Николай Горбунов

Как «выжать» максимум из технической поддержки производителя

Мы не пишем в техподдержку

В инженерном деле чудес не бывает – говорят, даже для истории с магическим переключателем [1] в конце концов нашлось логичное объяснение. Любая инженерная задача состоит из деталей, и любая проблема заключается в том, что какие-то конкретные детали работают и/или взаимодействуют не так, вне зависимости от того, знаем мы про них что-либо или нет. При этом опытный инженер всегда знает, что в задачах диагностики неисправностей додумывать или подразумевать что-либо очень вредно: как известно, допущение – праматерь всех провалов, и чем больше фактов, тем ближе истина, а по ложному пути можно блуждать сколь угодно долго.

Количество деталей, влияющих на работу технической системы, может быть невообразимо велико. О каких-то из них мы не знаем, потому что они выпадают из нашего кругозора, другие лежат на поверхности, но мы их даже не рассматриваем, наивно полагая, что уж они-то здесь совершенно ни при чём. Это абсолютно нормально: мировая инженерная практика изобилует весёлыми историями про курьёзы отладки наподобие электронной почты, которая не доставляется дальше, чем на 500 миль [2], сторожевой собаки, которая скулит перед тем, как зазвонит телефон [3], постороннего стука в кузове «Газели» [4] и т.п. Я однажды чуть ли не два месяца пытался понять причину стука во входную дверь своей квартиры между восемью и десятью вечера (два или три раза – «тук-тук-тук», подбегаешь, смотришь в глазок – никого), пока однажды, возвращаясь вечером домой, случайно не наткнулся на спускающегося навстречу соседа, который шёл выгуливать своего сеттера. Сеттер был ужасно рад, что его ведут гулять, и ошалело вилял хвостом, проходя мимо моей двери, он как раз успевал ударить по ней хвостом два-три раза. Как сделать время диагностики таких проблем предсказуемым?

Однажды, когда я был молодым специалистом и работал в техподдержке, меня вызвали прочитать мастер-класс на техническом семинаре в Запорожье. Когда семинар закончился, ко мне подошел представитель одного из заказчиков и спросил, известно ли мне что-нибудь про проблему с драйвером некой сетевой карты для операционной системы QNX4. Проблема была знакомой: производитель оборудования заменил микросхему на одной из своих плат, изменил номер версии платы (вытравленный в уголке мелким шрифтом), но номер для заказа оставил неизменным. Разница, как в анекдоте, оказалась потрясающей: со старой версией платы драйвер работал корректно, а с новой «валил» при запуске весь сетевой стек. По закону подлости проявилась эта ситуация в самый неподходящий момент: наш заказчик сделал и отладил прототип системы на плате старой версии, а когда пришло время ставить систему на объект, закупил оборудование по тем же самым номерам для заказа, и система внезапно отказалась работать. Мы подняли по тревоге техподдержку производителя, те признали проблему и через три дня предоставили исправленную версию драйвера; система была успешно развёрнута, а задача со спокойной совестью сдана в архив. С этого момента до семинара в Запорожье прошло больше года. Когда я спросил представителя запорожского заказчика: «У нас уже год как есть решение – почему же вы сразу не написали к нам в техподдержку?» – он ответил: «Мы не пишем в техподдержку. Если мы сталкиваемся с проблемой, которую не можем решить прямо сейчас, мы просто ищем другой подход».

Этот ответ меня тогда сильно озадачил – я не мог понять, почему проще перепроектировать систему, чем написать несколько строк по электронной почте и с некоторой вероятностью получить решение мгновенно. Работая впоследствии в должности менеджера проектов, я нашел ответ на этот вопрос: лучше плохо ехать, чем хорошо стоять. Руководство начинает паниковать, когда работа приостанавливается на неопределённый срок и её возобновление зависит от обстоятельств, которые не выглядят контролируемыми, например от работы инженеров техподдержки сторонней организации. Гораздо спокойнее загрузить своих людей другой работой и надеяться, что пока шар на их стороне, ситуация находится под контролем.

Созданию этой иллюзии во многом способствует реальный отрицательный опыт работы с техподдержкой – многие из вас наверняка сталкивались с ситуацией, когда техническая поддержка производителя либо вообще молчит, либо слишком медленно реагирует, либо реагирует быстро, но обсуждение проблемы затягивается на неопределённый срок и в конечном итоге не приводит ни к чему. Иногда обращаться в техподдержку мешает инженерная гордость (да я сам справлюсь, что я, не инженер?), иногда – интерес к проблеме (когда ещё залезешь в исходный код планировщика?), но результат всегда один: обращение в техподдержку часто используется как последний шанс, когда других вариантов не осталось, а время уже упущено. Между тем, если возникают проблемы, их надо решать быстро, ну или хотя бы предсказуемо.

В настоящей статье описывается, как устроена техническая поддержка, почему она работает так, как она работает, и как взаимодействовать с ней, чтобы получить от этого максимальный эффект. Приведённые рекомендации в большей степени относятся к работе с производителями программного обеспечения, хотя большинство из них будут полезны в любой практике диагностики и решения технических проблем.

Как работает техподдержка

Мухи и котлеты

Прежде всего, следует понимать, что в общем случае (наиболее часто это справедливо для больших компаний)

подразделение технической поддержки и подразделение разработки – это две различные единицы, работающие по разным правилам, использующие разные информационные системы, а часто и находящиеся в разных городах (а то и странах).

