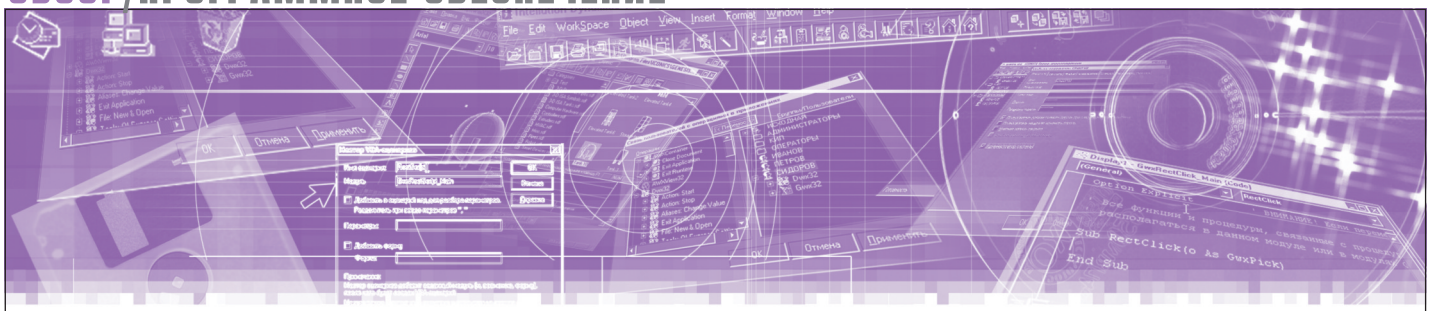

Николай Горбунов

Безопасность и сертификация программного обеспечения

Часть 1. Трудности перевода

В статье приводится обзор современной терминологической и нормативно-технической базы функциональной и информационной безопасности ПО, затрагивается ряд основополагающих вопросов качества ПО и их привязки к нормативной базе. Рассматриваются примеры программных продуктов, соответствующих современным требованиям сертификации, и практические подходы к подтверждению соответствия. В первой части речь идёт об исторически сложившихся терминологических разногласиях.

Объём кода ПО встраиваемых систем (в том числе критичных) в последние десятилетия растёт взрывными темпами, и количество разговоров о том, что ПО должно быть безопасным, растёт вместе с ним. Все с этим согласны; однако вокруг безопасности ПО традиционно существует терминологическая путаница, порождающая массу споров, зачастую столь же бурных, сколь беспочвенных. Поэтому перед тем как углубляться в детали и переходить к практике, имеет смысл потратить немного времени на то, чтобы определить понятия и конкретизировать предмет разговора. Итак, что же на самом деле понимается под безопасностью ПО? Чтобы ответить на этот вопрос, нужно сначала разобраться с рядом распространённых терминологических неоднозначностей.

Безопасность или защищённость?

Первая неоднозначность берёт начало из словарей общей лексики. Дело в том, что в английском языке есть два неоднокоренных слова, означающих безопасность, – ”safety“ и ”security”. Англо-русские словари общей лексики трактуют эти слова как синонимы, в результате в переводных источниках термином «безопасность» периодически обозначается то одно, то другое, в зависимости от предпочтений переводчика. Однако в технической лексике понятия ”safety” и ”security” являются чётко определёнными терминами, обозначающими в корне разные вещи, и чтобы сохранить смысловое различие этих терминов (а значит, и предотвратить искажение смысла при переводе), их следует переводить по-разному. В настоящей статье используется терминология из официальных переводов стандартов МЭК/ИСО:

- **функциональная безопасность** (safety) – отсутствие недопустимых рисков, зависящее от правильности функционирования системы, систем обеспечения безопасности и внешних средств уменьшения риска (ГОСТ Р МЭК 61508-4-2007);
- **информационная безопасность** (security) – защита конфиденциальности, целостности, доступности, а также неотказуемости, подотчётности, аутентичности и достоверности информации (ГОСТ Р ИСО/МЭК 27002-2012).

