов требований. Как и все модели, она не

полная, но схематично показывает организацию требований. Овалы обозначают типы информации для требований, а прямоугольники – способ хранения информации (документы, диаграммы).

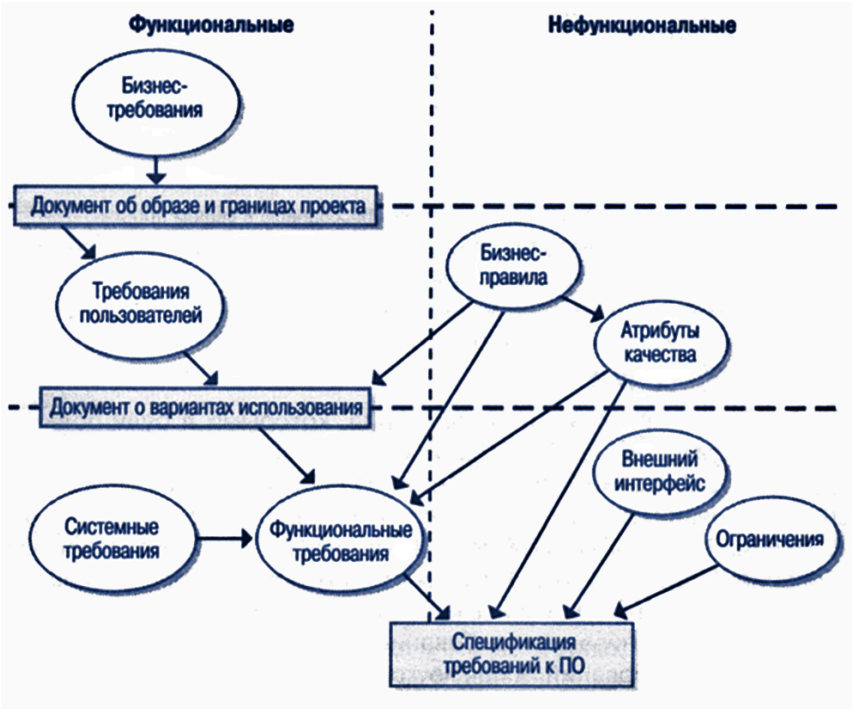

Рис. 15. Модель уровней требований к программному обеспечению

Бизнес-требования (business requirements) содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансирует проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга или «ясновидец» (специалист по системам машинного зрения). В *документе об образе и границах проекта* объясняется, почему организации нужна такая система, то есть описаны бизнес-цели, которые организация намерена достичь с ее помощью. Определение *границ проекта* представляет собой первый этап управления общими проблемами расползания границ.

Требования пользователей (user requirements) описывают цели и задачи, которые пользователям позволит решить система. К отличным способам представления этого вида требований относятся варианты использования, сценарии и таблицы «событие-отклик». Таким образом, в *документе о вариантах использования* указано, что клиенты смогут делать с помощью системы.

Термином *системные требования (system requirements)* обозначают высокоуровневые требования к продукту, которые содержат многие подсистемы. Говоря о системе, мы подразумеваем программное обеспечение или подсистемы ПО и оборудования. Люди – часть системы, поэтому определенные функции системы могут распространяться и на людей.

Бизнес-правила (business rules) включают корпоративные политики, правительственные постановления, промышленные стандарты и вычислительные алгоритмы. Бизнес-правила не являются требованиями к ПО, потому что они находятся снаружи границ любой системы ПО. Однако они часто налагают ограничения, определяя, кто может выполнять конкретные варианты использования или диктовать, какими функциями должна обладать система, подчиняющаяся соответствующим правилам. Иногда бизнес-правила становятся источником атрибутов качества, которые реализуются в функциональности.

Хотя в модели на рисунке показан поток требований в направлении сверху вниз, следует ожидать и циклов, и итераций между бизнес-требованиями, требованиями пользователей и функциональными требованиями.

Нефункциональные требования можно классифицировать следующим образом:

1) Требования производительности:

- скорость;
- пропускная способность (трафик);
- использование памяти (оперативная память, жесткий диск).

2) Требования надежности и доступности. Они предполагают вероятность неидеальной работы программы и ограничивают область ее несовершенства. С точки зрения доступности оценивается степень, в

которой программное приложение должно быть доступно пользователям.

3) Обработка ошибок. Требования такого вида описывают реакцию разработанной программы на возникающие ошибки.

4) Интерфейсные требования. Каким образом программа взаимодействует с пользователем и с другими программами.

5) Ограничения:

- точность;
- ограничения на инструменты и язык;
- ограничения проектирования;
- стандарты, которые должны быть использованы;
- платформы, которые должны быть использованы.