Подразделение технической поддержки обрабатывает входящие обращения и решает проблемы клиентов, связанные с продукцией компании. Инженеры технической поддержки знают больше всех про то, что беспокоит клиентов, и про поведение продуктов в типовых случаях, но про их внутреннее устройство могут не знать решительно ничего; как только они узнают достаточно, они обычно переводятся в подразделение разработки, потому что там больше платят. Задача инженеров технической поддержки – избавлять клиентов от возникающих проблем в краткосрочной перспективе, например, снабжать свежими версиями, в которых проблема уже решена, указывать на главы в документации, в которых описывается решение, рекомендовать обходные пути и т.п. Далеко не любая проблема, попавшая в техподдержку, дойдёт до разработчиков, просто потому что большая часть запросов сводится не к некорректному поведению продукта, а к более простым вещам типа недостаточно внимательного чтения документации. Техническая поддержка именно потому выделяется в отдельное подразделение, что отвлекать разработчиков на такие задачи выходит слишком дорого.

Подразделение разработки собственно создаёт продукт. Инженеры-разработчики знают про внутреннее устройство продуктов всё, но в общем случае могут понятия не иметь, как эти продукты применяются реальными заказчиками и с какими проблемами это сопряжено. Инженеры технической поддержки для них в этом смысле – один из важных источников обратной связи (наряду с отделом маркетинга, группой тестирования и т.д.), на основании которой формируются оперативные планы и планы выпуска новых версий. Задача разработчиков – сделать продукт таким, как написано в требованиях, и сделать это в указанные сроки. Работа разработчиков ритмична и основана на планах, и чтобы эти планы поменять, требуются серьёзные основания, кроме того, изменения в планах могут быть значительнее, чем изначально кажется, т.к. после разбирательства может выясниться, что проблема лежит глубже, и чтобы починить всё правильно, требуется серьёзная переработка системы. Поэтому решение часто имеет двоякую форму: обходной путь прямо сейчас, плюс долговременное решение в следующей версии согласно обновлённому плану выпуска.

Из вышесказанного следует один важный вывод: ваш запрос в техподдержку *никогда не попадает к разработчику немедленно* (за исключением разве что случаев, когда компания-производитель невелика и группы разработки и техподдержки сидят в одной комнате или вообще едины). Чтобы проблема дошла до разработчика, необходимо сначала удостовериться в том, что:

- проблема реально существует, то есть продукт не делает то, что должен делать, и это происходит определённым образом в определённых обстоятельствах;
- вы прочитали документацию и следовали ей, но это не дало необходимого решения;
- вы не пытаетесь сделать то, что сделать невозможно или по общепринятой практике делается совершенно иначе;
- проблема не является известной и либо решённой в более свежих версиях продукта, либо отклонённой/отложенной по объективным причинам.

Первоочередная задача технической поддержки – получить от клиента всю эту информацию. Чаще всего ещё в процессе её получения выясняется, что проблема не стоит выеденного яйца и решается в одно действие, но если этого не происходит, то на основании полученной информации принимается решение:

- признавать ответственность за проблему или нет (есть ли в возникновении проблемы вина производителя);
- принимать проблему в работу или отклонять;
- к какой области относится проблема (вызвана ли проблема ошибкой в ТЗ, продукте и/или документации);
- насколько решение проблемы важно относительно уже имеющихся текущих задач.

Процесс принятия этого решения у разработчиков носит название приоритезации (triage), и этим занимается специальная рабочая группа, состоящая из представителей разных подразделений, – с этого может, например, начинаться рабочий день. Пока проблема не прошла приоритезацию, в рабочий план она не попадёт и её решением никто заниматься не будет – это позволяет сохранить концептуальную целостность продукта и не скатиться в хаос при большом количестве заявок.

Таким образом, чтобы проблема была признана и ушла в работу как можно быстрее, всю необходимую информацию для этого надо предоставить сразу – это позволит сократить административные задержки, и вот почему.

Жизненный цикл заявки и системы учёта

Число заявок, поступающих в техподдержку за единицу времени, для крупных компаний (да и не только для крупных) может быть очень велико; при этом хорошим тоном считается реакция на заявку в течение одного рабочего дня. Чтобы уложиться в нормативы, нужно всегда иметь точную картину по текущим заявкам: что с ними происходит, кто с ними работает, какие из них к чему относятся, какие из них самые важные, и т.д. Вдобавок, кроме отслеживания текущего состояния заявок, важно ещё и строить статистику: сколько заявок по какой теме поступило за последний месяц, растёт число незакрытых заявок со временем или падает, кто из инженеров результативнее всех, и т.п.

Очевидно, что Excel здесь не обойтись, – мне доводилось работать в компаниях, где число заявок в базе данных переваливало за десятки тысяч. Чтобы ориентироваться в таких объёмах информации, нужна распределённая информационная система, способная учитывать и отслеживать состояние заявок, а также генерировать сложные отчёты. Для этого существует специальный класс программных продуктов – *системы отслеживания дефектов* (issue tracking systems, например, JIRA, TestTrack Pro, FogBugz, Mantis, Bugzilla – список можно продолжать). Если вы работаете с серьёзным производителем, будьте уверены, у него такая система есть.

Каждая поступающая в техническую поддержку заявка регистрируется в системе и получает уникальный номер (идентификатор). После этого она обязана пройти определённый перечень стадий («не трогали», «расследуем», «чиним», «ждём информации», «проверяем решение» и т.п.), пока не получит резолюцию («это уже было», «исправлено», «отклонено» и т.п.)