Иными словами, обеспечение функциональной безопасности включает в себя комплекс мероприятий по предотвращению неприемлемых последствий функционирования системы (гибель людей, ущерб окружающей среде и т.п.), в то время как обеспечение информационной безопасности сводится к тому, чтобы обрабатываемая системой информация постоянно была правильной, полной, была доступна непрерывно и только авторизованным объектам/субъектам, и все действия с ней можно было бы проследить при наличии соответствующих прав.

Поскольку область действия понятия «информационная безопасность» включает в себя защиту от несанкционированного доступа (НСД), в качестве синонима информационной безопасности в русскоязычной терминологии также часто употребляется термин «защищённость». Это не совсем корректно (так как защита от НСД является только одним из аспектов информационной безопасности), но удобно с точки зрения предотвращения путаницы, поскольку позволяет переводить неоднокоренные слова “safety” и “security” также неоднокоренными словами «безопасность» и «защищённость» соответственно.

Безопасность или надёжность?

Вторая неоднозначность на порядок сложнее и уходит корнями в техническую культуру XX века. Дело в том, что комплексный подход к оценке качества технических систем с позиции анализа рисков (а именно на анализе рисков

строятся базовые понятия функциональной безопасности) появился не так давно. При этом традиционным проверенным аппаратом оценки качества всегда была классическая теория надёжности, и это даёт соблазн распространить привычные методики на индустрию ПО. Однако на поверку оказывается, что не всё так просто.

Согласно ГОСТ Р 27.002-2009, *надёжность* сама по себе не является количественным показателем и определяется *готовностью*, которая, в свою очередь, зависит от *безотказности* и *ремонтопригодности*:

- **Надёжность** (dependability) – свойство готовности и влияющие на него свойства безотказности и ремонтопригодности и поддержка технического обслуживания.
  - **Готовность** (availability) – способность изделия выполнить требуемую функцию при данных условиях в предположении, что необходимые внешние ресурсы обеспечены.
  - **Безотказность** (reliability) – способность изделия выполнить требуемую функцию в заданном интервале времени при данных условиях.
  - **Ремонтопригодность** (maintainability) – способность изделия при данных условиях использования и технического обслуживания к поддержанию или восстановлению состояния, в котором оно может выполнить требуемую функцию (ГОСТ Р 27.002-2009).

Выражение «при данных условиях» в определении готовности подчёркнуто не зря – здесь таится один важный подводный камень. Классическая теория надёжности относится к отказу как к случайному событию и поэтому использует математический аппарат теории вероятностей. Однако, кроме случайных отказов, бывают ещё отказы *систематические*, то есть гарантированно повторяющиеся при определённой (зачастую очень сложной и поэтому редкой) комбинации условий (см. тот же ГОСТ Р 27.002-2009). Такие отказы нельзя описывать количественными статистическими показателями (по крайней мере, пока Нассим Талеб [1] не добрался до теории надёжности), так как при одних условиях они не возникают вообще, а при других возникают всегда. Таким образом, при внесении в рассмотрение систематических отказов показатели надёжности перестают быть однозначными, и к статистическому анализу добавляется необходимость тестирования в различных условиях.

Хорошим примером систематического отказа является авиационное происшествие с рейсом № 38 компании British Airways 17 января 2008 года, когда абсолютно исправный Boeing 777, успешно преодолев дистанцию в 8100 км между Пекином и Лондоном, при заходе на полосу аэропорта Хитроу внезапно потерял тягу обоих двигателей и совершил аварийную посадку в 270 метрах от взлётно-посадочной полосы. Расследование показало, что к происшествию привела сложная причинно-следственная цепочка: сначала длительный крейсерский полёт на большой высоте над холодной территорией (температура топлива упала до –30°C, и в нём образовалась взвесь ледяных кристаллов), потом резкое снижение (топливо «прогрелось» до –20°C, адгезивность ледяной взвеси резко возросла, и лёд нарос на стенках топливопроводов), а затем – попадание в турбулентный поток (автомат тяги резко увеличил подачу топлива, и перепад давления в топливопроводах единовременно смыл лёд со стенок, ледяная пробка закупорила топливо-масляный теплообменник, и двигатели лишились топлива на высоте 200 метров над землёй). Как описать такое в терминах теории вероятностей?