6) Обратные требования. Требования, определяющие, чего программа не будет делать (имеется ввиду тот функционал, которым возможно пренебречь).

7.2. Трассировка требований

Практически в любом проекте по разработке ПО независимо от уровня первоначальной проработки требований к нему не стоит рассчитывать на то, что требования всегда будут оставаться неизменными. В любой момент могут появиться новые требования, одни старые требования изменятся, а другие могут отпасть. Но основная сложность управления процессом изменения требований заключается в том, что изменения одних требований влияют на другие, и такие влияния нужно отслеживать.

Вид требования отражает различия анализа нового требования в контексте существующих соглашений. Целью такого анализа является поддержка целостности системы требований: нахождение противоречий между требованиями, разрешение противоречий, а при невозможности этого – достижение приемлемых компромиссов. В работах с меняющимися требованиями значительное место занимает отслеживание связей проекта, благодаря которому определяется деятельность, необходимая как для реализации требований, так и для распространения изменений, связанных с требованиями к проекту.

Требования должны быть заданы в виде, допускающем однозначное представление в моделях уровня анализа и конструирования, и способ такого представления должен быть унифицирован для всего проекта.

В проекте должны инструментально поддерживаться связи между требованиями и другими компонентами рабочих продуктов, и эта поддержка должна быть обеспечена.

Прохождение исходного требования через последовательность трансформаций от одного представления к другому, сопровождающееся соответствующим анализом, называется **трассировкой требования.**

Трассировка – основа анализа, проводимого в рамках управления изменениями требований.

В результате трансформаций строятся представления требований, вид которых приспособлен для выяснения целесообразности реализации требований. Если на некотором уровне трансформаций установлено, что данное требование отвергается, то его дальнейшие преобразования не производятся. На рис. 16 обозначены различные представления требований.

Исходное представление – текстовое описание пожеланий к системе, заданное в свободной форме.

Унифицированные представления – исходное представление требования разбивается на элементарные составляющие, которые описываются в базовом виде, приспособленном для дальнейшего использования на всех проектных уровнях.

Типизированное представление – каждое из элементарных составляющих требования приписывается к некоторому типу. В результате формируется набор атрибутов элементарных требований и их значений.

Модельные представления уровня анализа – образы элементарных требований как элементы аналитических моделей системы.

Модельные представления уровня конструирования – образы элементарных требований в диаграммах классов, состояний и других компонентах архитектуры системы.

Программные представления – программные рабочие продукты и их фрагменты, которые рассматриваются в качестве образов требований, представленных очередной версией системы.