и не будет закрыта по согласию с заявителем. Жизненный цикл заявки у каждой организации свой – всем удобно по-разному, и системы отслеживания дефектов позволяют это настраивать, но важно не это. Важно то, что жизненный цикл есть, для всех заявок он неизменен, и по мере продвижения по нему заявка обрастает подробностями – мало того, часто нельзя перевести заявку из одного состояния в другое, не предоставив определённую информацию.

Если посмотреть с точки зрения клиента, то можно сделать из этого следующие выводы.

- **Спрашивайте номер вашей заявки**. Если в ответ на вашу заявку вам первым делом сообщили её номер (ticket number, issue id – это может называться по-разному) – это хороший признак. Это значит, что у производителя есть система учёта, так что все детали по заявке будут храниться в одном и том же месте, ничего не потеряется, вы всегда будете в курсе, о чём идёт речь, а главное, сможете вспомнить всё необходимое, когда та же проблема снова объявится через полгода.
- **Предоставляйте всю требуемую информацию как можно раньше**. Необходимость предоставить информацию, запрошенную инженером техподдержки, продиктована в том числе и тем, что не имея её, он физически не сможет перевести заявку в состояние «принято в работу» (или как там у них), а значит, работа по заявке всё это время вестись не будет.

Кстати, у этого вопроса есть и своя тёмная сторона.

Игры, в которые играют люди

Требовать номер заявки имеет смысл ещё и с той точки зрения, что данные, не введённые в систему, не попадают в отчётность. Как совершенно справедливо писал Джоэл Спольски [5], как только вы вводите в своей организации показатели качества, организация перестаёт работать на качество и начинает работать на показатели. Эффективность работы инженеров технической поддержки в разных компаниях измеряется по-разному (см. комиксы Скотта Адамса «Техподдержка Догберта» [6]), но есть один общий момент: всегда можно легко посчитать (и поэтому все этим занимаются), сколько заявок назначено каждому инженеру. Разумеется, никто не любит, когда на нём «висит» много незакрытых заявок, поэтому есть соблазн либо регистрировать не все заявки, либо искать причины их отклонять, либо как можно быстрее «переводить стрелки», а то и комбинировать всё это. (Джоэл Спольски упоминает историю про одного программиста, который должен был написать функцию, возвращающую высоту строки электронной таблицы, но у него не было времени сделать это правильно, потому что начальство давило на него нереальными сроками. Он вышел из положения, написав функцию, возвращавшую константу «12», а потом дождался обнаружения этой ошибки и выделения рабочего времени на её исправление.)

Это выливается в серию игр, которые могут быть на руку конкретному недобросовестному сотруднику производителя, но совершенно не на руку клиенту, который хочет быстро решить проблему. Рассмотрим некоторые из них подробнее – в конце концов, предупреждённый вооружён.

Какой-такой номер? Случается так, что у производителя есть система учёта заявок, но она, как это часто бывает с корпоративными информационными системами, разработана не для людей и внедрена кое-как, соответственно, работать она не помогает, а только мешает. Сотрудники будут всеми правдами и неправдами избегать использования такой системы: применительно к технической поддержке, например, это значит не регистрировать заявки, а хранить их у себя в почтовом ящике, или регистрировать, но только образцово-показательные, или ещё как-нибудь. Трюк вроде безобидный, но на практике это всегда означает путаницу с приоритетами и потерю важных деталей – всё это откладывает решение вашей проблемы. **Вывод**: лучше требуйте номер заявки сразу и явно – даже если у вашего производителя нет системы учёта, вы, возможно, приблизите этим момент её внедрения.

Имитация бурной деятельности. Другой возможный трюк – активное переливание из пустого в порожнее в надежде, что клиенту надоест и он отстанет сам. Мне однажды довелось столкнуться с инженером техподдержки, который отвечал мгновенно и вроде бы по делу («попробуйте вот это»), но на самом деле вникнуть в проблему даже не пытался – это всегда видно по отсутствию встречных вопросов. На пятой итерации (на этот момент мы потеряли уже неделю) я вынужден был выйти на менеджмент верхнего уровня, и меня связали непосредственно с разработчиком, который и помог найти решение. **Вывод:** отсутствие встречных вопросов – тревожный признак, означающий нежелание помочь. Будьте готовы искать альтернативные пути и запаситесь соответствующими контактами, лучше из другого подразделения (например, отдела маркетинга или продаж), а также близкой для них аргументацией (например, возврат оборудования).

«У нас нет на это времени». Далеко не все заявки в результате сводятся к дефектам. Формально говоря, дефект – несоответствие *реального* поведения продукта *заявленному* (сравните с *ожидаемым*); таким образом, однозначно является дефектом только поведение продукта, противоречащее его опубликованным характеристикам или документации. Это очень полезное определение, так как оно не оставляет места для полемики: если такое противоречие обнаружено, производителю остаётся признать либо ошибку в описании, либо ошибку в продукте. Однако у такого определения есть и оборотная сторона – оно позволяет производителю считать все остальные заявки запросами на добавление новых функций (feature request) или улучшение уже существующих (enhancement request) и принимать решение об их принятии в работу или отклонении самостоятельно, считая обратную связь с пользователем сугубо рекомендательной. И тут мы сталкиваемся с процессом управления выпуском продукта (release management).

