Моделирование и анализ данных
2020. Том 10. № 1. С. 140–156
DOI: 10.17759/mda.2020100109
ISSN: 2219-3758 (печатный)
ISSN: 2311-9454 (online)

Modelling and Data Analysis
2020. Vol 10, no. 1, pp. 140–156
DOI: 10.17759/mda.2020100109
ISSN: 2219-3758 (print)
ISSN: 2311-9454 (online)

◇◇◇◇◇◇◇КОМПЛЕКСЫ ПРОГРАММ◇◇◇◇◇◇◇

УДК 004.4

Проблемы качества программного обеспечения

Лукин В.Н.*
МГППУ, Москва, Россия,
e-mail: lukinvn@list.ru

Рассматривается «вечно живая» проблема качества программного обеспечения, которая стоит тем более остро, что программные изделия встроены во всё что можно и нельзя. Что понимается под качеством? Каковы его критерии? Можно ли его измерить? Что мешает качеству и как его улучшить?

Ключевые слова: Качество программного обеспечения, стандарты качества, надёжность, удобство использования, защищённость, оценка качества, управление качеством, обучение качественной разработке.

1. ВВЕДЕНИЕ

Если даже каждый будет делать все,
что в его силах, это все же не будет ответом,
адекватным сложности проблемы качества.
Э.Деминг

Проблема качества программного обеспечения (ПО) обсуждается давно, и всё время остаётся актуальной. Действительно, на каждом шагу приходится сталкиваться с ПО, которое мы считаем некачественным. Но это наша точка зрения, разработчик же может иметь свой взгляд, согласно которому всё хорошо, а если с его продукцией трудно работать – учиться надо. Автору ПО, особенно начинающему, почему-то трудно понять, что программные продукты адресуются другим людям, которым нужно делать своё дело, а не разбираться в его креативных идеях. Вопросам качества посвящено много работ (например, [1, 2, 3, 4]), но проблема оказалась удивительно живучей: все понимают, что так жить нельзя, но живут.

Для цитаты:
Лукин В.Н. Проблемы качества программного обеспечения // Моделирование и анализ данных. 2020. Том 10. № 1. С. 140–156. DOI: 10.17759/mda.2020100109

***Лукин Владимир Николаевич,** кандидат физико-математических наук, доцент, Московский государственный психолого-педагогический университет, Москва, Россия, e-mail: lukinvn@list.ru

2. КАЧЕСТВО

Чтобы избавиться от некачественного ПО, надо, по крайней мере, идентифицировать «врага» и определить характерные черты «друга». Разумеется, это было сделано, в том числе, на уровне стандартов [5, 6, 7, 8]. В стандартах качество определяется по-разному, но одинаково неудовлетворительно. В стандарте ГОСТ Р ИСО/МЭК 9126–93 (ISO 8402:94) [8] качеством считается объём признаков и характеристик программ, который относится к их способности удовлетворять установленным или предполагаемым потребностям. ГОСТ РВ 51987–2002 определяет качество функционирования информационных систем как совокупность свойств, обусловливающих их пригодность в соответствии с целевым назначением. В ISO/IEC 25000:2014 [6] качество определяется как способность ПО при заданных условиях удовлетворять установленным или предполагаемым потребностям. Но если целевое назначение или установленные потребности ещё можно зафиксировать документально, скажем, в спецификации к продукту, то предполагаемые потребности абсолютно неизвестны. Даже неизвестно, чьи они: потребителя или производителя. И если один скажет, что ПО качественное, согласится ли с ним другой? Итак, стандарты не дают возможности однозначно определить, качественный ли продукт.

Хорошо, если не получается определить качество формально, попробуем обратиться к совокупности свойств, которые за него отвечают.

3. ХАРАКТЕРИСТИКИ КАЧЕСТВА

Стандарт ISO 8402:94 [8] определяет шесть групп качества: функциональная пригодность, надёжность, применимость, эффективность, сопровождаемость, переносимость. Каждой из них соответствует набор характеристик (всего их 21). В модели качества продукта, предложенной в стандарте ISO/IEC 25010:2011 (ГОСТ Р ИСО/МЭК 25010–2015) [7], группы качества называются характеристиками, а характеристики предыдущего стандарта – подхарактеристиками. Всего этот стандарт включает восемь характеристик, в которых 30 подхарактеристик: функциональная пригодность, надёжность, удобство использования, производительность, сопровождаемость, переносимость, совместимость, защищённость.

Мы видим, что в последнем стандарте, по сравнению с предыдущим, добавились две характеристики: совместимость и защищённость. Совместимости в [8] не было, а защищённость входила характеристикой в функциональную пригодность. Несомненно, выделение её в отдельную характеристику полностью оправдано. Во-первых, это, всё-таки, не функциональная характеристика, а во-вторых, в условиях значительно участившихся попыток взлома программного обеспечения, защищённости как элементу качества необходимо оказывать особое внимание. Кроме того, термин «применимость» заменён гораздо более подходящим «удобство использования».

Тема качества обсуждается в среде программистов-профессионалов много лет. Первым случаем, после которого качество ПО из разряда курьёзов (типа, неоплаченный счёт на 0,0 долларов привёл к штрафным санкциям неплательщику) перешло в проблему,

был, как известно, неудачный запуск американского космического корабля на Венеру. Стало понятно, что простые опечатки в программе (точка вместо запятой) могут стоить очень дорого. Тот случай породил теоретические исследования в области доказательного программирования, а технологии «подарил» структурное программирование.

Однако окончательно решить проблему качества не удалось. Более того, появлялись всё новые и новые признаки некачественного продукта, которые невозможно было игнорировать хотя бы потому, что от них зависело принятие или непринятие продукта заказчиком. Поэтому в программистском сообществе, наряду с достаточно медленно формирующимися стандартами, появлялись и другие подходы к набору характеристик.

Так, Р. Гласс [3] приводит и обосновывает признаки качества, несколько отличные от стандартов: переносимость, надёжность, эффективность, удобство работы, тестируемость, понятность, модифицируемость. Заметим, что здесь нет функциональной пригодности, которая в стандартах стоит в первой строчке. Р. Гласс принципиально считает, что это не характеристика качества, а естественное свойство любого работающего продукта: если он функционально непригоден, значит, о нём и говорить не стоит.

А. Купер [1] предлагает за основное требование к качеству систем, ориентированных не на специально подготовленного, а на обычного человека, взять простоту и удобство взаимодействия. Это, безусловно, разумно: если использование программного изделия требует специального обучения, а его использование преподносит неожиданные сюрпризы, качественным назвать его нельзя, каким бы эффективным оно не было.

Дж. Рейнвотер [9] считает, что программное изделие качественное, если пользователь им удовлетворён. Однако Р. Гласс [3] утверждает, что качество – лишь один из компонентов удовлетворённости, остальные – это выполнение требований заказчика, своевременная поставка, приемлемая цена. Их он в качество не включает.

А. Н. Терехов [10], на основе анализа нормативных документов, предлагает считать программное обеспечение некачественным, если выполняется хотя бы одно из негативных условий, отвечающих наиболее значимым характеристикам качества: неадекватность функционирования, недостаточное взаимодействие с другими средствами, отказы в процессе применения по назначению, неэффективность по времени отклика, неполнота отражения информации и потеря её актуальности, несоответствие хранимых данных и вводимой информации, нарушение конфиденциальности, некорректность сопровождающей и справочной документации.

Интересно посмотреть на пересечение свойств качества, указанных экспертами, и характеристик стандарта [7]. В нашем случае пересечение, с точностью до названия характеристики, следующее: надёжность, переносимость, удобство использования, защищённость, производительность, сопровождаемость.

Здесь необходимы некоторые комментарии. Прежде всего, рассмотрим понятие «сопровождаемость», которое включено в пересечение, хотя в явном виде у экспертов его нет. Учитывая, что Р. Гласс много лет занимался сопровождением, в своих публикациях неоднократно обращался к этой проблеме, он, казалось бы, должен его упомянуть. Но он, будучи специалистом, глубоко знающим суть проблемы, говорит об этом косвенно, разбивая сопровождение на три компонента: тестируемость, понятность, модифицируемость.

Заметим, что в пересечении нет такого понятия, как *функциональная пригодность*. Мы знаем отношение к нему Р. Гласса. Но ведь кто-то другой должен про это сказать! И действительно, у А. Н. Терехова есть неадекватность функционирования как отрицательный признак. Но адекватность функционирования и функциональная пригодность – вещи разные. Приложение может функционировать, быть пригодным для некоторого производственного процесса, приносить пользу, но не выполнять или не полностью выполнять какие-то его функции. Например, если вы приходите в отделение Сбербанка, чтобы оплатить коммунальные расходы, взять деньги с карточки и отметить платёж, пришедший на книжку, вам придётся несколько раз вводить пин-код, хотя вы никуда не отлучаетесь, а операционист выполняет, по сути, одну транзакцию. Функциональная пригодность здесь есть, как и неадекватность функционирования, которая снижает качество продукта.

К сожалению, никто, кроме А. Н. Терехова, не упоминает проблему, возникающую из-за некачественной разработки *базы данных*. А на это следовало бы обратить особое внимание: сколько здесь бывает неприятностей! Даже нашумевшая в своё время катастрофа при запуске корабля с космодрома «Восточный» произошла во многом, если не исключительно, по причине неактуальности информации, которая служила для ориентации.

Удобство использования и удобство взаимодействия можно считать синонимами, хотя разница есть: в первом случае пользователь работает с системой как с инструментом, ему нужно, чтобы эта «лопата» была удобной. Во втором предполагается двустороннее взаимодействие: пользователь должен понимать «потребности» другой стороны. Это налагает на разработчика дополнительные требования, скажем, понятность и однозначность диалога, включая графическое оформление и навигацию, исключение сленга, умолчаний, некорректных высказываний и т.п.

Термин «*производительность*», который принят в стандарте, видимо, более подходящий, чем эффективность. Последняя может толковаться излишне расширительно: заказчик, использующий систему, может считать её неэффективной, если она, скажем, не приносит прибыли. Понятно, что к производительности это не имеет прямого отношения. Эффективность по времени отклика (А. Н. Терехов) практически то же самое, но здесь подразумевается условие жёсткого реального времени, тогда как производительность может быть в среднем, что обычно устраивает пользователя.

Защищённость, упоминаемая всеми авторами и выступающая как отдельная характеристика в стандарте [7], в отличие от [8], где она входит в функциональную пригодность, с глобализацией информационных структур, становится всё более важной характеристикой качества. Сейчас даже простенькие студенческие поделки включают элементы защиты, хотя бы парольной. А серьёзную информационную систему, особенно если она имеет доступ к жизненно важной информации, вообще нельзя рекомендовать к использованию, если она не имеет профессиональной защиты.

Переносимость – это характеристика, которая требуется, если программное обеспечение планируется использовать в различных программно-аппаратных средах. Переносимость была особенно важна, когда подобных сред было много даже в рам-

ках одной организации, и везде должны были работать одинаковые программные изделия. Сейчас острота проблемы существенно ниже.