Ситуация усугубляется тем, что классический метод борьбы со случайными отказами – резервирование – от систематических отказов не только не спасает, но и ухудшает ситуацию, приводя к так называемым *отказам по общей причине* (common cause failures). Такие отказы проявляются во всей резервированной схеме одновременно, потому что если нагруженные и резервные модули реализованы одинаково, они будут подвержены действию одних и тех же событий-инициаторов – trigger events. (В описанном случае с рейсом № 38 резервирование топливо-масляного теплообменника, очевидно, не спасло бы ситуацию.) Вероятность таких отказов можно снизить, реализуя разные модули резервированных схем разными коллективами разработчиков и на основе различных принципов, методов и технологий, чтобы исключить совпадение событий-инициаторов. Это, впрочем, не отменяет необходимости тестирования.

Пример с системным отказом оборудования приведён здесь не случайно – ситуация с «надёжностью» ПО представляет собой ещё более полярный случай. Дело в том, что ПО – это *алгоритм*, а в работе алгоритма вообще не бывает случайных событий, он детерминирован. Поэтому использование термина «надёжность» применительно к ПО (например, ГОСТ 28806-90 хоть и признаёт особую природу отказов ПО, но всё равно применяет к нему термин «надёжность») зачастую только вносит путаницу и создаёт соблазн спросить про численные показатели, которые к характерным для ПО систематическим отказам неприменимы.

Качество алгоритма определяется детальностью его проработки, а значит, степенью понимания исходной задачи и корректностью донесения этого понимания до реализации. Процессы/потоки и примитивы синхронизации в многозадачной операционной системе (ОС) не меняют своё состояние под воздействием случайных факторов, значения переменных не изменяются сами собой (при условии исправности аппаратуры, разумеется) и т.д. Как результат вероятностный анализ может быть применим только к аппаратуре, на которой выполняется ПО, но не к самому ПО. *Все отказы ПО являются систематическими*, и для борьбы с ними годятся только системные средства; множество хороших иллюстраций к этому утверждению приводится в статье Нэнси Левесон «О роли ПО в катастрофах космических аппаратов» [2], опубликованной в 2004 году. Статья недвусмысленно демонстрирует, как недостатки в системной *культуре безопасности* привели к критическим отказам программного обеспечения и, как следствие, провалу пяти космических миссий NASA/ESA в период с 1996 по 1999 годы.

Но если не «надёжность», то что тогда?

В последнее время в англоязычной литературе часто употребляется словосочетание «функционально безопасное ПО» (safe software). При этом в ответ на прямой вопрос эксперты обычно признают, что этот термин не совсем корректен, поскольку функционирование программных компонентов явного риска в себе не несёт, и поэтому они не могут быть опасными или безопасными. Однако, будучи *некорректно реализованным* или *некорректно применённым*, программный компонент (как и любой другой) может стать причиной (или звеном в цепочке причин) опасного отказа системы с соответствующими последствиями.

Это важный момент, так как он отвечает на вопрос, почему «функционально безопасное» ПО не всегда является сертифицированным. Функциональная безопасность определена только для объектов, функционирование которых несёт в себе риск, а риск обретает конкретные

формы только на уровне либо системы в целом (например, химического производства или летательного аппарата), либо законченного функционального блока (например, двигателя). Поэтому *сертифицируется* либо вся система, либо законченный функциональный блок (и то последняя практика существует только в авиации), а компоненты (включая ПО) *подготавливаются к сертификации*, то есть разрабатываются в соответствии с нормативными требованиями, позволяющими применять их в системах с заданным уровнем функциональной безопасности, и снабжаются соответствующим пакетом подтверждающих документов (так называемым сертификационным пакетом – certification evidence).