Подобные решения находятся вне компетенции технической поддержки – их принимают те, кто составляет планы выпуска и рабочие графики. Как уже упоминалось, чтобы изменить рабочий график, нужны серьезные основания, так как разработчики всегда работают с перегрузкой (не спрашивайте, почему), и чтобы добавить в план новую задачу, нужно решить, что ради этого придется выкинуть. В этом нет ничего удивительного, если посмотреть на процесс разработки изнутри: на выпуск продукта отводится определённое количество календарного времени, и его нужно «справедливо» поделить между добавлением новых функций, на которых настаивают отделы маркетинга и продаж, а также улучшениями и устранением уже зарегистрированных дефектов, обнаруженных группой обеспечения качества и пользователями. Поскольку количество имеющихся ресурсов не бесконечно, уже на этом этапе возникает нешуточная потасовка между заинтересованными сторонами на предмет того, что включить в следующий выпуск, а что отложить на потом. У «взрослых» компаний для этого в процессе управления выпуском продукта есть формальная процедура ранжирования, позволяющая определить, что более важно и должно быть сделано обязательно (must-

have), что менее важно, но было бы полезно (like-to-have), а что заслуживает внимания, только если останется время (nice-to-have, – сюда обычно никто даже не заглядывает). В результате каждая из заинтересованных сторон, в том числе реальные пользователи, может претендовать на строго определённый процент рабочего времени инженеров.

Всё это хорошо работает в теории (если не считать разбитых носов на этапе ранжирования), с наивным допущением, что однажды принятый план не будет меняться и одной потасовки достаточно. Однако правда такова, что всё предвидеть невозможно, и планы приходится регулярно корректировать. В частности, обнаружение пользователем критического дефекта в середине рабочего цикла означает, что если на подобные случаи не предусмотрен временной резерв (или «гауптвахта»), то выделенное на исправление дефекта время нужно будет от чего-то отнять. Ещё хуже дела обстоят с дефектами некритическими или запросами на улучшение: фактически, чтобы план сохранял концептуальную целостность, процедуру ранжирования (а значит, и потасовку заинтересованных сторон) нужно повторять для каждой входящей заявки. Это утомляет и создаёт соблазн свести всё к более простой процедуре «кто громче всех кричит», а от остальных отмахиваться уклончивой формулировкой «у нас нет на это времени». С пользователями это работает лучше всего, так как рабочих планов производителя им «снаружи» не видно и они не могут проверить, насколько действительно важны задачи, ради выполнения которых их заявки были отклонены, – служба техподдержки просто транслирует им отрицательное решение. В совсем «клинических» случаях заявки могут отклоняться с аргументом «у нас нет на это времени», даже если ресурсы есть в избытке, просто напрягаться неохота.

Эффективных контрстратегий здесь за отсутствием рычагов давления нет и быть не может, однако есть один хороший способ повысить шансы на попадание заявки в рабочий план: если предметом вашей заявки является не критический дефект, аргументируйте необходимость её выполнения. Опишите, насколько проблема мешает вам в вашей работе и как её решение могло бы улучшить вашу жизнь, а возможно, и не только вашу, и принести производителю прибыль. Хороший менеджер по продукции не упустит возможности сделать продукт лучше, но, не видя реального применения продукта, он может чего-то не понимать. Убедите его, а он сделает всё остальное.

«У нас всё работает». Этот трюк также широко известен под названием «с нашей стороны пули вылетают». (Кстати, не стесняйтесь упоминать эту фразу, когда с вами проделывают такое, – помогает.) Одним из стандартных вариантов резолюции по заявке является «не удалось воспроизвести» (could not reproduce – CNR). Это означает, что последовательность действий, указанная клиентом в заявке как приводящая к ошибке, была повторена на стороне производителя и к ошибке не привела. Это даёт производителю полное право не признать дефект и требовать уточнений, а при отсутствии таковых – прекратить разбирательство и закрыть заявку как необоснованную.

Требование воспроизводимости проблемы возникло не на пустом месте: если проблему нельзя воспроизвести, её нельзя диагностировать. Загвоздка в том, что для воспроизведения проблемы на стороне производителя необходимо, во-первых, в точности повторить начальные условия (программно-аппаратную конфигурацию, настройки, внешние факторы и т.п.), а во-вторых, в точности повторить последовательность дей-

ствий, приводящую к ошибке из начальных условий. Поскольку заранее никогда не известно, какие факторы влияют на проблему, а какие нет, перечень начальных условий и шагов по воспроизведению проблемы таит в себе бесконечное число поводов для спекуляций. Как следствие, самый простой способ спустить проблему на тормозах – это заявить: «У нас всё работает» – и передать ход клиенту; через несколько итераций игры в «угадайку» он либо отстанет сам, потому что уже поджимают сроки, либо случайно наткнётся на решение. **Вывод:** не ленитесь описывать воспроизведение проблемы и приводить соответствующие иллюстрации. Чем подробнее проблема описана, тем сложнее будет выставить её как невоспроизводимую.

Кстати, о воспроизведении проблем.

Как их готовить

Как уже упоминалось, очень немногие инженеры обращаются в техподдержку сразу же, как только столкнутся с проблемой (если обращаются вообще), отчасти потому что решение технических проблем само по себе является увлекательнейшей задачей. Однако у самостоятельного решения проблем под давлением сроков есть неприятный побочный эффект: действия часто предпринимаются непоследовательно, наугад, а в худшем случае их результаты ещё и влияют друг на друга. В результате на момент, когда становится ясно, что без обращения в техподдержку не обойтись, инженер находится в состоянии, близком к унынию, и составленный им запрос не превосходит по информативности всхлипы утопающего: «После подключения платы X не запускается графическая оболочка. Просим содействия».

