Тестирование на основе моделей

В. В. Кулямин

Лекция 1. Качество программного обеспечения и методы его контроля

Данный курс лекций посвящен тестированию программного обеспечения (ПО).

Многим, особенно людям, считающим себя хорошими программистами, но не имеющим опыта работы над большими проектами с жесткими сроками готовности и требованиями по качеству и надежности разработанного ПО, может показаться, что тестирование не представляет особых проблем, по сравнению с созданием программ — сиди себе, да пробуй, что работает, а что — нет. К большому сожалению, это не так. Тесты для сложной системы сами по себе сложны. Кроме того, для сложных систем требуется очень много тестов, чтобы проверить корректность работы реализуемых ими функций во многих различных ситуациях, поэтому наборы тестов тоже представляют собой сложные программные комплексы, требующие особых методов для их разработки и сопровождения, и, как это не покажется странным, применения творческих способностей от людей, создающих их.

В этом курс рассказывается о специфических методах построения тестов, которые в последнее время обычно выделяются в особую область — *тестирование на основе моделей*. К значению этого термина мы вернемся несколько позже, а сейчас отметим, что эта область находится на границе между *теоретической информатикой (computer science)* и *инженерией программных систем (software engineering)*. Поэтому она требует как определенной математической подготовки, знакомства с техниками строгого описания свойств программного обеспечения, так и умения применять их на практике и оценивать практическую эффективность используемых подходов на пути к приемлемому компромиссу между полнотой тестирования и затраченными на него ресурсами.

Тестирование само по себе предназначено для проверки правильности или надежности создаваемых программных систем, точнее, для контроля их *качества*. Поскольку понятие качества программной системы не является вполне очевидным, несмотря на его интуитивную ясность, начнем с его более детального рассмотрения.

Качество программного обеспечения

Можно определить качество ПО, как его пригодность и удобство для решения тех задач, для которых оно разрабатывалось. Так же можно охарактеризовать и качество любого инструмента, предназначенного для чего-либо.

Однако у программных систем есть две особенности, отличающие их, если не от всех, то от многих других инструментов, используемых человеком в своей деятельности.

- Во-первых, цели создания программной системы чаще всего сложны и включают множество аспектов. Программное обеспечение, за редким исключением, не разрабатывается для решения ровно одной задачи. Гораздо чаще это целый набор связанных задач из некоторой области, включающий, кроме этого еще и экономическую эффективность использования данной системы и удобство работы с ней для того персонала, который имеется у организации-заказчика.
- Во-вторых, очень часто программы используются для решения несколько не тех задач, для которых они предназначались при создании. Набор целей, для достижения которых применяется данное ПО, изменяется со временем, что отражается и в постоянном добавлении новых возможностей и функций в новые версии программ.

Поэтому целостного и четкого понятия качества ПО не существует. Вместо этого используют различные *модели качества*, систематизирующие набор аспектов и метрик качества, рассмотрение которых необходимо для адекватной оценки качества разнообразных

программ. Такие модели могут изменяться со временем, поскольку изменяются потребности индустрии производства ПО, появляются новые группы возможностей или внимание разработчиков привлекается к новым аспектам качества, ранее считавшимся несущественными.

Наиболее широко на данный момент используется модель качества ПО, зафиксированная в наборе стандартов ISO 9126 [1-4]. В несколько упрощенном виде эта модель определяет 6 основных характеристик качества программного обеспечения. Каждая характеристика уточняется при помощи некоторого набора атрибутов.

Рисунок 1. Характеристики и атрибуты качества ПО по ISO 9126.

- *Функциональность*.
  Эта характеристика обозначает способность ПО решать определенный круг задач.. Это качественная характеристика, определяющая, что именно делает данная программа. Атрибутами функциональности являются, например, точность производимых вычислений и защищенность ПО — способность предотвращать доступ к его функциям тем людям или другим системам, у которых нет прав на это.
- *Надежность.*
  Это способность ПО поддерживать определенный уровень работоспособности в заданных условиях.
  Надежность является вероятностной характеристикой работоспособности ПО. Атрибутами ее являются, например, вероятность работы без ошибок на заданном интервале времени и среднее время восстановления после сбоя.
- *Удобство использования.*
  Удобство использования показывает, насколько ПО привлекательно, удобно в обучении работе с ним и при выполнении самой работы.
  К атрибутам удобства использования относятся, например, среднее время обучения пользователя выполнению определенных задач с помощью данной системы, а также усилия, затрачиваемые на решение фиксированной задачи.
- *Производительность.*
  Это способность ПО обеспечивать необходимую работоспособность по отношению к выделяемым для этого ресурсам. В соответствии с разными видами ресурсов — временем, объемом памяти, пропускной способностью сетевых соединений — выделяются и различные атрибуты производительности.