И, наконец, *надёжность*. Это свойство упоминается во всех оценках качества с самых ранних времён. Подразумевает оно предсказуемость и уровень доверия к процессу и результатам работы программного изделия, его отказоустойчивость и восстанавливаемость. В более ранние времена сказали бы – программно-аппаратного комплекса, но в последние годы надёжность аппаратуры многократно выросла, а программного обеспечения, похоже, снизилась, так что про аппаратную составляющую говорить не будем. А. Н. Терехов вместо термина «ненадёжность» использует «отказы в процессе применения по назначению», что имеет свой резон: если какую-то вещь используешь не по назначению, скажем, микроскоп для забивания гвоздей, трудно сетовать на ненадёжность. Правда, программные изделия редко используют не по назначению, разве что ПО для видеоконференций, которое применяют для дистанционного обучения в период эпидемии.

Причина ненадёжности – программные ошибки. Трудно вообще придумать, что, кроме программных ошибок, привело бы к ненадёжности. И ошибки эти чаще всего связаны с обработкой нештатных ситуаций. Приведём только пару примеров, первый из которых связан с крупными материальными потерями, а второй, к сожалению, с человеческими жизнями.

По сообщению Европейского космического агентства 23 ноября 2016 г., произошел сбой 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. Коллекция программных ошибок, которые видны из этого сообщения, просто поражает! Система, получив заведомо некорректные данные, не идентифицирует ситуацию как нештатную, а действует по основному алгоритму: парашют раскрывается, включается система посадки, после чего модуль падает на поверхность планеты.

Второй пример – катастрофа с самолётом Boeing 737 Max авиакомпании Ethiopian Airlines в Эфиопии 10 марта 2019 г., в которой погибло 157 человек. Через несколько минут после взлета пилот сообщил диспетчерам о проблемах в системе управления полетом: датчик угла атаки ошибочно показал, что нос судна опасно задран вверх, в результате автоматическая система предотвращения сваливания самостоятельно направила самолет вниз.

Таким образом, низкая надёжность – это типичная проблема разработки программного обеспечения, причин которой множество, но в первую очередь – безответственное руководство проектом, неверное планирование сроков выполнения, низкий уровень подготовки разработчиков. Последнее – в немалой степени следствие экспериментов с обучением информационным технологиям.

И, наконец, никто из названных специалистов не указывает *совместимость* как существенную характеристику качества. Впрочем, и в предыдущих стандартах её

тоже не было. Конечно, совместимость важна, если предполагается расширяемая модель разработки программного средства, которая предполагает включение внешних систем или же оно включается в другие системы. Кроме того, совместимость предполагает корректную работу при разделении ресурсов (сосуществование) и свободный обмен данными (интероперабельность).

В соответствии со стандартом ГОСТ Р ИСО/МЭК 25010–2015 [7] общее представление о качестве описывается двумя моделями: моделью качества при использовании и моделью качества продукта. Выше рассматривалось именно качество продукта.

Модель качества при использовании определяет пять характеристик и описывает воздействие продукции на заинтересованную сторону. Характеристики такие: результативность, производительность, удовлетворённость, свобода от риска и покрытие контекста (полнота контекста, гибкость). Под удовлетворённостью понимают полноценность, доверие, удовольствие, комфорт. Рассматривается экономический риск, риск для здоровья и безопасности, экологический риск.

Обратим внимание, что с точки зрения пользователя качество одного и того же программного продукта может оцениваться совершенно по-разному. Так, одну программную систему, активно внедряемую в медицинские учреждения, врачи оценивали весьма низко. На вопрос, есть ли у неё хотя бы одно достоинство, ответ был: «Да, у принтера почерк гораздо лучше, чем у врача». Правда, один из врачей отметил ещё одно достоинство: результаты анализов видны сразу. Но он же отметил и крайне неудобное взаимодействие: на одно действие нужно пять кликов (число, понятно, условное). Но руководство, своевременно получающее отчёты, наверное, было довольно. Хочется верить, что авторы подумали о таких характеристиках качества продукта, как надёжность, функциональная пригодность, сопровождаемость (на момент оценки использовалась версия 16.2!), но качество в процессе его использования испортило всю картину.

Анализируя программные системы, которые зарекомендовали себя как качественные, приходим к выводу, что основными требованиями к качеству лучше считать характеристики стандарта ГОСТ Р ИСО/МЭК 25010–2015. Детальное их описание и уточнение с использованием подхарактеристик даёт достаточно адекватное представление о качестве, как оно обычно понимается.

4. ОЦЕНКА ХАРАКТЕРИСТИК

Для оценки и сравнения характеристик качества желательно использовать некоторые измерительные алгоритмы на основе объективных критериев. Стандарты рекомендуют предусмотреть возможность объективного и воспроизводимого измерения каждой из характеристик качества для сопоставления с техническим заданием с учётом нормы допустимых ошибок измерения. В этом случае необходима процедура определения числовых значений параметров проекта. Количественные характеристики определяются непосредственно и могут быть представлены числовыми значениями. Но таких характеристик немного. Это производительность и, с некоторыми оговорками, надёжность. Остальные определяются косвенно. Для это-

го рекомендуется идентифицировать подмножество свойств, которое в совокупности покрывает характеристику, получить показатели качества для каждого свойства и определить значение качества для этой характеристики, используя специальные расчётные алгоритмы (функции измерения).

К количественным свойствам относят, в частности, объём текста программы, его сложность, сложность по управлению. Их вычисление определено различными методиками и может быть выполнено автоматически. Однако они составляют незначительную долю в исследованиях программных систем на качество. И польза от них сомнительная: формально удовлетворительный код может содержать некорректные функции, не соответствовать требованиям, неудовлетворительно обрабатывать исключения. Если код большой, не обязательно его качество высокое: сделать большой текст из маленького и ребёнок сможет. Если код мал, он, возможно, непригоден для сопровождения. Сложность текста программы тоже ни о чём не говорит. Она может быть следствием сложного алгоритма, реализованного очень квалифицированным программистом. Но, возможно, это результат многочисленных правок на этапе программирования или порождение новичка, у которого мысли путаются в голове. Таким образом, ценность подобных характеристик для качества ничтожна.

Надёжность ПО определяется, в основном, безошибочностью: зачастую некорректность – следствие ошибок, не выявленных во время тестирования. Она измеряется степенью покрытия ПО тестами. Мерой может служить относительное количество протестированных функций и маршрутов, но следует учитывать уровень потенциальной опасности отдельных элементов, оценка которого часто сложна и не однозначна.

Гораздо труднее оценить такую характеристику, как удобство взаимодействия. Обычно приводят примеры «хороших» и «плохих» пользовательских интерфейсов, но корректных критериев количественной оценки их качества нет.

Защищённость ПО – тоже трудно формализуемая характеристика. Методы взлома развиваются исключительно быстро, злоумышленник нередко использует такую уязвимость, как программные ошибки [11]. Оценка безопасности определяется экспертным, регистрационным или расчетным путем. В стандартах ГОСТ 28195, ГОСТ 15987 приводится метод оценки таких показателей качества, как конфиденциальность, защищенность от программно-технических воздействий и от несанкционированного доступа. В первом случае оценка – это вероятность сохранения конфиденциальности информации в течение заданного периода, во втором – вероятность отсутствия опасного воздействия в течение заданного периода, в третьем – вероятность сохранения защищенности информационных и программных ресурсов. Комплексная оценка формируются на основе частных показателей. Пример расчёта защищённости с использованием автоматического анализа программного кода приводится в работе [11].

Сопровождаемость и часть других характеристики количественно не измеримы, они оцениваются качественным способом. Примеры качественных шкал для измерения приведены в таблице 1.

Таблица 1

Примеры качественных шкал для измерения сопровождаемости

Характеристика и свойства качества	Мера	Шкала
Изучаемость		
трудоемкость изучения применения;	Чел.–часы	1–100
продолжительность изучения;	Часы	1–1000
объем эксплуатационной документации;	Страницы	10–1000
объем электронных учебников	Кбайты	100–10000
Модифицируемость		
трудоемкость подготовки изменений;	Чел.-часы	1–1000
длительность подготовки изменений	Часы	1–1000
Тестируемость:		
трудоемкость тестирования изменений;	Чел.–часы	1–1000
длительность тестирования изменений	Часы	1–100

Мы видим, что даже если меры выражены числовыми шкалами, их неопределённость не даёт возможности адекватно оценить качество. Возникает вопрос и о корректности этих числовых шкал. Например, если объем эксплуатационной документации 1000 страниц, это лучше, чем 10? Возможно, изделие столь некачественно, что без такого документа не обойтись, а короткая инструкция говорит о простоте использования, то есть о высоком качестве. Но короткая инструкция может быть неполной или выполненной небрежно, а объёмистый документ соответствует сложному и серьёзному продукту.

Несколько слов об измерении производительности. В «древние» времена, когда остро стоял вопрос реализации сложных вычислительных алгоритмов, производительность рассматривалась как одна из важнейших характеристик качества: неэффективная по времени программа из-за слабости вычислительной техники могла считаться сутками. С появлением мощных процессоров и ёмкой памяти острота проблемы исчезла и о ней стали забывать. Но с использованием глобальных сетей и необходимостью обработки больших данных проблема производительности снова выходит на первый план. Конечно, в современных условиях доступны значительные вычислительные мощности, однако их загруженность постоянно растёт. Кроме того, всё более значительную долю программного обеспечения составляют системы реального времени, которые весьма чувствительны к эффективности по времени, и их качество принципиально зависит от этой характеристики. Измерение производительности программной системы, которая работает во взаимодействии с другим программным обеспечением, в нагруженных средах, крайне сложная задача. Как правило, оценочные значения можно получить только в результате квалифицированно выполненного нагрузочного тестирования.

5. ПРЕПЯТСТВИЯ

Что можно считать основным препятствием к достижению качества? Однозначно на этот вопрос ответить невозможно: как различные характеристики в разных случаях могут иметь различный приоритет, так и причины нарушения качества в разных

ситуациях различны. Нельзя, например, не согласиться с Д. Платтом [2], что во многих случаях это результат испуганного согласия людей пользоваться навязываемым ему негодным ПО. А. Купер справедливо считает, что производство качественного ПО не требует намного больше ресурсов, мешают традиции производства. Посмотрим на жизненный цикл производства со стороны риска не достижения требуемого качества.

Прежде всего, это ошибки управления проектом, в частности – неверная оценка времени разработки. Почему-то «менеджеры» уверены в правильности своей оценки, а программистов считают способными вписаться в любые заданные временные рамки. В результате образуется технический долг (попросту недоделки), для ликвидации которого потребуется значительные средства, не говоря уж о том, что ПО всё это время будет источником значительных, порой катастрофических, неприятностей. Достаточно вспомнить «Фобос-грунт», проект запуска космического корабля на спутник Марса Фобос, который с треском провалился и породил большие материальные и репутационные потери для нашей страны. Причиной была неверная оценка времени разработки программного обеспечения, срок сдачи которого по естественным причинам перенести было невозможно, разве что на два года. На эту причину указывал Р. Гласс [12]: «Чаще всего основной причиной неуправляемости проекта является плохая оценка сроков».

Проблема корректной оценки трудоёмкости разработки связана, прежде всего, с природой программного обеспечения. В отличие от материального производства, каждый программный проект уникален, и даже если есть аналоги, особенности нового продукта могут существенно повлиять на оценку. Модели и методы оценки, которые используются в области информационных технологий, различаются, и порой весьма значительно [13]. Все они, конечно, не лишены недостатков: будущее точно предсказать нельзя. Но это не значит, что достаточно указания – и «программисты» всё сделают в срок: «Я же им плачу!». Кроме того, начальные оценки почти никогда не уточняются и не пересматриваются. И тут может быть полезным подход, основанный на динамической оценке [13, 14], хотя для него и требуются дополнительные ресурсы.

Считается, что ошибки обходятся тем дороже, чем раньше они сделаны и позже обнаружены. Тяжёлый случай – некачественно проведённый анализ требований, при котором теряются или неверно интерпретируются требования предметной области. К значительному снижению качества приводит неудачное проектирование данных: база данных – основа информационной системы, и неправильно спроектированная, она вынудит программиста разбираться с аномалиями, лишний раз заботиться о целостности и непротиворечивости данных, решать проблему производительности, которую можно было избежать.

На этапе кодирования самые большие неприятности исходят от слабой квалификации программиста. Недостаток квалификации может быть по разным причинам. Если это новичок, есть шанс, что он обучится. Нормальный программист ранее мог работать с другим классом задач, а с текущим не сталкивался. Хуже, когда в роли программиста откровенный дилетант или тупица, который ничему не может или не хочет обучаться. Существенный риск представляет сложность программы, которая может быть следствием сложности задачи. Даже если программа и не слишком

сложная, она может выполняться в многозадачной среде с взаимодействующими процессами, особенно в системе жёсткого реального времени, что требует особого внимания. Программа может быть сложной и при простой задаче, если её реализация беспорядочная, она не обдумывается, а пишется по факту. Такое чаще случается с начинающими программистами, но и опытный программист может попасть в ловушку кажущейся простоты, если поджимает время. Третья причина – наплевательское отношение к стилю написания программы.

Помимо сложности, есть опасный риск, связанный с некачественной отладкой и недостаточным тестированием программы. Разумеется, опытный тестировщик для достижения требуемого качества постарается обеспечить достаточно полное покрытие тестами. Программист, получив результаты тестирования, исправляет свою программу… Но на это требуется время, которого далеко не всегда хватает в конце проекта, обычно из-за ошибочной оценки сроков выполнения работ. В результате жертвуют полнотой тестирования и, соответственно, качеством. Заметим, что в этом случае хуже всего проверяется корректность обработки нештатных ситуаций, что в конечном итоге оборачивается большими потерями.

Как уже отмечалось, модель качества при использовании [7] описывает воздействие программного продукта на пользователя, который оценивает его через такую характеристику, как удовлетворённость. И если с продуктом неудобно работать, пользователь считает его некачественным. Автор был свидетелем ситуации, когда пользователи в знак протеста против неудобного взаимодействия повернули экраны к стенке и отказались работать с, вообще говоря, полезным продуктом. От него пришлось отказаться. Таким образом, неудовлетворительный интерфейс становится проблемой.

Создание качественного интерфейса – сложное профессиональное занятие, которое далеко не всегда может выполнять программист, разрабатывающий программный код. По мнению А. Купера [1], программисту вообще нельзя доверять эту работу: его точка зрения на программу обычно значительно отличается от точки зрения пользователя, и то, что представляется удобным ему, далеко не всегда удобно для пользователя. Он справедливо замечает, что нельзя считать качественным продуктом тот, где нужно «продираться через 11 экранов и в конечном итоге выяснить, что все равно придется звонить в компанию». Ещё более жёсткий взгляд у Д. Платта [2], который замечает, что мы с отвращением смотрим на интерфейс нужной нам программы, не понимая, как с нею управляться, и привыкаем выполнять множество ненужных движений для достижения простых результатов.

В области построения интерфейса разработаны стандарты, как корпоративные, так и международные. В них рассматривается представление интерфейса, его согласованность, состав компонентов, принципы проектирования. Приводятся рекомендации по цветовым решениям, определяются унифицированные действия диалога. Если все разработчики следуют стандарту, пользователь может при переходе на другую систему быть уверенным в правильности обычных действий.

Однако, даже соблюдая стандарт, легко сделать интерфейс некачественным. В большой степени качество взаимодействия с прикладной системой определяется

его стилем, который непосредственно оценивает пользователь [15]. Если выдерживается согласованный стиль, пользователь быстрее привыкает к системе, что снижает риск ошибок взаимодействия и, соответственно, уровень возможных потерь. Работа выполняется эффективнее за счёт того, что всеми необходимыми в данный момент функциями можно легко воспользоваться. К сожалению, программисты интерфейс чаще выстраивают под себя, а не ориентируются на пользователя. Если к тому же стараются сократить стоимость изделия, его качество становится ужасающим. И особый случай – интерфейс для слабовидящих. Если с программным изделием предполагается работа таких пользователей, необходимо оказывать особое внимание качеству и использовать специальные подходы [16].