Между тем чудес не бывает (по крайней мере, в технике), и к любой проблеме всегда ведёт одна или несколько чётко определённых причинно-следственных цепочек. Чтобы диагностировать проблему, в первую очередь надо выявить эти цепочки – это позволит *воспроизводить* проблему «по заказу», а значит, и однозначно определить, решена она или ещё нет. В свою очередь, чтобы минимизировать количество усилий по воспроизведению проблемы, её необходимо *локализовать*, то есть найти минимальную конфигурацию системы, в которой проблема проявляется. Это значительно увеличит шансы, что производитель сможет повторить указанные начальные условия без необходимости что-то додумывать («Требуются телепаты. Звонить сами знаете куда») или собирать у себя в лаборатории космический корабль целиком.

Локализации неисправностей, кстати, сильно способствует так называемый хороший стиль программирования. Однажды программист заказчика принёс мне распечатку из 10 страниц исходного текста на Си, в котором системный код был настолько густо перемешан с прикладным (представьте себе коктейль из обращений к портам ввода/вывода, системных вызовов ОС и обновлений графического интерфейса в одной 3-страничной функции), что разобраться в нем было совершенно невозможно, и в гневе произнес: «Ваша операционная система не работает». Потеряв три дня на разбирательства, мы выяснили, что в программе просто неправильно закодирован один оператор *if-then-else*, обрабатывающий значение *errno*. Будь код структурирован, мы потратили бы час.

В индустрии ПО, кстати, диагностике проблем посвящён целый культурный пласт, и особого внимания в нём заслуживает классификация дефектов по воспроизводимости. Согласно классификатору из известного документа jargon.txt [7], различают следующие виды дефектов:

- **дефекты Бора** (Bohr bugs) стабильно проявляются при одних и тех же обстоятельствах (не обязательно известных) и ведут себя детерминированно;
- **дефекты Гейзенберга** (heizenbugs) исчезают или меняют своё поведение при попытке их диагностировать (например, при подключении отладчика);
- **дефекты Шредингера** (schroedinbugs) присутствуют в системе, до поры никак не проявляясь. Будучи случайно обнаруженными (например, при анализе исходных текстов или принципиальных схем), внезапно проявляются и рушат всё на свете;
- **дефекты Мандельброта** (mandelbugs) проявляются при столь сложной комбинации обстоятельств, что претендуют на недерминированность (в терминах дефектов Бора).

Смех смехом, но всё это наводит на серьёзную мысль: не одно поколение инженеров разбило себе лоб в попытках воспроизведения дефектов, и иногда им не оставалось ничего, кроме юмора. Такая степень внимания к вопросу воспроизводимости не случайна: как уже упоминалось, если проблему нельзя воспроизвести, её нельзя ни диагностировать, ни решить. Поскольку проблема возникла у вас, то человеком, потенциально обладающим максимумом информации о том, как её воспроизвести, являетесь вы. Исходя из этого (а также всего сказанного ранее), самой выигрышной тактикой работы с техподдержкой производителя является:

1) самостоятельно добиться стабильного воспроизведения проблемы и зафиксировать начальные условия и последовательность шагов, приводящую к её проявлению;
2) максимально упростить начальные условия, чтобы их было легко повторить на стороне производителя, например, последовательно отключать от системы блоки, предположительно на проблему не влияющие, и повторять последовательность шагов, описанную в п. 1. В результате получается минимальная конфигурация, в которой проблема всё ещё воспроизводится;
3) заявить о проблеме производителю в формате «хотим сделать то-то – начальные условия такие-то – предпринимаем такие-то шаги – ожидаем того-то – наблюдаем то-то» (в устоявшейся практике англоязычных разработчиков эта схема называется “steps – expected – observed”);
4) отвечать на встречные вопросы, пока проблема не будет успешно воспроизведена на стороне производителя. С этого момента её начнут решать;
5) запросить оценку трудозатрат по решению проблемы, уточнить её (например, по описанному Эдвардом Йордоном [8] принципу «удвой и добавь ещё») и скорректировать свои рабочие планы соответственно.

Использование такой схемы сильно повышает вероятность, что по вашему описанию проблема будет приоритезирована, принята в работу, а возможно, и воспроизведена – в тот же день. Это важно, так как многие производители находятся в отдалённых часовых поясах, например, разница во времени между Москвой и США составляет 8–11 часов, а значит, даже при самом оптимистичном раскладе вы успеете обменяться с производителем 1–2 письмами в день, и то при условии, что останетесь на работе до поздней ночи. Соответственно, если вы не указываете в заявке какую-либо важную деталь, вас о ней обязательно спросят; каждый такой встречный вопрос автоматически добавит к решению проблемы минимум один рабочий день (а на практике от одного до трёх), и чем больше в вашей заявке белых пятен, тем больше времени вы на этом потеряете.

Как описывать проблему

Понятно, что в идеальном случае информации в описании проблемы должно быть по возможности ровно столько, сколько нужно. Если её будет меньше, вы потеряете время на встречных вопросах или погрязнете в описанных ранее играх, а если больше, потеряете время на её сбор, и в ней будет сложно ориентироваться. Однако суровая реальность такова, что, как уже упоминалось ранее, нельзя заранее предугадать, какие факторы влияют на проявление проблемы, а какие нет, поэтому имеет смысл принять за норму некий разумный минимум. Например, описание начальных условий может иметь следующий вид.

- **Аппаратная конфигурация**:
  - форм-фактор и конструктив;
  - процессорные платы;
  - платы расширения/мезонинные модули;
  - внешние интерфейсы и подключённые к ним устройства.
- **Программная конфигурация**:
  - операционная система (ОС);
  - установленное ПО (системное, связующее и прикладное);
  - параметры и настройки.

Это позволит производителю понять, из чего и как можно собрать минимальный действующий аппаратный макет и какое ПО на него установить, перед тем как начать экспериментировать.

Важно указать не только конкретные модели используемых устройств (производитель, код модели, номер версии), но и то, через какие интерфейсы они подключены и как именно, вплоть до номера слота. Только так можно быть уверенным, что макет, собранный в лаборатории производителя, будет максимально приближен к вашему (или хотя бы различия между ними будут заранее известны). Номера версий плат следует указывать обязательно: случается, что производитель оборудования вносит в дизайн изменения (например, заменяет какую-нибудь микросхему на более новую), а код модели и номер для заказа оставляет прежними – см. историю с семинаром в Запорожье. В результате получается, что с двумя платами одинаковой модели, приобретёнными по одному и тому же номеру для заказа, один и тот же драйвер работает по-разному.

Что касается программного обеспечения, то характер описания сильно зависит от применяемой ОС, так как в каждой ОС конфигурационная информация хранится в различном виде и в различных местах, а диагностические средства реализованы по-разному и выдают информацию в разных форматах. Основной момент здесь – обязательно указывайте полные номера версий: программные продукты, версии которых отличаются одной младшей цифрой, могут различаться как раз в той детали, которая имеет отношение к вашей проблеме. Обычно достаточно указать тип и версию ОС, а также имена и версии установленных программных пакетов; если потребуется предоставить вывод тех или иных диагностических утилит, вас об этом попросят.

По части воспроизведения проблемы лучше всего применять схему “steps – expected – observed”, описанную ранее.

- **Постановка задачи**, то есть какая цель преследуется.
- **Последовательность действий**, то есть что конкретно и как именно надо сделать, чтобы проблема проявилась.
- **Ожидаемый результат**, то есть как система должна себя повести в результате описанной последовательности шагов.

- **Реально наблюдаемый результат**, то есть как система ведёт себя на самом деле.

Постановка задачи здесь приводится не случайно, так как, понимая, что именно клиент хочет сделать, инженеры техподдержки могут предлагать альтернативные решения. Это, кстати, не такая редкая ситуация, как может показаться, – у меня в практике был случай, когда клиент пытался организовать обмен данными между двумя процессами через область разделяемой памяти и столкнулся с проблемой использования дальних указателей (что в защищённом режиме x86 в многозадачной среде, мягко говоря, не самый лучший способ). Проблема была решена раз и навсегда путём замены операций с указателями на POSIX-вызовы *shm_*()*.

Последовательность действий следует описывать по возможности детально: какие действия предпринимаются (вплоть до конкретных щелчков мыши – иногда проблема кроется, например, в неправильно выбранном пункте меню), и что при этом происходит. Если вы действуете по документации, то дублировать её, конечно, нет смысла, но обязательно сошлитесь на конкретные пункты определённого документа, а лучше ещё и приложите его к запросу – документы тоже имеют разные версии, и инструкция, по которой работали вы, может не совпадать с имеющейся у инженеров поддержки.

Указание ожидаемого результата тоже имеет особый смысл, так как иногда выясняется, что поведение, ожидаемое клиентом, не совпадает с поведением, описанным в документации или интуитивно следующим из интерфейса продукта. (Естественно, это с равной вероятностью может означать как ошибку в документации или недоработку интерфейса, так и неправильную трактовку ожидаемого поведения.)

Реально наблюдаемый результат желательно подкреплять подробными иллюстрациями – снимками экрана, файлами журналов, дампами памяти – всем, что может помочь понять, что именно работает неправильно и как конкретно оно это делает. Если вы находитесь с производителем в одном часовом поясе, не лишним будет предложить живую демонстрацию, скажем, с помощью веб-камеры или утилит трансляции рабочего стола (например, Mikogo, GoToMeeting или WebEx).

Хороший пример

В качестве показательного примера эффективного решения проблемы хочется привести недавний опыт по сборке образа ОС VxWorks 6.7 для COM-модуля MEN XM50 (сравните с обычным «не собирается образ»).

Постановка задачи

- Собрать базовый загружаемый образ ОС VxWorks 6.7 для COM-модуля MEN XM50.

Начальные условия

- Аппаратная часть:
  - инструментальный компьютер: P4 2,8 ГГц, ОЗУ 2 Гбайт.
- Программная часть:
  - инструментальная ОС: Windows XP Pro 2008 SP3;
  - инструментарий: Wind River VxWorks Platform GPP v3.7 (Workbench 3.1, VxWorks 6.7);
  - BSP: MEN VxWorks BSP для XM50 rev.1.5, номер для заказа 10EM09-60.

Последовательность шагов

1. На инструментальном компьютере распаковываем архив с BSP (10EM09-60.zip) во временную папку и выполняем установку и настройку BSP, как указано в файле README.txt из архива, пп. 1–4.
2. На инструментальном компьютере запускаем Wind River Workbench.
3. В Wind River Workbench открываем перспективу Application Development.

НОВОСТИ НОВОСТИ НОВОСТИ НОВОСТИ НОВОСТИ НОВОСТИ

Компания ПРОСОФТ стала дистрибьютором Eurotech

Компания ПРОСОФТ объявила о заключении партнёрского соглашения с одним из ведущих мировых поставщиков встраиваемых систем – компанией Eurotech. Включая в свою программу поставок продукцию Eurotech, ПРОСОФТ выводит на российский рынок широкий спектр встраиваемых решений, от процессорных плат и компьютерных модулей до «облачных» платформ.

Встраиваемые системы компании Eurotech разработаны для применения в жёстких условиях эксплуатации. Портфель продуктов включает в себя процессорные платы современной архитектуры: Atom, x86, PowerPC, XScale и Freescale RISC – и различных форм-факторов: VME, Open VPX, CompactPCI, PC/104, PC/104-Plus, EPIC, EBX, COM-Express, SO-DIMM, а также форм-факторов разработки Eurotech.

Основной стратегией Eurotech является предоставление инновационных и современных решений для рынка встраиваемых систем. Eurotech предлагает широкий спектр готовых защищённых переносных компьютеров и решений для таких отраслей, как транспорт, авиация, оборона, медицина и безопасность. Номенклатура продукции включает встраиваемые компьютеры, защищённые маршрутизаторы, панельные компьютеры, устройства обработки видеоданных и контроля пассажиропотока, а также компактные переносные компьютеры.

Заключение дистрибьюторского соглашения между ПРОСОФТ и Eurotech

Одним из важных аспектов развития Интернета является то, что интеллектуальные устройства могут эффективно связываться друг с другом и с приложениями, находящимися в центрах обработки данных. Eurotech разрабатывает и предлагает программно-аппаратные стандартные платформы, позволяющие эффективно подключать полевые устройства контроля данных к IT-инфраструктуре. Решения от Eurotech – это комбинация аппаратных платформ, встроенного микропрограммного обеспечения, операционных систем, программных платформ и внешней инфраструктуры, позволяющая заказчикам сосредоточиться только на их ключевой компетенции. Теперь любой заказчик, от отдельного разработчика до крупного системного интегратора, имеет возможность сократить время выхода на рынок и поставлять масштабируемые, экономически выгодные продукты на основе продукции Eurotech.

Официальный статус дистрибьютора Eurotech на территории России и стран СНГ позволяет компании ПРОСОФТ предлагать заказчикам высоконадёжные промышленные компьютеры и встраиваемые системы для ответственных применений от известного производителя. ●

4. В представлении Project Explorer делаем щелчок правой кнопкой мыши и в контекстном меню выбираем New/VxWorks Image Project. Открывается мастер создания проекта.
5. В окне мастера вводим имя проекта (vx67-xm50), нажимаем Next. Мастер переходит к стадии выбора платформы.
6. В окне мастера в группе Setup the project:
   - в поле Based on выбираем a board support package;
   - в поле BSP выбираем men_em9 (это только что установленный BSP для XM50);
   - в поле Tool chain выбираем diab.
7. Нажимаем Finish. Проект успешно создаётся и появляется в представлении Project Explorer.
8. В представлении Project Explorer делаем щелчок правой кнопкой мыши на новом проекте vx67-xm50, в контекстном меню выбираем "Build Project". Начинается процесс построения проекта.

Ожидаемый результат

- Проект собирается успешно.
- В инструментальной консоли (Build Console) ошибок нет.

Наблюдаемый результат

- Построение проекта завершается с ошибкой, проект vx67-xm50 в представлении Project Explorer помечается красным крестиком.
- В инструментальной консоли (Build Console) в журнале событий появляется строка: “make[1]: *** No rule to make target `c:/men/VXWORKS/LIB/MEN/vxworks-6.7/libPPC85XXdiab/menMtdRawLib.a', needed by `partialImage.o'. Stop”.
- Бинарный образ с именем vxWorks в каталоге $(WIND_BASE)\workspace\vx67-xm50\default\ не создаётся.

Обратите внимание: описание проблемы представляет собой готовую инструкцию, как получить такую же проблему с нуля. Может показаться, что в нём чересчур много деталей, но на самом деле это не так: конфигурация инструментального компьютера важна, так как среда разработки Wind River Workbench основана на Eclipse и имеет определённые притязания по части ресурсов; детализация шагов важна, поскольку сходные по названию пункты меню могут выполнять аналогичные, но не одинаковые операции, и т.д. Опять же, одна и та же задача может решаться различными способами, и один из них может приводить к проблеме, а другой – нет. В конечном итоге недостаточная детализация всегда оборачивается встречными вопросами, когда проблему не удаётся воспроизвести.

В данном случае проблему удалось воспроизвести с первого раза; в процессе решения также помогла трансляция рабочего стола утилитой Mikogo (офисы технической поддержки MEN и Wind River находятся в Германии, с поясным сдвигом относительно Москвы в 2 часа). В результате совместной работы двух групп технической поддержки проблема была решена за 4 дня; если хотите знать, в чём было дело, напишите мне. Скажу лишь, что всё оказалось очень просто (хотя и не так просто, как может показаться на первый взгляд).

Кстати, о простоте.

Возможно, всё ещё проще

Чёрт, как известно, кроется в деталях. Технические системы изобилуют деталями, и это превращает техническую поддержку в сложный и кропотливый процесс; сложность в нём, к сожалению, неизбежна, так как она обеспечивает фундамент для предсказуемого решения сложных проблем. Чтобы извлечь из процесса технической поддержки максимум пользы, важно знать, как он устроен, как работает и что ему требуется, чтобы работать эффективнее.

Недавно мне довелось помогать заказчику с решением одной неочевидной проблемы с драйверами устройств для Linux. На тот момент у меня уже был на руках черновик этой статьи, и я отправил его заказчику, порекомендовав составить запрос аналогичным образом. Заказчик составил запрос, который был переправлен производителю; ответное письмо инженера технической поддержки начиналось словами: «Большое спасибо за подробное описание. Если бы все заказчики составляли запросы таким образом!». Далее в письме приводилось решение, устраняющее проблему в одно действие.

Конечно, не все проблемы сложны, но их сложность никогда нельзя оценить заранее. Иногда проблема оказывается настолько простой по сути или настолько часто встречающейся, что для её идентификации достаточно нескольких ключевых слов – это зачастую и заставляет надеяться, что возникшая проблема как раз из таких и что тратить время на подробное описание не придётся. Разумный компромисс в таких случаях – сперва позвонить и спросить инженеров техподдержки, знакома ли им такая проблема. Всегда есть вероятность, что ответом будет «да», – тогда вы сэкономите немного времени; если же ответом будет «нет», всегда остаётся описанный в статье проверенный путь.

Избегать ошибок никогда не получается, поэтому выигрышная стратегия – не искать безошибочности, а уметь быстро находить и исправлять ошибки и извлекать из них опыт. Техническая поддержка создана, чтобы помогать вам в этом; однако, чтобы получить от неё максимальную отдачу, ей тоже надо помогать. Как это делать, вы теперь знаете, так что, как говорят англоязычные коллеги, shoot yourself in the foot [9]. Чудес обещать не могу, но через некоторое время вы обязательно увидите, что работать стало значительно проще, а количество осязаемых результатов в единицу времени существенно возросло.●

Литература

1. A Story About 'Magic' [Электронный ресурс]. – Режим доступа : http://catb.org/jargon/html/magic-story.html
2. Блог Алиева Рауфа. О жизни и о себе [Электронный ресурс]. – Режим доступа : http://rauf.livejournal.com/23552.html
3. Malianov's journal [Электронный ресурс]. – Режим доступа : http://malianov.livejournal.com/52991.html
4. Заповедник доброкачественных эмоций [Электронный ресурс]. – Режим доступа : http://oleg-chakrits.livejournal.com/1123822.html
5. Джоэл Спольски. Джоэл о программировании. – М. : Символ-Плюс, 2006. – ISBN 5-93286-063-4, 1-59059-389-8.
6. Dilbert by Scott Adams [Электронный ресурс]. – Режим доступа : http://dilbert.com/strips/comic/1999-08-04/
7. Jargon File [Электронный ресурс] // Wikipedia. – Режим доступа : http://en.wikipedia.org/wiki/Jargon_File
8. Эдвард Йордон. Путь камикадзе. Как разработчику программного обеспечения выжить в безнадёжном проекте. – М. : Лори, 2003. – ISBN 0-13-748310-4, 5-85582-085-8.
9. EnglishClub [Электронный ресурс]. – Режим доступа : http://www.englishclub.com/ref/esl/Idioms/S/shoot_yourself_in_the_foot_231.htm

Автор – сотрудник фирмы ПРОСОФТ
Телефон: (495) 234-0636
E-mail: info@prosoft.ru

НОВОСТИ НОВОСТИ НОВОСТИ НОВОСТИ НОВОСТИ НОВОСТИ

Новости ISA

С 1 января 2013 года к выполнению обязанностей президента Российской секции ISA приступил декан факультета радиоэлектроники, электроники и связи ГУАП доцент Александр Роальдович Бестугин. Он сменил на этом посту проректора ГУАП профессора Виктора Матвеевича Боера, который награждён почётным дипломом ISA.

Делегация Российской секции ISA приняла участие в работе ежегодного собрания ISA, в заседании исполкома ISA округа 12 и программе Automation Week в городе Орландо (США).

Началась подготовка к XIV Международному форуму «Формирование современного информационного общества: проблемы, перспективы, инновационные подходы». На форуме будет проведён круглый стол по международному сотрудничеству в рамках Международного общества автоматизации (ISA). Объявлена тема, которая будет обсуждаться на круглом столе: «Тенденции в глобальной автоматизации до 2020 года». Форум пройдёт с 3 по 7 июня 2013 года в Санкт-Петербурге. Сопредседателями круглого стола ISA будут профессор Gerald Cockrell – президент ISA 2009 года (США), Pino Zani – президент ISA 2002 года (Италия), Nelson Ninin – президент ISA 2010(Бразилия), профессор Анатолий Оводенко – глава представительства ISA в Российской Федерации.

Участники заседания исполкома ISA округа 12 в Орландо

15 ноября 2012 года состоялось первое занятие Интернет-семинара по управлению проектами (Practical project management: learning to manage the professional), который в очередной раз провёл для студентов, аспирантов и преподавателей ГУАП профессор университета штата Индиана (ISU) Gerald Cockrell. В октябре 2012 года 16 человек, успешно прослушавших семинар в 2011/2012 году, получили сертификаты ISU.

4 ноября 2012 года исполнилось 10 лет со дня официального открытия офиса представительства ISA в Российской Федерации. На имя Главы представительства ISA в РФ профессора Анатолия Аркадьевича Оводенко поступили поздравления от президента ISA 2002 года господина Pino Zani, президента ISA 2009 года профессора Gerald Cockrell, президента ISA 2010 года господина Nelson Ninin, члена исполкома ISA господина Billy Walsh. ●