Последнее утверждение может вызвать справедливый скепсис – множество современных производителей оборудования и ПО заявляют свои продукты как безопасные и в подтверждение этому демонстрируют официальные сертификаты (скажем, по МЭК 61508 SIL 4). Казалось бы, собрал систему из компонентов, имеющих сертификат, и вуаля. Однако здесь есть нюанс: уровень безопасности, по которому сертифицируется компонент, определяется по формальной методике для «сферического компонента в вакууме», то есть физическому смыслу интегрального уровня безопасности (SIL – Safety Integrity Level) на самом деле не соответствует. Это как со средним временем наработки между отказами (MTBF) – знание его значений для каждого из компонентов ещё ничего не гарантирует, так как конечные характеристики результирующей системы будут определяться тем, как именно она из этих компонентов будет составлена. Поэтому сам по себе сертификат говорит только о факте *соблюдения производителем методологии*, предусмотренной стандартом, вопросы же интеграции раскрываются в прилагаемом к сертификату «Руководстве по безопасности» (Safety Manual), описывающем, как именно этот компонент обязан интегрироваться в систему, чтобы она оставалась безопасной.

Чтобы подчеркнуть разницу между компонентами, *имеющими сертификат* (это, повторюсь, допускается не любым стандартом), и компонентами, *пригодными для сертификации*, для их обозначения используют разные термины – «*сертифицированный*» (certified) и «*сертифицируемый*» (certifiable) соответственно. (Последнее, кстати, является подмножеством первого, так как чтобы сертифицировать компонент, сертификационный пакет к нему разрабатывать в любом случае придётся.) В целом вопрос о том, имеет ли смысл сертификат безопасности, выданный на компонент (как аппаратный, так и программный) вне контекста системы, до сих пор является предметом «священных войн»: с одной стороны, использование сертифицированных компонентов призвано сократить стоимость сертификации всей системы, с другой, – как можно учесть все возможные сценарии интеграции в «Руководстве по безопасности», не совсем понятно.

Как бы там ни было, с сертификатом или без, сам по себе подход к качеству ПО, с точки зрения аппарата функциональной безопасности позволяет решить сразу две проблемы. Во-первых, проведение анализа рисков (см., например, РД 03-418-01 «Методические указания по проведению анализа риска опасных производственных объектов») означает моделирование всех возможных причинно-следственных связей, способных привести к отказу. В результате фактически получаются «антитребования» – в отличие от требований, описывающих, что и как система *должна* делать, анализ рисков даёт информацию о том, что и как она делать *не должна* и что произойдёт в противном случае. Будучи детализированными до уровня компонентов системы, эти «антитребования» дают необходимую основу для проектирования и тестирования. На более поздних этапах жизненного цикла могут выявляться дополнительные нюансы, не учтённые в изначальном анализе рисков, но при корректно поставленном процессе разработки это даёт необходимую обратную связь, и качество компонентов со временем повышается.

Во-вторых, анализ рисков автоматически даёт ранжирование возможных отказов по степени критичности, а значит, и адекватно распределяет усилия по системному противодействию. Очевидно, что избежать всех возможных отказов нельзя, а избыточные меры контроля сделают процесс разработки нерентабельным; ранжирование отказов позволяет сосредоточить усилия на обеспечении корректности наиболее критичных программных модулей и тем самым соблюсти баланс между корректностью программного кода и операционными затратами.

Разумеется, чтобы всё это работало, необходима соответствующая инфраструктура. Необходимо, чтобы требования были задокументированы, доступны и проверяемы, а неизбежные изменения в них отрабатывались корректно. Необходимо, чтобы требования корректно и своевременно отображались в программный код и соответствующие тестовые сценарии. Все проектные документы (не только код) должны архивироваться в системе управления версиями, чтобы ничего не потерялось и всегда можно было обратиться к любой версии любого документа. В процессе кодирования необходимо соблюдать «правила гигиены», помогающие сделать код тестируемым, сопровождаемым и избежать скрытых ошибок (например, связанных с интерпретацией синтаксиса языка или использованием спорных техник программирования). Всему этому необходимо обучать персонал. И так далее. Всё это входит в понятие «культура безопасности» и реализует комплексный подход к обеспечению качества ПО; конкретные требования могут отличаться в зависимости от целевой индустрии и приводятся в соответствующей отраслевой нормативной базе (об этом см. далее).

Некоторые производители ПО для безопасных применений идут ещё дальше и добавляют в свои продукты дополнительные средства обеспечения отказоустойчивости (например, подсистемы Error Detection & Reporting в ОС VxWorks и High Availability Framework в ОС QNX Neutrino), что позволяет дополнительно упростить реализацию функций безопасности, в частности, обнаружение и обработку отказов.

Подробнейший терминологический и методологический разбор темы функциональной безопасности приводится в [3], к сожалению, правда, в данной (прекрасной, к слову) работе теме ПО уделено непропорционально мало внимания.

Одно не может без другого

Инцидент с иранской ядерной программой и «червём» Stuxnet наглядно продемонстрировал, что проблемы функциональной и информационной безопасности нельзя рассматривать в изоляции друг от друга, так как в современном мире тесно связанных информационных систем брешь в информационной безопасности легко способна привести к нарушению безопасности функциональной. Например, в случае с тем же Stuxnet уязвимость в системе информационной безопасности завода по обогащению урана привела к тому, что вредоносный код проник в цеховой контроллер АСУ ТП, а оттуда – в программируемый логический контроллер (ПЛК), управлявший цен-

трифугами для разделения изотопов. В результате действий вредоносного кода центрифуги были выведены из безопасного режима вращения и физически разрушены, то есть несоблюдение режима информационной безопасности поставило под угрозу функциональную безопасность ядерного объекта.

Основная сложность здесь заключается в том, что интересы функциональной и информационной безопасности зачастую не просто не совпадают, но напрямую конфликтуют. Анекдотический пример такого конфликта приводится в статье [4] «Разрешение противоречий между требованиями функциональной и информационной безопасности в киберфизических системах» – суть истории в том, что однажды аналитики некоего крупного европейского производителя автомобилей представительского класса обнаружили, что одна из новых моделей непропорционально часто становится объектом угона. При допросе одного из задержанных угонщиков выяснилось, что данную модель смехотворно легко вскрыть – достаточно забраться ей на крышу и подпрыгнуть, и двери разблокируются автоматически. Анализ показал, что требования функциональной безопасности предписывали облегчить покидание салона при переворачивании автомобиля на крышу. Для этого была добавлена функция автоматической разблокировки дверей при переворачивании; событие же переворачивания фиксировалось по факту увеличения давления на крышу (силу и продолжительность давления, очевидно, учесть забыли). Увеличив таким образом безопасность транспортного средства, производитель сделал его уязвимым для несанкционированного доступа.

Единого решения у данной проблемы пока нет (очевидно, по причине её «молодости»), ситуация усугубляется ростом применения многоядерных процессоров, несущих богатый потенциал для снижения стоимости сертификации, но пока не подкреплённых методиками оценки безопасности, как функциональной, так и информационной. Хороший обзор на эту тему даёт [5], в настоящей статье данный аспект не рассматривается.

В следующей части статьи речь пойдёт о зарубежной и отечественной нормативно-технической базе функциональной и информационной безопасности ПО. ●

Литература

1. Талеб Николас Нассим. – Чёрный лебедь: под знаком непредсказуемости. – М. : КолЛибри, 2009.
2. Leveson Nancy G. The Role of Software in Spacecraft Accidents [Электронный ресурс] // Journal of Spacecraft and Rockets. – 2004. – № 41. – Режим доступа: http://sunnyday.mit.edu/papers/jsr.pdf.
3. Фёдоров Ю.Н. Справочник инженера по АСУ ТП: проектирование и разработка : Уч.-практ. пособие. – М.: Инфра-Инженерия, 2008.
4. Mu Sun, Sibin Mohan, Lui Sha, Carl Gunter. Addressing Safety and Security Contradictions in Cyber-Physical Systems [Электронный ресурс] // Режим доступа: http://www.techrepublic.com/resource-library/whitepapers/addressing-safety-and-security-contradictions-in-cyber-physical-systems/
5. Паркинсон Пол. Многоядерные вычислительные среды и безопасность ПО // Современная электроника. – 2013. – № 8, 9.

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