- *Переносимость.*
  Эта характеристика показывает сохранение работоспособности ПО при изменении его окружения.
  Ее атрибутами являются, например, возможность развертывания или установки ПО в различных окружениях и его адаптируемость — способность приспосабливаться к работе в различных окружениях при помощи действий, зафиксированных в документации.
- *Удобство сопровождения.*
  Удобство сопровождения определяет трудоемкость анализа, исправления ошибок и внесения изменений в ПО.
  Его атрибутами являются, в частности, удобство проведения тестирования, удобство внесения изменений и риск возникновения неожиданных эффектов при изменениях.

Требования к программному обеспечению

Перечисленные характеристики качества ПО позволяют систематическим образом определять *требования* к данной программной системе, последовательно рассматривая различные ее аспекты. Требования как раз и говорят о том, какие конкретные свойства и характеристики данная система должна иметь, чтобы ею можно было пользоваться для решения тех задач, для которых она предназначена.

Требования определяют, какое поведение системы является правильным и желаемым, а какое должно рассматриваться как некорректное. Поэтому требования играют первостепенную роль при тестировании — именно они и проверяются с его помощью.

Сами требования определяются при анализе задач, стоящих перед рассматриваемой программной системой и ее окружения. Они могут извлекаться из различных источников.

- Требования обычно фиксируются в документе, создаваемом при решении о разработке системы и называемом *техническим заданием* (а по-английски — *requirements specification, спецификация требований*). Иногда этот документ создается представителями заказчика, но часто его приходится писать самим разработчикам на основе информации, полученной из других источников.
- Важным источником требований являются *стандарты*, регламентирующие функции и состав систем, работающих в определенной предметной области. Такие стандарты создаются на основе опыта, накопленного в большом количестве проектов, в ходе которых такие системы создавались или дорабатывались, и поэтому содержат много ценной информации о свойствах и ограничениях на работу таких систем.
  Часто работа системы связана с выполнением каких-то действий, регулируемых существующим законодательством и нормами, действующими в данной области. Нормы и законы тоже являются разновидностью действующих стандартов, хотя обычно они формулируются менее четко и однозначно, чем технические стандарты и правила.
- Разработчики сами могут осознать, что что-то должно быть сделано определенным образом — так обычно возникают *внутренние ограничения задач*, не соблюдая которые, невозможно решить их правильно. Чаще всего ограничения такого рода можно усмотреть при детальном анализе решаемой задачи.
- Иногда ряд требований к новой системе можно сформулировать на основе анализа *уже существующих систем* для решения схожих задач.
- Б*о*льшая часть требований формулируется на основе явно высказываемых *пожеланий* пользователей системы, их руководителей, заказчиков ее разработки и других заинтересованных лиц.
- Наконец, наиболее нечетким, но, тем не менее, достаточно важным источником требований являются невысказанные явно *потребности и нужды* пользователей создаваемой системы, которые, несмотря на это, все же часто поддаются анализу.

При извлечении требований из перечисленных выше источников их четкость и согласованность возрастают при движении от потребностей и пожеланий к стандартам и техническому заданию.

Тестирование проверяет соответствие требованиям, и поэтому чем более точно и ясно они сформулированы, тем аккуратнее и полнее можно провести тестирование. Если какие-то требования определены нечетко, проверка их тоже может быть выполнена лишь с определенной степенью точностью, и системы, ведущие себя сильно по-разному, могут быть признаны удовлетворяющими ему в одинаковой мере. Получаемые при этом оценки качества таких систем будут во многом субъективными.

В некоторых случаях тестирование кажется возможным и в отсутствии всяких требований, по крайней мере, документально зафиксированных. В этих случаях, однако, оно использует какие-то свойства, которые считаются само собой разумеющимися и, соответственно, представляющие собой неявные требования. Пример такого свойства — отсутствие сбоев при работе программы. Если такие неявные требования действительно должны быть выполнены, можно проводить тестирование на их основе. Если же это не так, то есть иногда сбой может рассматриваться как корректное поведение системы (если, скажем, вводятся совсем некорректные исходные данные), то подобное тестирование может привести к неправильным выводам о наличии ошибок в системе.

Чтобы требования к программному обеспечению можно было уверенно использовать при его разработке и тестировании, они должны обладать рядом характеристик, которые зафиксированы в стандартах, регламентирующих разработку программного обеспечения.

Два таких стандарта — IEEE 830 [5] и IEEE 1233 [6] — определяют следующие характеристики правильно составленных требований к ПО.

- *Адекватность*, соответствие реальным потребностям пользователей ПО.
- *Однозначность*, отсутствие двусмысленностей и возможностей разного толкования.
- *Полнота* — отражение в требованиях всех существенных потребностей и всех ситуаций, в которых система должна будет функционировать.
- *Непротиворечивость* или согласованность между разными элементами требований.
- *Систематичность* представления — требования должны быть описаны в рамках некоторой системы с четким указанием места каждого требования среди остальных, с определением связей и зависимостей между ними и приоритетности для различных заинтересованных лиц.
- *Прослеживаемость* — требования должны иметь четко определенные связи с модулями разрабатываемой системы, частями проектной документации и тестами, чтобы всегда можно было определить, для выполнения или проверки каких требований создан каждый из этих элементов и насколько он им соответствует.
- *Проверяемость* или возможность для каждого требования однозначно установить при помощи того или иного анализа системы, выполнено это требование или нет.
- *Модифицируемость* или возможность внесения изменений в набор требований с быстрым исправлением всех возникающих при этом дефектов с точки зрения других характеристик.

В ходе работы над создаваемой программной системой или переработки уже существующей всегда, в том или ином виде, проводится *анализ требований*, цель которого — подготовить представление требований к ПО, имеющее все указанные характеристики. Анализ требований обычно включает следующие виды деятельности.

- ***Выделение требований***. Его задача — определить полный список требований, уточнить недостаточно четко сформулированные и определить возможные компромиссы в тех случаях, когда различные заинтересованные лица высказывают противоречащие друг другу требования.

Выделение требований включает определение доступных источников требований, извлечение требований из них, в ходе чего, собственно, и фиксируются отдельные требования, и согласование требований, полученных от разных источников, при возникновении необходимости в этом.

При извлечении требований может применяться широкий диапазон различных техник — от простого анализа задач, анализа имеющихся документов, до проведения интервью, опросов и семинаров, с использованием специальных методов для фиксации как высказываемых пожеланий и формулировок, так и эмоционального состояния опрашиваемых, чтобы в дальнейшем оценить правдивость и полноту предоставленных сведений.

- ***Систематизация и описание требований***, их сведение в единую систему и составление представляющих ее моделей, отражающих различные аспекты собранных требований. При систематизации особое внимание уделяется полному отражению всех извлеченных сведений в требованиях, определению связей и зависимостей между требованиями, идентификации требований, необходимой, чтобы иметь возможность ссылаться на них из различных проектных документов.
- ***Валидация и верификация требований.***
  Задача этих видов деятельности — проверка необходимых свойств требований к ПО. Валидация представляет собой проверку адекватности и полноты требований, то есть проверяет, что зафиксированные в требованиях ограничения и пожелания действительно представляют потребности пользователей, заказчиков и других заинтересованных лиц, а также, что все их существенные потребности нашли соответствующее отражение в требованиях.
  Верификация проверяет внутреннюю согласованность, непротиворечивость и однозначность требований, а также их проверяемость и возможность проследить связи требований друг с другом, с кодом, тестами и другими проектными документами.

Тестирование и другие методы контроля качества ПО

Тестирование является методом контроля качества программного обеспечения, выполняющим проверку соответствия его характеристик требованиям. Однако тестирование — не единственный такой метод. Что же отличает его от других методов контроля качества?

Тестирование определяется как проверка соответствия поведения программной системы требованиям, выполняемая по результатам реальной работы этой системы в некотором конечном наборе специально созданных ситуаций [7,8]. При этом проверяемая система обычно называется *тестируемой системой* (system under test или SUT по-английски).

В этом определении можно отметить следующие аспекты.

