УДК 378.14

ПОДГОТОВКА КАЧЕСТВЕННЫХ ПРОГРАММИСТОВ: ПРОБЛЕМЫ ОБУЧЕНИЯ

Лукин В.Н.,
доцент, кандидат физико-математических наук, профессор кафедры прикладной информатики и мультимедийных технологий факультета информационных технологий Московского психолого-педагогического университета (МГППУ)

Программное обеспечение, во многом определяющее жизнь современного человека, нередко обладает крайне низким качеством, которое во многом зависит от уровня подготовки специалистов в вузе. В статье рассматриваются варианты решения проблемы обучения программированию с учётом компетенций, профессиональных стандартов, стандартов качества программного обеспечения. Предлагается подход, позволяющий выработать профессиональные навыки.

Ключевые слова: *программная ошибка, качество программ, качество обучения, компетенции, профессиональный стандарт, производственная практика.*

Среда обитания современного человека во многом определяется окружающим его программным обеспечением (ПО), качество которого порой бывает ниже всякой критики. Человек с недоумением смотрит на интерфейс нужной ему программы, не понимая, как с нею управляться, он привыкает выполнять множество ненужных движений для достижения простых результатов. Некоторые программы содержат «дополнительные функции», которые ничего, кроме неприятностей, не доставляют, другие не содержат или глубоко маскируют простые и естественные. Мелкие бытовые неприятности, связанные с некачественным ПО, перемежаются с катастрофическими. Порой возникает ощущение, что нормальных программ почти нет, все с какими-то дефектами. Д. Платт [5] утверждает, что современное программное обеспечение отвратительно, и приличных слов, чтобы выразить этот факт, нет. Но раз отказаться от него мы не можем, давайте что-то делать, чтобы улучшить ситуацию. Приведём три достаточно свежих примера, иллюстрирующих качество ПО.

Сообщение Европейского космического агентства 23 ноября 2016. Сбо*й в работе блока, измеряющего угловые скорости, произошёл при входе «Скиапарелли» в атмосферу Марса. Получив*

неверные данные, система управления неправильно вычислила высоту, которая оказалась отрицательной, то есть ниже уровня поверхности. Это привело к преждевременному раскрытию парашюта и срабатыванию тормозных двигателей, а затем к активации на высоте 3,7 километра системы посадки, которая должна была сработать уже после того, как модуль сел бы на Марс.

Здесь просто букет взаимосвязанных ошибок: система, вычислив высоту и получив явно некорректные данные, не удосужилась перепроверить измерения или принять какие-то экстренные меры, а стала действовать по штатному сценарию. Пусть даже модуль зарылся вглубь Марса, но парашют раскрывается (преждевременно!), включаются тормозные двигатели и модуль, понятно, падает на поверхность.

Сбой в «Мегафоне». 19 мая 2017 *«МегаФон» сообщил о «массовых сложностях со связью». Генеральный директор компании «МегаФон» С. Солдатенков: «Технический сбой на нашем оборудовании, повлиявший на работу ряда услуг, устранён. Минувшей ночью и утром мы провели необходимые работы по восстановлению полноценной работы сети и окончательной ликвидации последствий ошибки программного обеспечения. Тем абонентам, кого затронула авария, мы предоставим компенсацию». Генеральный директор оператора Yota, работающего на сетях «МегаФона», В. Добрынин: «Мы приняли решение выплатить компенсацию до 20 млн руб. клиентам, которые остались в этот день без возможности совершать звонки».*

Этот случай, конечно, не столь катастрофичен, но для обычных людей гораздо более заметен. И опять программное обеспечение! Таких примеров можно привести очень много. Крайне неприятны ошибки, приводящие к появлению уязвимостей в ПО, что способствует его взлому. Это касается, к сожалению, и государственных информационных ресурсов.

Неудачный запуск с космодрома Восточный 28 ноября 2017. *Комиссия Роскосмоса опубликовала официальные результаты расследования аварии, которая привела к потере спутника «Метеор-М». К нештатной ситуации привело поведение разгонного блока «Фрегат» после его отделения от ракеты. «Это выявило скрытую проблему в алгоритме, которая не проявлялась десятилетиями успешных пусков связки «Союз-Фрегат», — указывается в сообщении. После отделения разгонного блока система управления «начала выдавать управляющее воздействие на разворот орбитального блока в требуемое угловое положение». В это время средства телеметрии зафиксировали нештатное угловое положение «Фрегата», который ушёл с расчётной траектории. «Сложилось такое сочетание параметров стартового стола космодрома, азимутов полёта ракеты-носителя и разгон-*

ного блока, которое не встречалось ранее. Соответственно, не было выявлено при проведённой наземной отработке баллистической траектории согласно действующим методикам», — указывается в заявлении комиссии. «Проведя всесторонний анализ, члены комиссии считают, что проявление этой некорректности алгоритма могло и не произойти при запуске с космодрома Восточный этой же полезной нагрузки с этим же разгонным блоком на этой же ракете. Пуск прошёл бы штатно, например, летом, либо в случае, если бы районы падения отделяемых частей РН лежали в стороне от выбранных», — предполагают в Роскосмосе. Риски при запуске были застрахованы на сумму 2,6 млрд. руб. Глава правительства Дмитрий Медведев заявил, что понесённых в результате гибели спутников потерь не покроют «никакие страховые выплаты».

Опять полное собрание удивительных суждений, суть которых одна: никудышнее качество программного обеспечения. Посмотрим: проблема в алгоритме якобы пряталась десятилетиями, но тогда не было подобных пусков и, соответственно, программного обеспечения! Далее, сочетание параметров, которое сложилось, должно обязательно встретиться ранее, чтобы его предусмотреть? Тогда методики контроля вообще никуда не годятся. И ещё лучше: Роскосмос предполагает, что проявление некорректности алгоритма могло и не произойти, если бы пуск прошёл, например, летом, либо в случае, если бы районы падения отделяемых частей лежали в стороне от выбранных. Добавим: если бы звёзды сложились иначе. Всё!

Конечно, если бы проблема качества была лишь в «эффективных менеджерах», не умеющих планировать разработку ПО, было бы не так страшно. Но беда в том, что юные авторы программ, которые идут из вузов, реально не умеют производить качественный программный продукт.

Будем считать, что качественный продукт производит качественно обученный выпускник, а качество обученности определяется степенью соответствия возможностей специалиста требованиям профильного производства, в нашем случае — предприятия, специализирующегося на информационных разработках. Тогдацель обучения программированию в вузе — умение создавать качественное программное обеспечение. Разумеется, качество процесса обучения не в полной мере гарантирует качество специалиста: помимо технологии обучения важно и то, чему обучают, и то, кого обучают, и то, кто обучает.

Приведём некоторые высказывания Роберта Гласса [1], касающиеся качества и особенностей разработки ПО.

• Улучшение качества разработки ПО увеличивает количество

пользователей, срок использования и потенциальные возможности по доработке.

- Наиболее важный фактор в разработке ПО — это сами программисты.
- Программисты предрасположены к определённым ошибкам из-за особенностей мышления.
- Фаза устранения ошибок — самая трудоёмкая.
- 80% работы по созданию ПО приходится на интеллектуальную деятельность, остальное — это рутина, и только она поддаётся автоматизации.
- Результаты методик и инструментов гораздо скромнее обещанного и дают улучшение качества и производительности на 5–35%.
- Большинство учёных в области ПО больше защищают свои теории, нежели занимаются исследованием. В результате у нас много разрекламированных, но неэффективных методик инструментов.

Обратимся к профессиональным стандартам [8], которые регламентируют деятельность по созданию программного обеспечения, и рассмотрим некоторые его положения, относящиеся именно к программисту, чтобы понимать, чему учить.

Программист (профессиональный стандарт)

Основная цель вида профессиональной деятельности: разработка, отладка, проверка работоспособности, модификация программного обеспечения.

Обобщённые трудовые функции:

- разработка и отладка программного кода;
- проверка работоспособности и рефакторинг кода программного обеспечения;
- интеграция программных модулей и компонентов и верификация выпусков программного продукта;
- разработка требований и проектирование программного обеспечения. Если говорить о программировании в узком смысле, то из обобщённых трудовых функций мы выбираем только разработку и отладку программного кода.

Разработка и отладка программного кода:

- формализация и алгоритмизация поставленных задач;
- написание программного кода с использованием языков программирования, определения и манипулирования данными;
- оформление программного кода в соответствии с установленными требованиями
- работа с системой контроля версий;
- проверка и отладка программного кода.

Рассмотрим некоторые существующие подходы к подготовке качественных программистов. Это «компетентностный» подход, подход, основанный на стандартах качества программного обеспечения и подход,

ориентирующийся на производственную практику.

«Компетентностный» подход

Критерий качества, ориентированный на внешний мир, попытались применить к студентам и бюрократы от образования, взяв, как всегда, иностранный опыт и, как всегда, применив его максимально криво. Было придумано новое значение слова «компетенция», которое в русском языке означает либо круг полномочий или прав (нам это не интересно), либо знание и опыт человека в какой-то области. Во множественном числе обычно не употребляется, говорят, например, «он компетентен в вопросах программирования».

Теперь возьмём некоторые учебные дисциплины из области программирования и посмотрим, какие «компетенции» следует иметь прослушавшим соответствующие курсы (стиль оригинала сохранён, но грамматические ошибки исправлены).

Итак, существующие «компетентностные» критерии качества выпускника не удовлетворяют профессиональному стандарту программиста. Не удивительно, что на третьем курсе многие студенты не умеют ни программировать, ни грамотно составить алгоритм достаточно простой задачи, скажем, перемножения матриц.

Но это ещё не всё. В стандарте образования мы не увидим разделов, посвящённых такому принципиально важному вопросу, как обучение разработке *качественных* программ. Получается, что самое главное в подготовке студента ложится, в конце концов, на преподавателя, на его желание и умение подать материал на должном уровне.

Разумеется, сказанное не отрицает подход, основанный на знаниях и умениях студента, которые объединены термином «компетенции». Просто при формальных оценках надо особое внимание уделять именно их содержанию. В нынешнем виде их нельзя качественно проконтролировать, и преподаватели, насколько я знаю, просто их игнорируют.

Что можно предложить? Ну хотя бы самые простейшие варианты, которых должно быть не слишком много, но которые должны быть понятными всем участникам процесса и, конечно, проверяемыми. Например, следующие. *Информатика и программирование:*

- знание основ структурного и объектно-ориентированного программирования;
- умение разработать и реализовать основные алгоритмы;
- владение не менее чем двумя алгоритмическими языками;
- представление о разработке программ в сетевой среде.

Практикум по программированию:

- знание основ тестирования ПО;
- умение отлаживать ПО;

• владение инструментальными средствами разработки ПО.

Структуры и алгоритмы компьютерной обработки данных:

• знание теории баз данных (БД);

• владение методами проектирования баз данных;

• умение работать в среде хотя бы одной системы баз данных;

• умение формировать запросы к БД хотя бы на одном языке запросов.

Стандарты качества программ

Разберёмся с качеством программы как с предметом обучения. Итак, какую программу считать качественной?

Согласно ГОСТ РВ 51987–2002, под качеством функционирования информационных систем понимается совокупность свойств, обусловливающих их пригодность в соответствии с целевым назначением.

▼ Таблица 1

Код	Содержание	Примечания
	Информатика и программирование	
ОПК-3	Способность использовать основные законы естественнонаучных дисциплин и современные информационно-коммуникационные технологии в профессиональной деятельности	К программированию не имеет отношения
	Практикум по программированию	
ПК-1	Способность проводить обследование организаций, выявлять информационные потребности пользователей, формировать требования к информационной системе	Не имеет отношения к дисциплине
ПК-2	Способность разрабатывать, внедрять и адаптировать прикладное программное обеспечение	Подходит, но непонятно, как обучить внедрению
ПК-3	Способность проектировать ИС в соответствии с профилем подготовки по видам обеспечения	Не по теме дисциплины
ПК-7	Владеть знаниями о содержании основных этапов и тенденций развития программирования, математического обеспечения и информационных технологий	Подходит, но лишь с исторической точки зрения
ПК-8	Способность программировать приложения и создавать программные прототипы решения прикладных задач	Формулировка некорректна: создание прототипа предшествует программированию приложения
ПК-9	Способность составлять техническую документацию проектов автоматизации и информатизации прикладных процессов	Не по теме дисциплины
ПК-22	Способность анализировать рынок программно-технических средств, информационных продуктов и услуг для создания и модификации информационных систем	Не по теме дисциплины

Код	Содержание	Примечания
	Структуры и алгоритмы компьютерной обработки данных	
ОПК-5	Владеть информацией о направлениях развития компьютеров с традиционной (нетрадиционной) архитектурой; о тенденциях развития функций и архитектур проблемно-ориентированных программных систем и комплексов	Не имеет отношения к дисциплине
ОПК-7	Способность использовать знания основных концептуальных положений функционального, логического, объектно-ориентированного и визуального направлений программирования, методов, способов и средств разработки программ в рамках этих направлений	Не по теме дисциплины
ОПК-8	Способность использовать знания методов проектирования и производства программного продукта, принципов построения, структуры и приёмов работы с инструментальными средствами, поддерживающими создание программного обеспечения	Не имеет отношения к дисциплине
ОПК-10	Способность использовать знания методов, архитектуры, алгоритмов функционирования систем реального времени	Не по теме дисциплины
	Рекурсивно-логическое программирование	
ОПК-7	см. выше	Частично подходит
	Разработка и стандартизация программного обеспечения	
ОПК-7	см. выше	Частично подходит
ОПК-9	Способность использования знания методов организации работы в коллективах разработчиков ПО, направления развития методов и программных средств коллективной разработки ПО	К разработке — косвенное отношение, к стандартизации — никакого

В стандарте ISO 9126 качество определяется перечислением свойств, которые представлены шестью группами: функциональная пригодность, надёжность, применимость, эффективность, сопровождаемость, переносимость. Конечно, оценивать качество студенческих работ с позиции стандарта вряд ли разумно, но мы должны, по крайней мере, видеть цель.

В производственных условиях используются различные, нередко достаточно сложные, методы контроля качества. К сожалению, в процессе обучения студентов в вузе они обычно не рассматриваются. Ситуацию можно улучшить, если с первого занятия требовать выполнения хотя бы очевидных критериев качества программ и обучать методам его достижения.

Согласно профессиональному стандарту, базовой учебной дисциплиной будем считать «Программирование». Тогда основная цель обучения — формирование навыка программирования, а основной

вид занятий — лабораторные работы.

Обычный ход лабораторных занятий, при условии, что лекционный материал усвоен и на семинарах задачи разобраны, следующий:

- даётся задание, как правило, очередная задача по теме;
- студент разрабатывает программу и отлаживает её;
- преподаватель проверяет, что получилось, и оценивает.

Могут быть следующие варианты процесса, при которых требуется вмешательство преподавателя:

1. Студент не может написать программу.
2. Программа написана, но не проходит трансляция (есть синтаксические ошибки).
3. Программа запускается, но работает не так, как задумано.
4. Программа отлажена и сдаётся преподавателю.

Первый случай, как ни странно, — довольно распространённое явление. Связан он с тем, что студент либо не готов к занятию, либо не знает, с чего начать, либо не умеет синтезировать решение, хотя всё понимает. Бывают и другие причины. Самое простое и обычное — студент не готов к занятию, гораздо сложнее последнее. Иногда, хотя и редко, так и не удаётся растолковать студенту суть программирования. Но часто помогает детальный разбор задачи и предложение её модифицировать или расширить.

Программа написана, но не проходит трансляция. Обычно студент может разобраться в синтаксических ошибках, но он нередко взывает: «У меня не работает!» Что делать? Указывать на ошибку? Тогда это будет вечно: халява соблазнительна. Сказать, чтобы разбирался сам? Но тогда зачем преподаватель? Лучше посмотреть, указать на диагностическое сообщение и предложить им воспользоваться. Конечно, локализация ошибок не однозначна, но в программах учебного характера обычно всё просто.

Программа запускается, но работает явно не так, как задумано, и это понимает сам студент, но ошибки не видит. Можно сесть рядом со студентом и указать на ошибку. Но здесь, опять же, есть опасность работы вечным отладчиком: в следующий раз он уже даже не станет её искать. Полезно посмотреть, не типичная ли это ошибка. Если да, указать её. А лучше в ходе лабораторных работ заранее привести примеры типичных ошибок. Если ошибка сложная, показать, как её найти и исправить. Именно в этом и заключается отладка. К сожалению, ей в учебном плане практически не уделяется внимания, хотя причина большинства ошибок — небрежно отлаженные программы. Необходимо в течение всего курса обращать внимание на различные методы выявления ошибок, от анализа кода до методов «грубой силы».

Студент должен понимать, что процесс отладки программы, особенно при небрежном тексте, занимает гораздо больше времени, чем её написание.

Программа отлажена и сдаётся преподавателю. В «большом мире» этому процессу соответствует сдача программы заказчику. Цель проверки — не только поставить отметку, даже не только обнаружить ошибки и указать на них, а в большей степени показать студенту, что такое качественная программа и как добиться качества.

Теперь проанализируем применимость положений стандарта качества к программам в учебном процессе.

Функциональная пригодность. Это наиболее очевидная и легко проверяемая характеристика. Думаю, каждый преподаватель требует, чтобы студенческая программа выполняла именно те функции, которые заданы. Если задание включало какие-то условия, необходимо проследить, чтобы все они были выполнены, пусть это и займёт время. Полезно, хотя и не обязательно, подготовить набор функциональных тестов, контролирующих все заданные условия, и продемонстрировать их работу студенту.

Распространённые ошибки. Реализован неполный набор функций. Причины — не хватило времени, не хватило знаний или умения, элементарная лень: надежда, что и так «прокатит».