Существует ещё одно препятствие, о котором редко упоминают. Это переход в процессе выполнения проекта на новые элементы среды разработки, такие, как язык программирования, СУБД, операционная среда или новая версия инструментов. Такое чаще бывает при большого или сложного программного изделия, где качество критично. Конечно, чаще переход происходит в отчаянной ситуации, и руководство видит единственный путь спасения гибнущего проекта в такой соломинке, как некое новшество. Но не зря говорят, что коней на переправе не меняют: богатая практика подобных действий говорит об их порочности. Помимо того, что при этом заметно снижается производительность разработчиков, качество становится неудовлетворительным хотя бы потому, что у разработчиков нет опыта и недостаточно времени для обучения.

6. УПРАВЛЕНИЕ КАЧЕСТВОМ

Термином «управление качеством» называют действия, направленные на улучшение качества продукции. Стандарт ИСО 9000:2005 определяет его как «часть менеджмента качества, направленная на выполнение требований к качеству». Объект управления качеством – процесс создания продукции, субъект – руководители, которые воздействуют на процесс создания продукции. Вместе они образуют систему управления качеством. Группа оценки качества проверяет его параметры в процессе выполнения проекта и перед выпуском продукта. В зависимости от тяжести дефектов продукт либо возвращается на доработку, либо блокируется ошибочная функция, либо вносятся косметические правки.

Продукт проверяется на соответствие формальным требованиям и содержанию. Но что такое «формальные требования»? Соответствие стандартам? Техническому заданию? И даже если продукт полностью всему этому соответствует, ещё нет гарантии, что он качественный. Тем более, что и внятно сформулировать эти требования крайне сложно, и проверить практически невозможно. Достаточно вспомнить рядовой сценарий из области тестирования ПО, когда тестировщик указывает на ошибку, а программист говорит, что это «не баг, а фича». Содержательная проверка требует ещё большего труда, времени и квалификации контролёров. К сожалению, порой какой-то из этих составляющих не хватает, и вторая часть проводится сокращённо, особенно, если в компании поощряются формально-бюрократические процедуры.

Регулярный контроль выполнения проекта и оценки его качества даёт возможность своевременно принять меры, пока ситуация не перешла в необратимую фазу. Разумная форма контроля – обсуждение и экспертная оценка проекта на любой стадии его готовности. Различают концептуальные, технические и административные обсуждения. С точки зрения управления качеством наиболее продуктивны технические обсуждения (сессии). Если они проводятся на протяжении всего периода разработки, и на них затрагиваются вопросы качества, а не только «прошли ли мы очередную веху», есть шанс получить удачный продукт. Обнаруженные недостатки исправляются, результат докладывается на следующей сессии. Важно, что на технических обсуждениях производится не защита сделанного, а выявление затруднений. Следует учесть, что это рабочие по своей сути встречи, здесь нужна открытая атмосфера, а не отчёт перед начальством.

Ну а самое худшее, что сейчас, к сожалению, становится главным – это стремление «менеджеров» к максимально быстрому выбросу ПО на рынок. Если раньше негодные продукты делали с оглядкой, то сейчас существуют даже методологии типа Scrum, которые рекомендуют выпускать очередную версию как можно быстрее, вне зависимости от степени её готовности. Возник совершенно разрушительный термин «минимально пригодный продукт», которым маскируют откровенный полуфабрикат, да и то заведомо негодного качества. И, несмотря на это, производитель заявляет, что он заботится о пользователе: обновления-то идут! Тенденция весьма тревожная.

7. ОБУЧЕНИЕ КАЧЕСТВУ

Естественно полагать, что качественное программное обеспечение производит качественно обученный специалист, нужный на профильном производстве [17]. Тогда в вузе нужно учить не просто программировать, а создавать качественный продукт. К сожалению, в процессе обучения понятие качества если и даётся, то вскользь, да и то на старших курсах, когда уже поздно что-то менять в подготовке студента как программиста [4].

Конечно, методы контроля качества, которые используются на производстве, в вузе далеко не всегда применимы. Однако если знакомить студентов как можно раньше хотя бы с очевидными требованиями к качеству и добиваться их выполнения, ситуацию можно улучшить. Это касается и написания программы, и конструирования интерфейса, и проектирования базы данных.

Положительно на привычку к качественному программированию влияет участие студентов старших курсов в реальных проектах, пользу которого отмечал ещё академик А. П. Ершов («Откуда берутся люди, способные создавать надежное программное обеспечение», «Программирование», 1976 г.). Правда, широкой поддержки по разным причинам это предложение не получило, хотя студентов к реальному программированию подключали в рамках хоздоговорных или дипломных работ. Профессор А. Н.Терехов [10] также считает, что это наиболее эффективный способ получить качественных программистов.

Следующий вариант – привлечение студентов в реальные проекты, разрабатываемые на кафедре. Будем считать, что основная цель этого – подготовить качествен-

ного программиста. В этом случае студент должен знать потребности конкретного заказчика и критерии качества, которое необходимо обеспечить. Конечно, он должен дорожить возможностью работы в проекте. Чтобы освободить время на сессию, проект должен быть либо небольшой, на один семестр, либо делиться на автономные модули. Сложность проекта должна, с одной стороны, соответствовать подготовке студента, с другой – стимулировать получение новых знаний. Студенты весьма изобретательны, что хорошо, но попытки писать витиеватый код необходимо пресекать.

Ещё один вариант – работа студентов в реальных проектах организаций, не имеющих прямого отношения к учебному процессу. Результат зависит от организации, где работает студент, от характера проекта и роли в нём студента. Организация с высоким уровнем зрелости – хорошая школа, но её требования к студенту бывают завышенными.