- ***Проверка соответствия требованиям.***
  Как уже указывалось, тестирование возможно лишь при наличии требований к программе. Если от системы ничего не требуется, она может делать все, что угодно, и это нельзя считать неправильным.
  Тестирование позволяет проконтролировать качество программной системы ровно настолько, насколько полно, четко и недвусмысленно определены требований к ней. В тех случаях, когда явные требования не сформулированы, тестирование можно использовать, только основываясь на некоторых неявно подразумеваемых, но желательных свойствах тестируемой системы, например, отсутствии сбоев в ее работе.
- ***Результаты реальной работы системы.***
  Тестирование основывается на результатах реальной работы тестируемой системы. Проверяемая программа должна выполняться, чтобы проводимую при этом проверку можно было считать тестированием.
  Этим тестирование отличается от методов контроля качества ПО, основанных на анализе проектных документов, без выполнения самой программы. Таковы, например,

статический анализ корректности кода или *аналитическая верификация* при помощи доказательства нужных свойств программы. Тестирование отличается и от методов, использующих функционирование некоторых моделей программы, а не ее самой, например, от *проверки моделей* (model checking), при которой проверяется, что некоторая модель программы обладает некоторым свойством, и отдельно делаются выводы о возможности переноса этого свойства на саму программу.

Использование реальной работы тестируемой системы позволяет применять тестирование для проверки корректности поведения системы в ее рабочем окружении, на месте ее эксплуатации, что невозможно сделать при помощи других методов контроля качества ПО.

- ***Специально созданные ситуации.***

  Тестирование всегда выполняется в специально созданных ситуациях.

  Такие ситуации называются *тестовыми ситуациями*, а процедура или программа, при выполнении которой создается одна или несколько тестовых ситуаций и проверяется правильность поведения системы в них, называется *тестом*. Подготовка к проведению тестирования всегда включает подготовку или разработку тестов.

  Составляющие тестовых ситуаций будут рассматриваться в следующей лекции, а различным методам разработки тестов посвящено основное содержание этого курса.

  Эта характеристика отличает тестирование от пассивного наблюдения за поведением системы или *верификационного мониторинга* (runtime verification), при котором собираются и проверяются результаты реальной работы программы, но эта работа не управляется, не направляется на возникновение определенных ситуаций.

- ***Конечный набор ситуаций.***

  Тестирование всегда выполняется в конечном наборе ситуаций. Более того, возможное количество ситуаций, возникающих во время тестирования, ограничивается практическими соображениями достижения приемлемого компромисса между затратами времени и ресурсов проекта на разработку тестов и тестирование и пользой от него — количеством обнаруживаемых ошибок, полнотой и адекватностью получаемой оценки качества тестируемой системы.

  Практически значимые системы сейчас настолько сложны, что количество ситуаций, которые необходимо протестировать для полной проверки одной такой системы, превосходит возможности сколь угодно щедро финансируемого проекта и потребует не одной человеческой жизни для выполнения. Поэтому полное тестирование, хотя и возможно теоретически, в силу конечности любой вычислительной системы, практически совершенно невыполнимо.

  С одной стороны, это означает, что проведение тестирования никогда не дает полной гарантии корректности тестируемой системы, полного отсутствия в ней ошибок. С другой стороны, это обстоятельство делает чрезвычайно важным выбор тестов для выполнения. За счет выбора тестов можно получить как большой набор, выполняющийся долго и не дающий существенной информации о качестве тестируемой системы, так и компактный, но проверяющий большое количество разнообразных аспектов поведения системы и позволяющий оценить ее качество достаточно адекватно (хотя и без абсолютных гарантий).

  Критически важно для правильного выбора тестов использовать адекватный, отражающий реальную ситуацию *критерий полноты тестирования*. Различные способы определения таких критериев и методы выбора тестов в соответствии с ними также рассматриваются в следующих лекциях данного курса.

Перечисленные аспекты определяют как достоинства, так и недостатки тестирования по сравнению с другими методами контроля качества программного обеспечения. В отличие от аналитической верификации, тестирование не может гарантировать полного отсутствия ошибок в коде системы, но зато может проверить корректность ее работы на месте эксплуатации, при взаимодействии с другими системами, что сделать при помощи

аналитической верификации крайне тяжело. Тестирование всегда ограничено по ресурсам, но и предоставляет гибкие возможности управления полнотой и объемом проводимых проверок за счет разнообразных техник разработки и выбора тестов.

Для успешного проведения тестирования огромное значение имеют требования к тестируемой системе, использовавшиеся в ходе тестирования тестовые ситуации и критерии полноты тестирования. В общем случае и требования, и критерии полноты представляются в виде некоторых *моделей*, на основе которых выбираются тесты и выполняются проверки. Даже при отсутствии явных таких моделей, они присутствуют неявно, в сознании проектировщиков тестов и тестировщиков всегда есть какие-то критерии проверки правильности поведения тестируемой программы, представляющие требования, и критерии продолжения или прекращения тестирования после получения ряда результатов, основанные на понимании его неполноты или достаточной полноты. