АКАДЕМИЯ НАУК СССР

ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР

СООБЩЕНИЯ ПО ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ ЭВМ

С. В. СОРОКИН

СОПРОВОЖДЕНИЕ РАЗВИВАЮЩИХСЯ ПРОГРАММ

ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР АН СССР

МОСКВА 1986

УДК 519.686

ОТВЕТСТВЕННЫЙ РЕДАКТОР
канд. техн. наук Г. Н. В
Волошин

В настоящей работе рассмотрен круг вопросов, относящихся к эксплуатации системных программ. В ней показано, как возникают обязанности обслуживаемого персонала по обеспечению удовлетворительной эксплуатации от простого консультирования до необходимости сложного преобразования машинного кода в программу на языке высокого уровня. Для создания благоприятных условий сопровождения пользователями эксплуатируемых ими программ предлагается регулирование их взаимоотношений с разработчиками.

Изложенный материал использует более чем десятилетний опыт обслуживания автором ряда трансляторов и диалоговых систем на ЭВМ БЭСМ-6. Данная работа предназначается для всех, интересующихся вопросами сопровождения программ.

С О Д Е Р Ж А Н И Е

В В Е Д Е Н И Е

Коллективное использование программ на вычислительной машине требует создания службы, отвечающей за эксплуатацию программного обеспечения. Она обслуживает программы и их пользователей: устанавливает программы, копирует их для защиты от случайного уничтожения, консультирует пользователей, администрирует или регулирует отношения обслуживающего персонала с пользователями, если требует обслуживаемая программа, выделяет ошибки в ней и отыскивает пути их обхода, а также выполняет другие работы, обеспечивающие пользователям эффективный счёт.

Обслуживаемые программы могут быть как стабильными, так и развивающимися. Если время от времени появляется в эксплуатации следующая версия программы, в которую добавили новые возможности, или просто устранили ошибки, то такая программа будет называться развивающийся, а программа, которая длительное время эксплуатируется не меняясь – стабильной.

Развивающиеся программы часто создают по частям. Сначала разрабатывается так называемое ядро программы готовое к эксплуатации, а потом к ядру добавляются новые функции. Так идет непрерывная разработка. Поскольку в переданных в эксплуатацию версиях программы встречаются и устраняются ошибки, то можно говорить о сопровождении в разработке развивающихся программ.

Сопровождение по определению, записанному в ЕСПД, представляет собой «процесс модификации существующей программы вычислительной машины, обусловленный необходимостью устранения выявленных в ней ошибок и (или) изменения ее функциональных возможностей» [1]. Хотя разработка связана с созданием новой программы, а сопровождение – с модификацией существующей, они зачастую неотличимы, как неотличима непрерывная разработка от сопровождения, обусловленного дополнением программы новыми функциями. Сопровождение, обусловленное устранением ошибок, является отладкой (стадия разработки), перенесенной в стадию эксплуатации. Поэтому, когда сопровождением развивающейся программы начнет заниматься специалист по обслуживанию (обслуживающий), то получится, что к первоначальным разработчикам подключаит-

ся новый. Такое подключение требует четкого разграничения прав и обязанностей сторон, чревато конфликтами и порождает ряд проблем.

Почему возникает необходимость обслуживающему приобрести программистскую квалификацию и заняться сопровождением программ, какие при этом возникают проблемы, какие возможны решения – этим вопросам посвящена данная работа. Материал в ней иллюстрируется примерами обслуживания и сопровождения программ на БЭСМ-6.

1. Консультирование и доработка документации для пользователя

После установки программы обслуживающий консультирует ее пользователей. На консультациях он уточняет свои знания о программе и ее документации через постоянный контакт с пользователями. Ему хорошо известны слабые места, поэтому ему легче, чем другим, доработать документацию.

Ниже излагаются рекомендации по составлению документации, предназначенной пользователю. Она, в частности, может быть оформлена разными способами, в том числе и в соответствии с Государственным стандартом ЕСПД [1].