Функции реализованы неверно или не полностью. Причины — как и в первом случае, кроме того, студент мог не понять или неверно интерпретировать условие.

Неверно обрабатываются исключительные ситуации. Обычно это ошибки ввода или ошибки при работе с файлами. Причина, помимо предыдущих — уверенность, что пользователь будет работать «правильно». Очень коварные ошибки, по типу близкие к ошибкам применимости.

Возможные осложнения. При безнаказанности студент привыкает к небрежному программированию, что крайне негативно сказывается не только на старших курсах, где более сложные задачи требуют более точной работы, но, что хуже, в дальнейшем, если он станет профессиональным программистом. Особое внимание следует уделять обработке исключительных ситуаций. Этот тип ошибок вызывает у студента ощущение, что к нему придираются.

Задача транспонировать матрицу. Программа требует ввода количества строк и столбцов. Ввожу разные числа — ошибка. «Числа должны быть равными!» –«А зачем их два?» — «На всякий случай». Объясняю, что всё лишнее вредно. Ввожу отрицательное число — принимает, но снова ошибка. «Размерность не может быть отрицательной!» Говорю, что с детства знаком с этим фактом, но раз не запрещено — значит, разрешено? Студент

с трудом понимает, что в жизни, если на стенке написать: «Задавать только числа от 1 до 5», кто-нибудь тут же введёт 7. Мне удалось наблюдать, как лаборантка, беседуя с подружкой, присела на клавишу пробела и вводила этот пробел довольно долго, пока буфер не переполнился.

Решение. Студент должен обязательно выполнить все условия и ограничения, приведённые в задаче и следующие из её постановки. Он должен понимать, что снижение оценки или отказ в приёме работы — это следствие неполной или некачественной функциональности. Он должен научиться тестировать программу не для демонстрации её правильности, а для поиска ошибок.

Надёжность. Вообще говоря, надёжность определяется частотой появления ошибок в процессе опытной эксплуатации. В нашем случае этот аспект можно связать с количеством ошибок, определяемых преподавателем в процессе сдачи.

Распространённые ошибки. Студент каждый раз приносит одну и туже работу, в надежде, что преподаватель рано или поздно исправит все ошибки.

Тот же вариант, но обнаруженные преподавателем ошибки исправляет сам студент, после чего уверяет, что все ошибки исправлены, а про другие разговора не было.

Студент ошибки исправляет и делает новые.

Возможные осложнения. В мягком варианте приёма работ, пока не будет достигнуто требуемое качество без учёта количества подходов, студент перестаёт думать об ошибках, предоставляя это преподавателю. В профессиональной жизни такая привычка приводит к отладке своих систем руками пользователя, что маскируется термином «бета-тестирование». Пользователя это раздражает, и нередко неудачная сдача системы заказчику — закономерное следствие.

Решение. Полезно в общую оценку за задачу включить «коэффициент надёжности», зависящий от количества подходов, за которые студент сдаёт работу. Если оценка зависит от этого показателя, студент при итерационном зачёте поостережётся сдавать что попало.

Применимость. Применимость (дружественность, эргономичность) — важная характеристика, порой основная. Она определяет удобство взаимодействия пользователя и программного изделия. К сожалению, в учебных программах внимания этому вопросу почти не уделяется, тогда как специалистов в области построения дружественного интерфейса катастрофически не хватает. Нередко заказчик не хочет платить за продукт, работа с которым доставляет мучения. В вузе в случае отсутствия специального курса задача применимости сводится к построению пользовательского интерфейса в упрощённом

варианте. На качественное обучение нет ни времени, ни возможностей, но студент должен понимать, что такое хорошо, а что такое плохо. Обычно проблема обостряется, когда студент научится писать достаточно сложные программы, не всегда на первом курсе. Ошибки применимости разнообразны, контроль качества достаточно сложен.

Распространённые ошибки.

1. Неопределённость: непонятно, какие задачи решаются, какие действия нужно выполнять. Интерфейс скрывает как цель, так и действия, по которым можно было бы о ней догадаться.

2. Логика действий соответствует, скорее, структуре программы, чем естественной деятельности пользователя.

3. Расположение управляющих и информационных элементов не соответствует естественной навигации (слева направо, сверху вниз).

4. Двусмысленные или непонятные надписи и сообщения.

5. Неудачная компоновка: форма или слишком плотно заполнена, или пуста, или перегружена в одном месте и пуста в другом.

6. Слишком мелкий кегль основного шрифта.