Небольшая динамичная организация, которая производит интересные продукты – вариант весьма привлекательный, но студент нередко забрасывает учёбу, что обычно приводит к печальным последствиям. Худший вариант – рутинная работа, не прибавляющая знаний и стимулирующая низкое качество.

Очень неплохая альтернатива практике – разработка небольших, но нужных прикладных программ в рамках курсовых или дипломных работ, о чём, собственно, и упоминал А. П. Ершов. При этом, разумеется, должны явно формулироваться требования к качеству. Этот вариант может успешно совмещаться с другими.

Программирование – технологический этап, которому, в отличие от прочих, можно действительно научить студента [18]. Нужно, чтобы он чувствовал, что такое качественная программа, знал, что настоящий специалист задумывается о долгой жизни своего изделия, которую даёт, в частности, такое свойство, как «прозрачность» кода. Следует отказаться от демонстрации различных программных трюков, характерных для изучаемого алгоритмического языка: их использование в реальных программах характерно, скорее, для амбициозного новичка, чем для профессионала. Следует обращать внимание на стиль написания программ. Он, конечно, не синоним качества, но вероятность лучшего качества при хорошем стиле выше.

Есть одно интересное предложение, о котором упоминает Р. Гласс [3] и которое достаточно детально рассматривает В. Ш. Кауфман [19]. Это использование опыта классных программистов, которые по разным причинам ушли из активного программирования. Работа в области информационных технологий – удел молодых, а тот, кто получил хороший опыт и приобрёл мастерство, неожиданно оказывается не у дел. Молодёжь проходит тем же путём, делает те же ошибки, и отрасль практически не развивается качественно. Роль «стариков» при этом – делиться опытом, который позволит улучшить картину.

8. ЭПИЛОГ

Мы все обречены жить в «цифровом» мире, пользоваться неосязаемыми предметами, которые, тем не менее, существенно влияют на качество жизни. Мы, как и десять, и пятнадцать лет назад справедливо возмущаемся корявыми изделиями наших

горе-«айтишников». История с эпидемией, из-за которой приходится работать удалённо, показала, что заявленные амбиции, прямо скажем, и близко не соответствуют реальной потребности в качестве программного обеспечения, например, для технологии массового дистанционного проведения учебных занятий. Тем не менее – вдумаемся. Материальные орудия труда совершенствовались тысячелетиями, пока не стали эргономичными, пока их создатели не поняли особенности человеческого восприятия инструмента. История реального взаимодействия массового пользователя и нематериального инструмента насчитывает от силы 40 лет. За это время кардинально изменились возможности представления информации, каждый следующий прорыв «отрицает» предыдущие находки, и юное поколение мастеров не успевает вникнуть в фундаментальные идеи предыдущего поколения, увлекаясь возможностями в ущерб потребностям. Стремление быстро написать программу и показать миру своё мастерство провоцирует некачественную работу, которая, тем не менее, нравится автору. А когда он начинает понимать свои ошибки, он вдруг становится «стариком», и следующая поросль идёт мимо по тем же граблям.

Но надежда, что потребности однажды будут услышаны теми, кто имеет возможности, живёт, и залог этого – труд преподавателей, мастеров и дееспособных ветеранов.

Литература

1. *Купер А.* Психбольница в руках пациентов. – СПб: Символ–Плюс, 2004.
2. *Платт Д.* Софт – отстой! И что с этим делать? – СПб: Символ–Плюс, 2007.
3. *Гласс Р.* Программирование и конфликты 2.0. – СПб: Символ–Плюс, 2010.
4. *Лукин В.В., Лукин В.Н., Лукин Т.В.* Технология разработки программного обеспечения: учеб. пособие. – 4–е изд. – М.: Вузовская книга, 2019. – 294 с.: ил.
5. Стандарты: http://www.gost.ru/, обновления и новые: http://protect.gost.ru/.
6. Стандарты: ISO/IEC 25000:2014(en) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE
7. Стандарты: ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models ГОСТ Р ИСО/МЭК 25010–2015 Информационные технологии. Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов.
8. Стандарты: ГОСТ Р ИСО/МЭК 9126–93. Оценка программной продукции. Характеристики качества и руководства по их применению.
9. *Рейнвотер Дж.* Как пасти котов. Наставление для программистов, руководящих другими программистами. – СПб.: Питер, 2008.
10. *Терехов А.Н.* Технология программирования: Учебное пособие. – М.: Интернет–Университет Информационных Технологий; БИНОМ. Лаборатория знаний, 2006.
11. *Быстрицкий Н.Д.* Методика и инструментальное средство оценки корректности функционирования информационных ресурсов: Дисс. … канд. технических наук, 2018.
12. *Гласс Р.* Факты и заблуждения профессионального программирования. – СПб: Символ–Плюс, 2008.
13. *Лукин В.Н., Бахиркин М.В.* Динамическая оценка времени разработки программных систем. Монография. – М.: Изд-во МАИ, 2019. – 160 с.: ил.
14. *Бахиркин М.В., Зинченко А.С., Кирпичников А.П., Лукин В.Н., Ткаченко Д.П.* Модель динамической оценки стоимостных, временных и функциональных показателей процесса про-

ектирования и разработки программ и программных систем. Вестник Казанского технологического университета. 2014, т.17, № 7, с. 284–290.