Любые сведения легче усваиваются, если они изложены наглядно. Для программ этого лучше всего можно добиться пользуясь примерами. Первый пример должен быть предельно простым и отражать назначение программы. В нем следует изложить цель, которая возникает почти у всех начинающих пользователей.

Например, в некоторых диалоговых системах при первом сеансе работы пользователя происходит регистрация, в которой у него запрашиваются некоторые сведения, скажем номер полки, куда оператору ЭВМ следует положить выдачу. Таким образом, одной из целей первого примера может быть регистрация. Второй и главной целью первого примера может быть ввод пользовательской программы на магнитный диск, ее коррекция, выполнение и удаление с диска. Эти цели достигаются подробным описанием действий пользователя и ответов ему ЭВМ. В действия человека надо внести типичные ошибки, чтобы, демонстрируя диагностику сбойных сообщений, по возможности уберечь неопытного пользователя от непонятных ситуаций. Попутно было бы неплохо сообщить сведения общего характера, которые следует знать пользователю.

Теперь, когда он ориентируется в основных функциях системы, в следующих примерах ему можно показать рекомендуемые приемы работы, если некоторые цели он может достичь разными средствами. Примеры следует составлять так, чтобы пользователь, познакомившись с ними, мог самостоятельно работать с обслуживаемой программой.

Для оптимальной работы с ней, пользователю нужен список всех ее команд с полным перечислением их возможностей. Такие списки есть, как правило, в документации к каждой программе, но, к сожалению, часто этими списками документация и ограничивается.

Время от времени пользователю приходится встречаться с незнакомой для него диагностикой или какой-либо другой особенностью программы. Лучше, если сведения о них он найдет в документации, в которую необходимо включить еще сведения о всех особенностях программы, о каждом тексте диагностики и его назначении.

Теперь об ошибках в программе и способах их обхода. Автор программы никогда не поместит их в документацию, потому что если они обнаружены, то должны исправляться. Но в действительности процесс исправления автором часто затягивается. Пользователю необходимо знать пути обхода ошибки, если он натолкнулся на нее, а значит документация должна содержать сведения об ошибках и путях их обхода. Когда они будут устранены, информация о них исключается из документации.

Документацию для пользователя следует располагать на магнитном диске или на магнитной ленте, чтобы можно было легко вносить изменения, печатать столько экземпляров, сколько нужно, и только необходимые страницы.

В последнее время, когда появились достаточно емкие и быстрые устройства внешней магнитной памяти, стала развиваться идея давать пользователю инструкцию по частям во время диалога. Например, приказы «Школа» и «Информация» в системе Джин [2], всевозможные «меню», «помощь», подсказки [3, 4]. Такие системы действительно помогают автоматизировать консультацию, но все же не на сто процентов. Во-первых, попадаются расхождения между текстом подсказки и реальным содержанием приказа. Во-вторых, на обслуживающем остается такая функция, как толкование недостаточно ясных мест в подсказке. А в-третьих, у пользователя должна быть уверенность, что в случае необходимости ему всегда можно будет обратиться к человеку, который поймет его и поможет.

Итак, развитие идеи подсказки представляет пользователю дополнительный комфорт, сокращает его время на поиск нужного места в документации и автоматизирует консультации, сокращая на них время обслуживающего. Но полностью освободить его от консультаций подсказка не может, да это и не нужно, так как через консультации он получает сведения о работе программы, об ошибках в программе. В процессе консультирования также могут возникнуть идеи об улучшении программы.

2. Обеспечение работы программы с ошибками

О многих ошибках в эксплуатируемой программе обслуживающий узнает во время консультирования пользователей, когда они просят его помочь разобраться в непонятных для них действиях программы. Сведения об ошибках, конечно, сообщаются разработчику, чтобы он устранил их, а пока они не устранены, ищется способ их обойти (выполнить программу, чтобы они не проявлялись) и дать возможность пользователям продолжать свою работу.

В одних случаях операторы или команды, которые неправильно работают, можно заменять другими операторами или командами. В других случаях можно изменить форму представления данных, которая не очень существенна для пользователя, например, вставлять лишний пробел или удалить пустую строку.

Еще один тип ошибок связан с нарушением структуры данных на магнитном диске. Был период, когда из-за ошибки в обслуживаемой диалоговой программе коллективного пользования портился общий каталог и ни один пользователь не мог с ней работать. На этот период было организовано дежурство обслуживающих в машинном зале для оперативного восстановления каталога.

Для распознавания ошибок, связанных с нарушением структуры данных, обслуживающему требуется изучить, в каком виде на магнитном диске должны храниться данные. Но часто разработчики неохотно сообщают эти сведения, так как есть опасность, что обслуживающий не все сделает правильно и привнесет новые ошибки. Это может случиться в первую очередь из-за безответственности обслуживающего. Но тогда ему нельзя доверить обеспечение работоспособности программы. Опытный обслуживающий подстрахуется, и его случайная ошибка не будет иметь отрицательных последствий. Если все же разработчик откажется предоставить сведения о структуре данных на магнитных дисках, то эту структуру обслуживающий все равно получает, но только самостоятельно, ставя эксперименты и теряя время.

Если программа содержит много ошибок, обслуживающий может задержать ее запуск в эксплуатацию, оттестировать и предъявить ошибки разработчику. При этом он должен решить, что лучше для пользователя – не работать с этой программой совсем (есть другая) или работать и терпеть ошибки. Ту же операцию нужно проводить и с каждой новой версией программы. Итак, наличие ошибок в программе и стремление обслуживающего уберечь от них пользователей порождает еще одну его обязанность – тестирование.

3. Тестирование новых версий программы

Проверку правильности программы обслуживающему лучше всего начинать на стадии формирования к ней требований, если конечно, разработчик поделится с ним своими планами. Ранее ознакомление с программой позволит увидеть и исправить ошибки, когда они легко исправляются [5, 6], а также позволит легче ее усвоить и узнать многие полезные сведения о ней, которые не попадут в документацию.

Знакомство обслуживающего с тестами программы, знание им типичных ошибок разработчиков позволяет ему определить, что хорошо, а что недостаточно отлажено в программе, и предложить свои тесты, способные обнаружить ошибки. Во время эксплуатации некоторые тесты также служат, как примеры, хорошим дополнением к инструкции для пользователя.

Главное же достоинство тестов с точки зрения обслуживающего – контроль разработчика. Разработчик может оттестировать новую версию своей программы, а может передать ее в эксплуатацию неоттестировав, такое встречается довольно часто. Тогда может оказаться, что обнаруженные ранее ошибки не устранены, что привнесены новые, и пользователям придется искать способы их обхода. Поэтому, как об этом было отмечено в предыдущем разделе, обслуживающему следует тестировать каждую новую версию. К имеющимся тестам необходимо добавлять новые, которые, кроме тестирования новых возможностей, демонстрируют, что в старой версии программы ошибки есть, а в новой они исправлены.

Для того, чтобы сократить время, затрачиваемое обслуживающим на повторное тестирование программы, необходимо автоматизировать очередность пропуска тестов, сбор и накопление результатов тестирования, сравнение полученных результатов с результатами предыдущего тестирования и печать расхождений. Для некоторых программ при этом необходимо создавать имитаторы окружающей программно-аппаратной среды [5, 6].

Итак, тестирование очень трудоемко, поэтому лучше, когда тестируют свою программу сам разработчик или специальная бригада по тестированию [5, 6]. Но в тех случаях, когда это не делается или делается некачественно, тестирование вынужден взять на себя обслуживающий. Поэтому в эксплуатацию вместе с программой и документацией должны передаваться тесты и результаты тестирования.

4. Поиск и устранение ошибок

Тестирование позволяет определить, что в программе есть ошибка, а устранение ее обычно возлагается на разработчика. Но не всегда разработчик может исправить ее так быстро, как хотелось бы пользователю, или вообще он не считает нужным ее исправлять. То же относится к другим модификациям программы. Переход пользователя на другую аналогичную программу, в которой этих ошибок нет, часто бывает невозможен или нежелателен. В этих условиях обслуживающему ничего другого не остается, как самому вносить изменения в программу.

Ему, чтобы заняться сопровождением, необходимо овладеть, если он не владел, программированием на языке, на котором написана программа, и потратить много сил на ее изучение. Кроме того, у него нет уверенности, что с большим трудом разобранный модуль программы не будет заменен разработчиком на другой в очередной версии. Так возникает задача быстрее, чем заменится версия, найти и исправить ошибку, или произвести другую необходимую модификацию.

Если программа написана с учетом ее дальнейшего сопровождения или если автор ее с готовностью тратит свое время на объяснение смысла тех или иных частей своей программы, то поиск и исправление ошибки в чужой программе не вырастает в проблему. Но так бывает не всегда.

Самый простой способ найти нужное место в огромном тексте неструктурированной программы без посторонней помощи состоит в следующем.

Выдвигаются гипотезы о причине и месте ошибки, которые охватывают все возможные случаи. Проверяется истинность гипотез и охват ими всех случаев. При проверке находится место и причина ошибки.

Рассмотрим пример поиска во время выполнения программы. Изменилось расположение транслятора на диске, надо найти и исправить его вызов из другой программы. Выдвигается две гипотезы: одна – экстракод сдвига определяет место транслятора на диске, другая – это место задается в константе экстракода обмена. В отладчике [7] задаются остановы по экстракоду обмена и экстракоду сдвига. Во время останова анализируется, какая гипотеза истинна, а какая – ложна. Так определяется способ и место вызова транслятора, что позволяет исправить его вызов.

Пример поиска по тексту программы. В программе, в которой дозаказывается бобина, выдается диагностика «НЕТ БОБИНЫ». И бобина установлена, и диагностика выдается – значит есть ошибка. Выдвигается гипотеза, что ошибка происходит про организации опознания бобины, и фрагмент программы с опознанием находится рядом с операто-

ром выдачи диагностики. Он ищется с помощью программы-редактора в исходном тексте программы по контексту «НЕТ БОБИНЫ». Найденный фрагмент анализируется, и в данном случае гипотеза подтвердилась. В этот фрагмент вносится изменение, и таким образом ошибка исправляется.

Иногда не удается сформулировать гипотезу или не удается определить алгоритм автоматизированной проверки гипотезы. Тогда применяется метод сравнения. Пусть $\{a_1, a_2, a_3, \dots\}$ след (логический путь выполнения программы или, другими словами, результат прокрутки) правильный работы программы с конкретными данными в конкретной программно-аппаратной среде. Пусть $\{b_1, b_2, b_3, \dots\}$ – след неправильной работы программы. Или данные, или программно-аппаратная среда программы в этом запуске несколько отличаются от предыдущего запуска. Множество расхождений Д $\{i: a_i \neq b_i\}$ анализируется и определяется причина этих расхождений.

Пример. Обслуживающий сменил внешнюю программу (изменилась программно-аппаратная среда). Пользователь требует вернуть старую внешнюю программу, так как его перестала работать. Ошибку у себя он не видит, так как ни его программа, ни занные, по его мнению, не менялись. Анализ расхождений, проведенный обслуживающим по методу сравнения, показал, что ошибка была у пользователя, который забыл про свои измененные данные на магнитной ленте. Итак, возвращать старую внешнюю программу не надо, так как в новой нет ошибки, просто совпали по времени смена внешней программы и проявление ошибки пользователя.

Метод сравнений, кроме поиска ошибок в данных и программно-аппаратной среде, можно использовать при поиске ошибок в программе. В этом случае подбираются два слегка отличающихся теста. С одним программа работает правильно, с другим – ошибочно. В множестве Д игнорируются расхождения за счет различия тестов. Остальные элементы определяют место и причину ошибки. Проблемы здесь следующие:

а) необходимы автоматические средства для получения следа (прокрутка);

б) если след очень велик, то необходимы средства для автоматического получения множества Д (компараторы [6, 8]);

в) необходимо взаимодействие вышеупомянутых средств (транслятор, отладчик, прокрутка, компаратор).

И еще несколько слов о проверке корректности исправления ошибок. Если исправляется константа или переменная, то надо проверить, используются ли они где-нибудь еще или нет. Если во всех случаях требуется исправленная константа или переменная, то можно ее исправить, если же не во всех случаях – то надо завести новую. То же самое от-

носится и к модулю, в котором производится исправление. Если к нему имеется несколько обращений, то следует проверить, как исправление отразится на каждом из них.

После исправления надо составить тест, который бы иллюстрировал ошибку и исправление ее. Этот тест должен входить в систему тестов. Далее, для подстраховки от привнесения ошибки, имеет смысл проверить программу на всей системе тестов [8].

5. Поиск нестабильных ошибок

В процессе отладки, когда не удается увидеть ошибку при помощи листинга, используют промежуточные печати, с помощью которых определяют место проявления ошибки: до печати или после. Для этой цели в программу вставляют операторы промежуточной печати и выполняют ее с данными, на которых программа работает с ошибкой. Подобный процесс настолько широко распространен, что, по-видимому, хорошо известен каждому программисту [9]. Нестабильные же ошибки – ошибки принципиально иные, и многим программистам никогда не приходилось с ними иметь дело. Нестабильную ошибку, в отличие от обычной, стабильной, нельзя искусственно воспроизвести, т. е. не удается подобрать такие данные, при которых она себя проявит.

Такие ошибки встречаются в системах разделения времени, в файлово-редактирующих системах. В системах разделения времени программа может быть в любой момент прервана, и ей будут переданы новые данные для обработки. Эти данные могут быть уникальными, и часто невозможно определить, какими же они были в момент ошибки. В файлово-редактирующих системах содержимое файла меняется в процессе выполнения приказов пользователя, и он может не помнить содержимого файла в начале работы с программой, а также может не помнить последовательности своих приказов. Более того, часто встречаются случаи, когда пользователь считает содержимое файла одним, а на самом деле оно другое, т. е. при тех неповторимых данных, которые сложились у пользователя, ошибка проявляется, а при похожих данных у обслуживающего ошибка не проявляется.

Это очень тяжелая ситуация для обслуживающего. Можно очень долго на разные лады варьировать данные, а ошибка так и не проявится. Но ошибка в программе осталась. Они может возникнуть у другого пользователя при совсем других данных, т. е. причина ошибки должна быть найдена и ошибка исправлена.

Можно попробовать свести нестабильную ошибку к стабильной. Если бы все данные программы с помощью какой-либо другой программы или аппаратного средства фиксировать и записывались куда-нибудь на внешнюю память, чтобы можно было эти данные

снова передать программе, тогда бы достаточно было выяснить, в какое время работал пользователь с программой, взять соответствующие данные и выполнить программу с ними столько раз, сколько нужно для локализации ошибки.

Но, к сожалению, такое протоколирование не всегда бывает возможным, так как, с одной стороны, необходима сервисная программа, ведущая протокол, а с другой стороны, может не иметься достаточно внешней памяти для хранения этих протоколов. Можно пойти по пути сокращения объема протокола и резкого упрощения протоколирующей программы. Вместо нее в отлаживаемую программу можно вставить так называемые ловушки, которые бы распознавали искомую ошибку и записывали кое-какие переменные программы в протокол. Причем буфер для протокола должен быть закольцован т. е. при переполнении его новая запись пишется в начало, затирая самую старую. Если ловушка обнаружит ошибку, то буфер с протоколами выдается на печать или запоминается для последующего анализа.

При методе ловушек есть опасность, что необходимые для обнаружения ошибки значения переменных не будут зафиксированы. В этом случае надо сменить переменные, записываемые в протокол, или расположить ловушки в других местах программы. Так или иначе, ловушки дают информацию о том, что работает в программе, а что не работает, способствуя поиску ошибки.

Есть еще метод, с помощью которого можно локализовать нестабильные ошибки. Каждый модуль программы перед началом работы должен проверить данные на правильность, а в случае неправильности выдать соответствующую диагностику. Такой путь позволяет приблизить место проявления ошибки к самой ошибке, что также способствует ее отысканию.

Есть еще метод, предложенный Майерсом [5]. Он в равной степени предназначен для поиска как стабильных, так и нестабильных ошибок.

Сначала должна быть составлена серия тестов, которая пропускается через программу. В случае с нестабильной ошибкой эти тесты проходят и ошибку не фиксируют. Вместе с ними анализируется ситуация, при которой проявилась нестабильная ошибка. Автор программы определяет, что сработало в ней правильно, а что – неверно. Так мысленно он подвергает сомнению каждый модуль и затем реабилитирует его на основании тестов, где программа проработала правильно. Остаются модули, которые в одних случаях проработали правильно, а в других – неверно. Далее анализируется, за счет чего они отличаются. Если этой информации недостаточно, то конструируется новый тест, позволяющий прояснить ситуацию и сузить область в программе, которая подозревается на ошибку.

Метод Майерса предназначен для разработчика программы, который хорошо представляет себе ее части и их взаимодействие, может подключить интуицию для поиска ошибки. Но даже примененный автором, метод не гарантирует, что ошибка будет обязательно найдена. В практике встречаются случаи, когда опытные программисты долго и упорно бьются над поиском ошибки и найти ее не могут. Они или переписывают подозреваемую часть программы, или оставляют так как есть, надеясь, что дальнейшая эксплуатация даст результаты, проясняющие место, где находится ошибка.

Итак, для поиска нестабильных ошибок предложено несколько подходов, но ни один из них не дает гарантию, что ошибка будет найдена. Имеются нестабильные ошибки, которые автор программы не может найти, но эти ошибки, по мнению обслуживающего, все же должны быть исправлены. В этих условиях обслуживающий, если он считает себя высококвалифицированным программистом, должен как бы бросить вызов разработчику, проявить изобретательность и суметь самостоятельно отыскать нестабильную ошибку. Именно поэтому обслуживающий, занявшийся сопровождением программы, должен научиться искать ошибки не хуже, чем ее разработчик.

6. Сопровождение программ в машинном коде с помощью ретранслятора

Встречаются ситуации, когда сопровождаемая программа имеется только в машинных или объектных кодах, т. е. исходные тексты программы отсутствуют. Если программа имеет несколько десятков команд, то разбирать ее – сложное дело, а если число команд измеряется сотнями, то это практически невозможно. Делать вставки в такие программы очень сложно, поэтому их обычно не модифицируют.

Если программа небольшая, то ее можно запрограммировать заново. Для больших программ предлагается ретрансляция, т. е. запуск программы-ретранслятора, которая по машинному коду, полученному в результате трансляции с исходного языка, генерирует текст на этом же языке.

Принципиальных трудностей для разработки ретрансляторов нет. Например, для ассемблера БЕМШ написан ретранслятор с МГУ [10]. Сложнее создать ретранслятор для языков высокого уровня. Но еще сложнее создать ретранслятор, когда помимо машинного кода есть близкая программа на исходном языке (аналог) и желательно воспользоваться ее комментариями и именами констант, переменных, процедур и так далее для получения нового текста.

Ретранслятор, работающий с аналогом, должен сначала выделить в программе-аналоге участки, которые функционально совпадают с программой в машинных кодах

(ПМК), потом ретранслировать участки ПМК, функционально несовпадающие с аналогом, и объединить их с участками, выделенными в аналоге. Чем ближе аналог к ПМК, тем легче выделить совпадающие участки и меньше объем ретранслируемых участков в машинном коде.