7. Неудачное сочетание атрибутов шрифта (гарнитура, кегль, стиль, цвет).

8. Разновеликие однотипные управляющие элементы.

9. Текст отображается на разных языках одновременно.

10. Пёстрые цвета, плохо сочетаются цвета шрифта и фона.

11. Грамматические ошибки или сленг.

Естественно, здесь отражены лишь некоторые из возможных типов ошибок интерфейса, реально их гораздо больше.

Возможные осложнения. Внешний вид изделия, если он не оговорён заранее, говорит о степени заботы мастера о клиентах, то есть о том, стоит ли ему доверять серьёзную работу. Если не приучать студента к аккуратному оформлению задания, к оценке своей работы с точки зрения возможного пользователя даже в самых простых задачах, из него невозможно получить хорошего прикладного программиста. В дальнейшем он либо сам переучится, либо будет мучить пользователей своими творениями, объясняя, какие они замечательные.

Студент сдаёт работу, из её интерфейса не видно, какую задачу он решает. «Я сделал, что Вы задали» — «Уточните условие». Уточняет. Пытаюсь работать, не получается. «Сначала нужно нажать кнопку <Показать файл>» — «Откуда это известно?» — «Это очевидно». Студент играет роль живой инструкции. Вторая работа. «У Вас шрифт слишком мелкий» — «Если делать крупнее, всё не поместится» — «Не нужно перегружать форму, с нею трудно работать, подумайте о компоновке» — «Так всё на виду». Объясняю, что если автор дома все

вещи равномерно рассыплет по полу, они тоже будут на виду, но работать с ними будет сложно.

Следующий пример взят из реальных работ студентов третьего курса. В таблице приведены основные ошибки интерфейса.

Решение. Необходимо сразу же, в момент постановки задачи объяснить, в какой форме требуется сдавать решение. Ошибки в интерфейсе должны быть столь же весомы, как и другие. Полезно взять на себя роль пользователя и оценивать работу в его терминах. Студент должен почувствовать необходимость правильных интерфейсных решений. Все апелляции к свободе творчества, личным пристрастиям и прочему, если они противоречат базовым принципам построения интерфейсов, следует пресекать: это отговорки, чтобы ничего не переделывать. Нужно стимулировать выработку стиля, в котором студент будет выполнять и другие работы.

Сопровождаемость. В учебных программах она сводится к модифицируемости. Следует обращать внимание студентов, прежде всего, на стиль написания программных текстов. Студент должен знать, что программа пишется однократно, а читается многократно, и её отладка может занимать до 90% времени изготовления программы. Поэтому всякие слова типа «мне и так понятно» не работают. Если молодой специалист приходит наниматься в программисты, опытный руководитель проекта по стилю составит мнение как о его квалификации, так и об умении работать в команде. Необходимо требовать ясную структуру текста, чтобы одним взглядом оценить его сложность. Комментарии обязательны, структурные единицы (процедуры, функции, модули и т.п.) озаглавлены, программы снабжены именем автора, датой изготовления, кратким описанием особенностей. Комментарии поясняют не что делается, а зачем. Имена переменных

Таблица 2

Левая форма	**Правая форма**
Неудачная форма, в оригинале не помещается на экран по высоте	Заглавие формы по умолчанию
Нет заглавия	Неясен сценарий работы
Размеры полей таблицы не соответствуют содержанию	Шрифт различной гарнитуры, различного размера и стиля
Бессмысленный текст «Label5»	Бессмысленный текст «Edit4»
Два языка: «Label5» и остальное	Два языка: «Edit4» и остальное
Грамматические ошибки («колизии»)	
Сленг («ИНФА»)	

должны быть разумными. На протяжении всего текста следует выдерживать одинаковый стиль, программа не должна походить на лоскутное одеяло. Традиционный контроль — структурное тестирование.

Распространённые ошибки. Программа не идентифицирована: нет названия, автора, номера версии, дат последнего изменения.

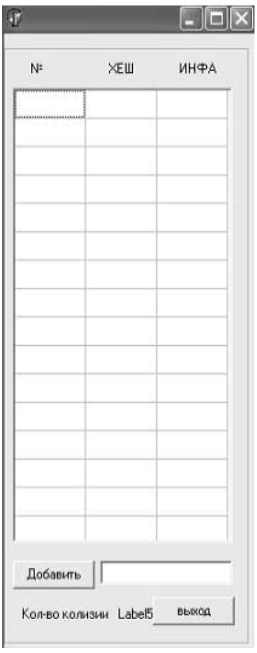
Структурные единицы (процедуры) непоименованы, нет описания их действий.

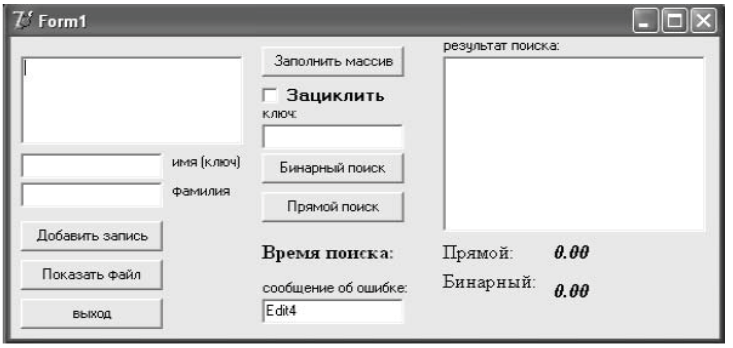
Комментариев нет, их недостаточно или они некорректны, например, содержат текст: «Здесь складываются два числа» или «С этого места программа может глючить, я не проверял».

Текст программы плохо структурирован, что приводит к потере времени при поиске ошибок или при модификации программы.

Имена переменных лишены смысла или противоречат их использованию.

Возможные осложнения. Сопровождение программы — самый длительный период её жизненного цикла, на протяжении которого она неоднократно подвергается модификации. Модификация — это жизненный цикл программы в миниатюре, но ещё добавляется поиск места внесения изменения. Плохо написанная программа многократно увеличивает время поиска и уменьшает надёжность изделия. Вот почему фирмы, разрабатываю-





▲ *Рис. 1.*

щие программные продукты, требуют от сотрудников соблюдения «корпоративного стиля». Студент должен быть уверен: если он хочет стать профессионалом, он должен уметь придерживаться принятого (возможно, и не им) стиля.

Решение. В силу исключительной важности и высокой стоимости этапа сопровождения, следует обязательно, с самого первого дня, требовать следовать определённому стилю. Это плохо воспринимается студентами, особенно не новичками, которые успели испортить стиль в школе. Но приемлемого качества программирования без этого недостигнуть.

Производственная практика

Посмотрим теперь на то, как реальная производственная деятельность студентов влияет на их умение и желание разрабатывать качественные программы.

В статье «Откуда берутся люди, способные создавать надёжное программное обеспечение» (журнал «Программирование», 1976 г.) академик А.П. Ершов выразил мнение, что студент на старших курсах должен участвовать в реальных проектах. Основная идея — знакомство с потребностями производства в качественном программном обеспечении, что позволит своевременно овладеть соответствующими навыками. Однако, насколько мне известно, широкой поддержки это предложение не получило. Причина, на мой взгляд, заключалась в консервативности учебного процесса, в который не вписывалась практика на стороннем предприятии. Тем не менее, при любой возможности студентов подключали к реальному программированию в рамках хоздоговорных или дипломных работ.

Ещё дальше в этом направлении пошёл проф. А.Н. Терехов [7], который утверждает, что никакие лабораторные занятия не заменят работы в коллективе программистов, поэтому вуз должен либо *иметь своё производство*, либо *заключить договор на разработку* ПО. Исходя из своего опыта, он пришёл к выводу, что это наиболее эффективный способ получить качественных программистов. С подобным тезисом нельзя не согласиться, но, к сожалению, оба варианта не так просто реализовать, для этого нужно, как минимум, большое желание руководства вуза. Если учесть, что программистов выпускают все, кому не лень, не только профильные структуры типа ВМК МГУ, такое желание вряд ли обязательно появится.

Следующий вариант практики — *привлечение студентов в реальные проекты*, разрабатываемые на кафедре. Здесь результат зависит от цели. Если основная цель — подготовить качественного программиста, результат, при определённых условиях, получится. Если же студент

используется просто как бесплатная рабочая сила, то вряд ли.

Рассмотрим необходимые условия успеха в первом случае. Во-первых, студент должен знать конкретного заказчика, видеть, для кого он старается. Во-вторых, он должен знать критерии качества, которые необходимо обеспечить. В-третьих, он должен дорожить возможностью работы в этом проекте (проект имеет ценность в глазах студента, попасть в него не просто, работа в нём престижна). Большое значение имеет характер проекта. Он должен быть либо небольшой, на один семестр, либо делиться на автономные модули несколько меньшего, из-за интеграционного тестирования, размера. Причина — необходимость освободить студенту время на сессию. Сложность проекта должна, с одной стороны, соответствовать подготовке студента, с другой — стимулировать получение новых знаний. Следует учитывать, что студенты весьма изобретательны, и это свойство нельзя не поддерживать, пресекая, разумеется, попытки писать витиеватый код, удорожающий отладку и сопровождение. Очень полезно включить в проект научный компонент, дающий возможность студенту подготовить статью или выступить на конференции.