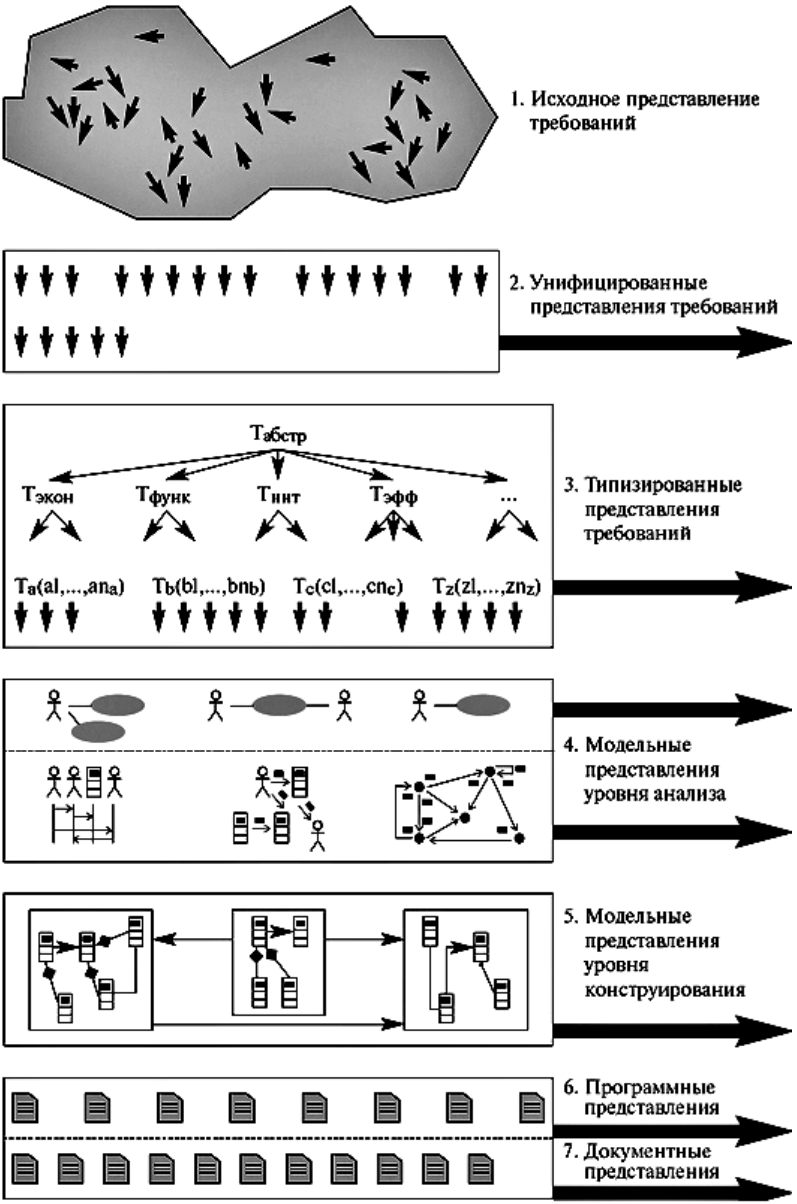

Рис. 16. Виды представлений требований

Документные представления – фрагменты документов, сопровождающих программный код и предназначенных для поддержки деятельности пользователей.

Приведенная схема (рис. 16) демонстрирует, что в той или иной форме вынуждены делать разработчики для преодоления трудностей управления требованиями. Она может рассматриваться в качестве проекции жизненного цикла на задачи анализа требований.

8. Планирование, управление проектом, его завершение

Направляющей силой, ведущей проект к достижению цели, является общий план проекта. Для большинства проектов план исходит из того, что каждый из этапов жизненного цикла программного обеспечения выполняется, предоставляя результаты для следующего этапа. Поэтому задачи, которые требуется решать на каждом этапе, диктуются сразу всеми задачами проекта.

8.1. Исследования, моделирование, оценка технологий и прототипы

Непосредственно перед началом работы над проектом необходимо оценить производительность, масштабируемость, подобрать среду разработки и инструменты и т.д. Значительно лучше, когда это всё будет сделано обоснованно и подкреплено определенными исследованиями.

Исследования, оценка и использование прототипов позволят ещё до начала работы над проектом понять все возможности и ограничения технологий, которые планируется применить.

Исследования бывают фундаментальные и прикладные.

Первые – это процесс открытий и изобретений в надежде создать что-то полезное. Однако у результата такого исследования может и не быть коммерческого применения. С другой стороны, прикладное исследование на основе логических построений и анализа ситуации в некоторой отрасли ведёт поиск потенциально выгодных решений и пытается превратить гипотезы в конкретные идеи, которые помогут создать некоторый продукт.

Прикладные исследования – наиболее важная форма исследований в контексте разработки ПО. Они могут обеспечить критически важное преимущество в конкурентной борьбе, особенно когда потребности пользователей, ПО аппаратные платформы и технологии претерпевают стремительные изменения. Они также дают невероятные возможности группам, способным предвидеть потребности потребителей и задействовать новые технологии для их удовлетворения. Если, занимаясь созданием новой технологии, хочется «остаться на плаву» вопреки всем неожиданностям, то параллельно циклу разработки необходимо вести непрерывную исследовательскую работу.

Каждые 2-3 года на рынке появляются новые, более совершенные технологии. Независимо от того, связаны ли новшества с графическим интерфейсом пользователя, клиент-серверными продуктами, моделями компонентных объектов или Интернетом, всегда следует идти в ногу с фундаментальными нововведениями и стараться, чтобы большие перемены не оставили вас позади. Нужно направлять исследования на поиск, анализ и мониторинг серьёзных изменений на рынке.

Одна из самых важных областей исследования – анализ инноваций и направлений работы конкурентов. Чтобы успешно состязаться с ними, надо понимать их технологии, знать их сильные и слабые стороны.

По завершении исследовательского проекта или при смене приоритетов исследования бывает важно задокументировать результаты и сделать их доступными коллективу. Необходимо довести до сведения группы все без исключения результаты исследований, как положительные, так и отрицательные. Если исследовательский проект действительно обладает большим потенциалом, было бы разумно создать прототип и продемонстрировать группе его возможности, пояснив принцип работы.

Прежде чем приступать к проекту, обязательно нужно разобраться в технологии, намеченной для использования в нём. Это особенно важно для проектов, в реализации которых будут задействованы новые инструменты, компоненты, платформы или решения.

Сегодня практически ни одна программа не создаётся на основе одной технологии. Следует ещё до начала работы над проектом выяснить,

соответствуют ли возможности каждой намеченной для применения новой технологии нуждам проекта. Оценка технологий позволяет решать поставленные вопросы путём тестирования и совместного обсуждения, что помогает обнаружить новые проблемы, о существовании которых никто даже не подозревал до начала использования этой технологии.

Оценку технологий можно проводить по следующим критериям:

- ***качество*** – приемлем ли уровень качества технологии;

- ***совершенство*** **–** обеспечивает ли технология должную производительность, масштабируемость и устойчивость;

- ***поддержка*** – обеспечена ли новая технология адекватной поддержкой;

- ***простота использования*** **–** не слишком ли сложна новая технология в использовании и при отладке;

- ***профессионализм команды*** **–** хватит ли у команды мастерства для применения этой технологии.

Почти всегда возникает ряд важных вопросов, связанных с реализацией той или иной технологии. Моделирование – важная методика, которая поможет получить необходимые ответы.

Создание прототипа – важный этап, который любая группа разработчиков может осуществить ещё до начала работы над проектом. Работа с прототипом поможет понять, как эффективно воплотить ключевые функции программы, оценить сложность реализации ключевых технологий и необходимое для этого время, а также свести к минимуму общий риск ошибок и срыва планов.

8.2. Задачи и оценка времени для их выполнения

Задачи – это основные строительные блоки плана, они являются представлением конкретной работы, которую нужно сделать. Кратко и точно сформулированные задачи позволяют быстро обнаружить отставание от плана. Если задачу нельзя завершить за 1-2 недели, её следует разбить на две или большее меньших задач. Исполнение плана, составленного из долгосрочных задач, труднее контролировать.

При составлении списка задач обязательно нужно учитывать их взаимосвязи в рамках проекта. Например, зная, как одни задачи зависят от других, можно расположить их в нужной последовательности, причём ключевые задачи всегда должны завершаться в первую очередь. Нужно выяснить, сколько времени займёт каждая задача. Это можно сделать, оценив время, необходимое для выполнения некоторой задачи (т.е. выдвинув обоснованное предположение о сроках).

В плане необходимо наиболее полно отразить все этапы выполнения проекта и их сроки. Нужно также обозначить контрольные точки выполнения проекта. По наступлению данных контрольных точек необходимо проводить анализ состояния проекта.

К аспектам, влияющим на удачное выполнение проект,а можно отнести:

- распараллеливание процессов работы (к примеру, разработку необходимо вести параллельно с тестированием);
- частые сборки и тесты сборок;
- постоянный обмен информацией по проекту между участниками команды (проведение собраний, обсуждение прогресса, успехов или неудач, поощрение за успехи).

В плане необходимо также учитывать и такие этапы, как бета-тестирование и подготовка кандидата на выпуск.

Необходимо также помнить и о возможном итеративном зацикливании, которое может возникнуть при неудовлетворительном выполнении очередного этапа.

8.3. Цикл управления проектом

Помимо планирования, команде разработчиков необходимо вести управление проектом. К инструментам управления можно отнести:

- наблюдение,
- контроль,
- оценивание.

Если управление рассматривать в качестве способа организации работы исполнителей над проектом, то менеджерскую активность, свя-

занную с ним, можно представить в виде цикла управления, включающего в себя следующие процессы:

- ***постановка задач*** – определение заданий для разработчиков, проверка понимания заданий и другие действия, необходимые для решения задачи текущего периода;

- ***выделение ресурсов и указание контрольных сроков*** – обеспечение осуществимости решения задачи периода;

- ***текущий мониторинг активизированных процессов решения задачи*** – набор действий, которые позволяют менеджеру быть в курсе того, как исполнители справляются со своими сферами ответственности, определенными для них постановкой задачи периода;

- **мероприятия в контрольных точках** – контрольные и оценочные действия, позволяющие сформулировать итоги решения поставленной задачи с точки зрения запланированных результатов и с позиций качества процесса выполнения работ;

- **оптимизация процесса разработки** – коррекция расстановки кадров, решений и планов.