15. *Зотова А.А., Лукин В.Н.* Пользовательский интерфейс: история и современность. «Новые информационные технологии». Тезисы докладов XV Международной студенческой школы-семинара – М.: МИЭМ, 2007. – с. 50–55.
16. *Дидзигури Д.Г., Лукин В.Н.* Проблемы построения пользовательского интерфейса для слабовидящих. Моделирование и анализ данных: Труды факультета информационных технологий МГППУ; вып. 3 – М.: МГППУ, 2007. – с. 51–59.
17. *Лукин В.Н.* Подготовка качественных программистов: проблемы обучения. Моделирование и анализ данных: научный журнал. 2017, № 1. – М., 2017, с.29–41.
18. *Лукин В.Н.* Качественные программы как предмет обучения. Моделирование и анализ данных: Труды факультета информационных технологий МГППУ; вып. 4 – М.: МГППУ, 2009. – С. 54–65.
19. *Кауфман В.Ш.* Ниша занятости дееспособных ветеранов: https://kaufmanict.eu/ideas/Privychnologija_vs_nedomyslije.htm

Software Quality Problems

Lukin V.N.*
MSUPE, Moscow, Russia,
e-mail: lukinvn@list.ru

We consider the "ever-living" problem of software quality, which is even more acute because software products are built into everything that can or cannot be. What is meant by quality? What is its criteria? Is it possible to measure it? What hinders quality and how to improve it?

Keywords: Software quality, quality standards, reliability, usability, security, quality assessment, quality management, training in quality development.

References

1. Kuper A. Psikhbol'nitsa v rukakh patsientov. – SPb: SimvoL–Plyus, 2004.
2. Platt D. Soft – otstoi! I chto s ehtim delat'? – SPb: SimvoL–Plyus, 2007.
3. Glass R. Programmirovanie i konflikty 2.0. – SPb: SimvoL–Plyus, 2010.
4. Lukin V.V., Lukin V.N., Lukin T.V. Tekhnologiya razrabotki programmnogo obes-pecheniya: ucheb. posobie. – 4–e izd. – M.: Vuzovskaya kniga, 2019. – 294 p.: il.
5. Standarty: http://www.gost.ru/, obnovleniya i novye: http://protect.gost.ru/.
6. Standarty: ISO/IEC 25000:2014(en) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE
7. Standarty: ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quali-ty models GOST R ISO/MEHK 25010–2015 Informatsionnye tekhnologii. Sistem-naya i programmnaya inzheneriya. Trebovaniya i otsenka kachestva sistem i programm-nogo obespecheniya (SQuaRE). Modeli kachestva sistem i programmnykh produktov.
8. Standarty: GOST R ISO/MEHK 9126–93. Otsenka programmnoi produktsii. Kharak-teristiki kachestva i rukovodstva po ikh primeneniyu.
9. Reinvoter Dzh. Kak pasti kotov. Nastavlenie dlya programmistov, rukovodyashchikh drugimi programmistami. – SPb.: Piter, 2008.
10. Terekhov A.N. Tekhnologiya programmirovaniya: Uchebnoe posobie. – M.: InterneT–Universitet Informatsionnykh Tekhnologii; BINOM. Laboratoriya znanii, 2006.
11. Bystritskii N.D. Metodika i instrumental'noe sredstvo otsenki korrektnosti funktsionirovaniya informatsionnykh resursov: Diss. … kand. tekhnicheskikh nauk, 2018.
12. Glass R. Fakty i zabluzhdeniya professional'nogo programmirovaniya. – SPb: SimvoL–Plyus, 2008.
13. Lukin V.N., Bakhirkin M.V. Dinamicheskaya otsenka vremeni razrabotki programm-nykh sistem. Monografiya. – M.: Izd-vo MAI, 2019. – 160 p.: il.

For citation:
Lukin V.N. Software Quality Problems. *Modelirovanie i analiz dannykh = Modelling and Data Analysis*, 2020. Vol. 10, no. 1, pp. 140–156. DOI: 10.17759/mda.2020100109 (In Russ., abstr. In Engl.)

Lukin Vladimir Nikolaevich, PhD (Physics and Mathematics), Associate Professor, Moscow State University of Psychology and Education, Moscow, Russia, e-mail: lukinvn@list.ru

14. Bakhirkin M.V., Zinchenko A.S., Kirpichnikov A.P., Lukin V.N., Tkachenko D.P. Model' dinamicheskoi otsenki stoimostnykh, vremennykh i funktsional'nykh poka-zatelei protsessa proektirovaniya i razrabotki programm i programmnykh sistem. Vestnik Kazanskogo tekhnologicheskogo universiteta. 2014, t.17, № 7, pp. 284–290.
15. Zotova A.A., Lukin V.N. Pol'zovatel'skii interfeis: istoriya i sovremennost'. «Novye informatsionnye tekhnologiI». Tezisy dokladov XV Mezhdunarodnoi stu-dencheskoi shkoly-seminara – M.: MIEHM, 2007. – pp. 50–55.
16. Didziguri D.G., Lukin V.N. Problemy postroeniya pol'zovatel'skogo interfei-sa dlya slabovidyashchikh. Modelirovanie i analiz dannykh: Trudy fakul'teta infor-matsionnykh tekhnologii MGPPU; vyp. 3 – M.: MGPPU, 2007. – pp. 51–59.
17. Lukin V.N. Podgotovka kachestvennykh programmistov: problemy obucheniya. Modelirovanie i analiz dannykh: nauchnyi zhurnal. 2017, № 1. – M., 2017, pp. 29–41.
18. Lukin V.N. Kachestvennye programmy kak predmet obucheniya. Modelirovanie i analiz dannykh: Trudy fakul'teta informatsionnykh tekhnologii MGPPU; vyp. 4 – M.: MGPPU, 2009. – pp. 54–65.
19. Kaufman V.Sh. Nisha zanyatosti deesposobnykh veteranov: https://kaufmanict.eu/ideas/Privychnologija_vs_nedomyslije.htm