Выделение совпадающих участков невозможно ни на исходном языке в случае, когда ПМК всю ретранслировали на исходный язык, ни на машинном языке, когда аналог транслируется в машинный код, так как в первом случае имена в программах не будут совпадать, а во втором – не будут совпадать адреса машинных ячеек. Поэтому сравнивать программы следует тогда, когда они будут переведены на некоторый промежуточный, искусственно созданный язык. Тогда сравнивать их можно программой-компаратором.

Идею ретрансляции ПМК с аналогом следовало бы проверить на реальном трансляторе с языка высокого уровня, поскольку такой процесс необходим для сопровождения развивающихся программ, поступающих в эксплуатацию без исходных текстов.

7. Правовой аспект сопровождения

Разработчик программы сам или представляющая его организация, с одной стороны, и пользователь его программы, с другой стороны, во многих случаях не могут договориться о сопровождении. В спорных ситуациях очень часто все решается в интересах разработчика. Он может не дать сведения о внутренней структуре, не дать тесты, не исправлять ошибки и резко осложнить сопровождение пользователем, лишив его исходных текстов и документации по сопровождению. Пользователь же практически ничего не может ему противопоставить. Правовое регулирование взаимоотношений разработчика и пользователя могло бы внести вклад в решение этой проблемы.

В международном плане попытки решить ее в рамках ЮНЕСКО пока не увенчались успехом [11], на государственном уровне проблема тоже не решена. Как пишет Гельб [12], дело в том, что профессиональные программисты не привлечены к ее решению. Ниже предлагаются принципы, которые могли бы быть использованы при решении этой проблемы.

Автор программы защищает свое право на:

– продажу, т. е. на возмещение затрат;

– разработанный алгоритм, который неочевиден и может с успехом использоваться в других программах;

– стиль реализации алгоритма, который автор выбрал по своему вкусу и который обеспечивает дальнейшее развитие программы.

Пользователь защищает право на соответствие эксплуатируемой программы его запросам, т. е. право на сопровождение.

Желательно найти компромисс, который бы удовлетворил обе стороны. Предлагается следующее:

1. Регламентируются условия, при которых пользователь может заняться сопровождением. Если автор программы предпочитает заниматься сопровождением сам, то он обязан удовлетворять все запросы пользователя. Если автор не может это сделать, то он должен сотрудничать с сопровождающим, которого найдет пользователь для удовлетворения своих запросов. Если автор не хочет сотрудничать, то все материалы, необходимые для сопровождения, должны быть переданы сопровождающему, выбранному пользователем.

2. Защита стиля обеспечивается тем, что он формулируется автором, новый разработчик (сопровождающий), в случае внесения правок, его соблюдает в течение некоторого срока (например, два года). Контроль за соблюдением стиля возлагается на автора.

3. Отказ передачи необходимых для сопровождения материалов не должен аргументироваться необходимостью защиты алгоритма, так как алгоритм, представляющий самостоятельный интерес, должен защищаться сдачей его в фонд алгоритмов и программ.

4. Денежные суммы, которые пользователь платит за получение программы, следовало бы со временем уменьшать вплоть до нуля. Тогда бы, с одной стороны, отпали всевозможные ограничения на распространение программ, а заодно, и на их сопровождение, с другой – разработчик смог бы восполнить затраты на их разработку. Чтобы разработчик не боялся, что сопровождающий будет конкурировать с ним, продавая его улучшенную программу, предлагается передавать вместе с программой только документацию для пользователя, а остальную часть. Необходимую для сопровождения, передавать с некоторой задержкой, скажет на полгода. За это время разработчик, если он продолжает развивать свою программу, успеет учесть интересы пользователя и улучшить ее.