Второй случай, а именно — студент как рабочая сила, обычно не имеет перспектив. Продукт либо не получается, либо имеет столь низкое качество, что впору говорить про его отрицательное воздействие на качество подготовки.

Ещё один вариант практики — *работа студентов в реальных проектах сторонних организаций*, не имеющих прямого отношения к учебному процессу. Тут всё зависит от того, в какой организации работает студент, каков характер проекта и какую роль играет в нём студент. Серьёзная организация с высоким уровнем зрелости — очень хорошая школа, но она нередко предъявляет к студенту требования, которым он не может удовлетворять. Молодая, динамичная организация, изготавливающая интересные и востребованные продукты — случай весьма привлекательный, но увлекающийся студент нередко забрасывает учёбу, что обычно приводит к печальным последствиям. Худший случай — рутинная работа, не только не прибавляющая знаний, но ещё и стимулирующая низкое качество (сойдёт и так, лишь бы побыстрее). Интересно, что в любом случае у студентов порой появляется снобизм, обычно необоснованный, который заметно мешает учёбе.

И, наконец, очень неплохая альтернатива практике: *разрабатывать небольшие, но нужные прикладные программы* в рамках курсовых или дипломных работ. Плюс — хорошее руководство, жёстко ограниченное время и внятные требования к качеству. Этот вариант может успешно совмещаться с другими вариантами практики.

Как влияет практика на качество?

• Вырабатывается стереотип профессиональной деятельности;

• студент понимает, что именно требуется заказчику;

• продукт труда оценивается не только преподавателями, но и коллегами;

• становится понятным, какие именно знания и умения нужное ещё получить.

Почему практика может не дать нужного эффекта?

• Отсутствие интересной практической задачи;

• трудности с руководством: не каждый преподаватель умеет руководить коллективом разработчиков;

• если работа попадает на периоды сессии, приходится жертвовать либо работой в проекте, либо подготовкой к экзаменам;

• неустойчивость коллектива студентов-программистов, что приходит к перераспределению работ и потере времени на формирование взаимоотношений;

• естественный уход старших, опытных участников.

Итог

Если мы хотим подготовить нужных специалистов, следует уделять особое внимание качеству программ, не тратить время на демонстрацию различных программных трюков, характерных для изучаемого алгоритмического языка, не гнаться за количеством кое-как сляпанных программок. Есть и ещё одна опасность. Не секрет, что некоторые студенты, рассчитывая на подушевое финансирование и зная, что их не выгонят, вообще бросают учёбу и пытаются взять преподавателя измором. В таких условиях, при давлении сверху и снизу, преподавателю очень трудно удержаться в рамках качественного контроля, он «снижает требования», по сути, идя на подлог. Опыт показывает, что в основном студенты разумно принимают требования к качеству, если они видят их справедливость и не считают их придирками. Итак,

• главная компетенция выпускника — умение создавать качественное программное обеспечение;

• существующий набор требуемых компетенций не способствует подготовке профессиональных программистов;

• подготовка студента должна явно связываться с качеством продукта;

• любым способом следует обеспечить реальную практическую работу.

Литература

1. *Гласс Р.* Факты и заблуждения профессионального программирования. СПб: Символ-Плюс, 2007.
2. *Купер А.* Психбольница в руках пациентов. СПб: Символ-Плюс, 2004.

3. *Лукин В.Н.* Сопровождение систем и стиль программирования. В сб: Материалы XIX Международной конф. по вычисл. механике и совр. прикладным программным системам, Алушта. М.: Изд-во МАИ, 2015. С. 723–725.
4. *Лукин В.Н., Чернышов Л.Н.* О подготовке специалистов в области ПО. Восьмая конференция «Свободное программное обеспечение в высшей школе»: Тезисы докладов / Переславль. М.: Альт Линукс, 2013.
5. *Платт Д.* Софт — отстой и что с этим делать. Симбо, СПб.-М., 2008
6. *Тарасов С.* Дефрагментация мозга. Софтостроение изнутри. СПб.: Питер, 2013.
7. *Терехов А.Н.* Технология программирования: Учебное пособие. М.: Интернет Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2006. Стандарты: http://www.gost.ru/, обновления и новые стандарты: http://protect.gost.ru/.