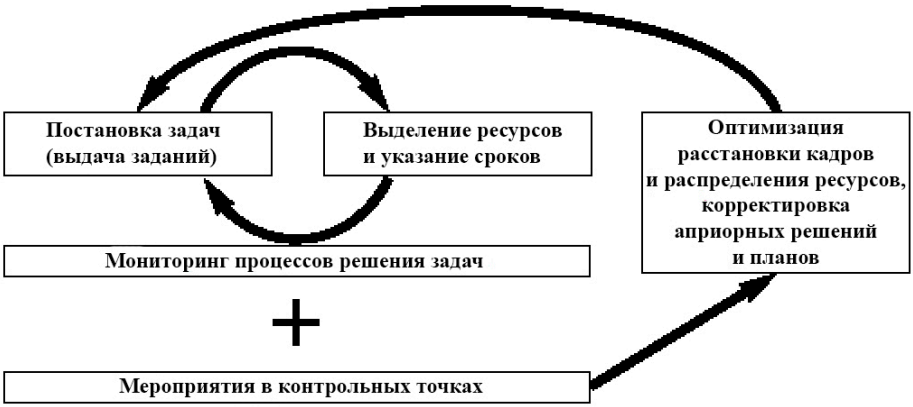

Рис. 17. Цикл управления в проекте

Составляющие цикла управления естественно связывать с контрольными точками линии жизни проекта, используя их как точки отсчета управленческой деятельности. Это именно те моменты, когда выдаются задания разработчикам на очередные работы и подводятся ито-

ги завершаемой ими деятельности, вложенной в деятельность проекта в целом. Схематически этот цикл изображен на рис. 17.

8.4. Завершение проекта

Момент, когда произошло завершение всех этапов, когда разработанное ПО отправлено заказчикам, можно считать завершением проекта. При этом и само *завершение проекта* является важным, требующим внимания этапом.

Закрытие проекта венчает работу команды. Оно также позволяет участникам осознать всю важность проекта, а также почувствовать, что их вклад и старания получили признание и были оценены по достоинству.

Каждый член команды должен быть уверены, что его усилия не пропали зря. Важно, чтобы ни у кого не возникали такие вопросы, как: «Была ли моя работа значима? Стоило ли так напрягаться? Была ли моя работа замечена?» – или, что ещё важнее: «Буду ли я также выкладываться в следующий раз?».

При правильно проведённом закрытии необходимо:

- известить всех об окончании проекта и о передаче программы заказчику;
- выделить индивидуальные достижения, вклад и преданность делу;
- отметить общие усилия и эффективность работы команды в целом;
- помочь участникам команды увидеть их ошибки и извлечь из них урок;
- решить текущие проблемы с кадрами или проектом;
- начать подготовку к следующему проекту.

Одним из хороших приемов управления командой является предоставление повышения квалификации работникам по завершениим проекта. Когда работа над проектом позади, участники команды смогут сосредоточиться на изучении новых технологий или новшеств в уже известных им методиках, появившихся во время работы над последним проектом.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Салливан, Э. Время – деньги. Создание команды разработчиков программного обеспечения / Э. Салливан. – М.: Русская редакция, 2002. – 544 с.

2. Мараско, Дж. IT-проекты: фронтовые очерки / Дж. Мараско. – СПб.: Символ-Плюс, 2007. – 384 с.

3. Макконелл, С. Профессиональная разработка программного обеспечения / С. Макконелл. – СПб.: Символ-Плюс, 2006. – 240 с.

4. Спольски, Д. Руководство командой разработчиков программного обеспечения. Прикладные мысли / Д. Спольски. – М.: ООО «ИД Вильямс», 2008. – 144 с.

5. Microsoft Solutions Framework. Дисциплина управления рисками MSF, вер.1.1 [Электронный ресурс] / Корпорация Майкрософт (Microsoft Corp.), 2002. URL: http://www.microsoft.com/rus/msf.

6. Куприянов, А.В. Документирование функциональной спецификации программного средства: метод. указания к лабораторной работе №3 по курсу «Проектирование программных комплексов» / А.В. Куприянов. – Самара: Самарский государственный аэрокосмический университет, 2011. – 16 с.

7. Фатрелл, Р.Т. Управление программными проектами: достижение оптимального качества при минимуме затрат / Р.Т. Фатрелл, Ф.Ш. Дональд, Л.И. Шафер. – М.: Вильямс, 2004. – 1135 с.

Учебное издание

Благов Александр Владимирович

МЕНЕДЖМЕНТ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Учебное пособие

Редактор Ю.Н. Литвинова
Доверстка: Е.С. Кочеулова

Подписано в печать 26.11.2014. Формат 60×84 1/16.
Бумага офсетная. Печать офсетная. Печ. л. 5,25.
Тираж 100 экз. Заказ . Арт. – Д1(5)/2014.

Самарский государственный
аэрокосмический университет.
443086 Самара, Московское шоссе, 34.

Изд-во Самарского государственного
аэрокосмического университета.
443086 Самара, Московское шоссе, 34.

ДЛЯ ЗАМЕТОК

ДЛЯ ЗАМЕТОК