З А К Л Ю Ч Е Н И Е

Хорошо поставленное обслуживание позволяет пользователям намного быстрее освоить программное обеспечение вычислительных машин, быстрее преодолевать задержки в работе с программами. Модернизация программ при сопровождении повышает эффективность их использования. Короче говоря, квалифицированное обслуживание и сопровождение резко повышает производительность труда пользователей.

Поскольку в среде программистов еще недостаточно понимается значение обслуживания и сопровождения, то распространено мнение, что с появлением экранных редакторов и таких сервисных средств как «меню» и «помощь», необходимость в обслуживании и сопровождении пропадет, и в качестве примера приводится программное обеспечение для персональных компьютеров. Действительно, появление экранного редактора, «меню» и «помощь», а также применение многих других приемов надежного программирования заметно сократит количество ошибок в программах и данных, позволит получать соответствующие пояснения.

Но избежать ошибок при создании программ пока в принципе невозможно. Остаются ошибки, которые ошибками раньше не считались, потому что, в частности, не были сформулированы многие требования к психологии. Например, время ответа на экран дисплея должно быть предсказуемо пользователем [3]. Языки общения с ЭВМ должны быть такими, чтобы пользователь практически не делал синтаксических ошибок, а если все же они встретятся, то диагностика должна быть понятной.

Что же касается персональных компьютеров, то, несмотря на многие их достоинства, в их программном обеспечении имеется достаточно ошибок, есть претензия к документации, но самый большой недостаток – это огромная масса однотипных программных средств, например, для редактирования, для печати. Происходит это потому, что новый программист видит недостатки имеющихся программ-редакторов, считает слишком сложным для себя вносить исправления и заново разрабатывает еще одну программу. При этом появляются какие-то новые возможности, а какие-то все-таки теряются. Вот и создаются новые однотипные программы, усложняя пользователю процесс освоения программного обеспечения.

Таким образом сопровождение и обслуживание, хотя и в разной степени, но необходимо для очень широкого круга программ. Они заслуживают специального внимания и требуют разработок методических приемов.

Л И Т Е Р А Т У Р А

1. Единая система программной документации – М.: Изд-во стандартов, 1985.

2. Кобелев В. В. Диалоговая сервисная система «Джин» – М.: ИТМ и ВТ АН СССР, 1982.

3. Шнейдерман Б. Психология программирования. Человеческие факторы в вычислительных и информационных системах. – М.: Радио и связь, 1984.

4. Коноплева Н. М. Адаптивная «Меню-система» на средствах стандартной системы управления базами данных.: в сб. Материалы шестой межреспубликанской школы семинара «Интерактивные системы» Книга 2. – Тбилиси: Мецниереба, 1984, с. 401-402.

5. Майерс Г. Искусство тестирования программ. – М.: Финансы и статистика, 1983.

6. Ван Тассел Д. Стиль, разработка, эффективность, отладка и испытание программ. – М.: Мир, 1981.

7. Подергина Н. В., Самофалов В. В. Символьный диалоговый отладчик ОС ДИСПАК (описание языка). – Свердловск: Уральский научный центр АН СССР, 1979.

8. Гласс Р., Нуазо Р. Сопровождение программного обеспечения. – М.: Наука, 1982.

9. Безбородов Ю. М. Индивидуальная отладка программ. – М.: Наука, 1982.

10. Бурцев Ю. В., Якименко А. В. О детрансляторе модулей загрузки автокода БЕМШ.: в сб. Программные и технические средства СКП ЭВМ МГУ. – М.: МГУ, 1984, с. 104.

11. Гельб А. Б. Работы международных организаций по правовому регулированию коммерческой реализации программного обеспечения ЭВМ. – Программирование №3, 1980, с. 88-91.

12. Гельб А. Б. К вопросу о возможности и целесообразности правовой охраны программного обеспечения.: в сб. Всесоюзн. научн. техн. конф. Производство и сопровождение программных средств как продукции производственно-технического назначения. Тез. докл. – Калинин: НПО Центрпрограммсистем, 1985, с. 100-104.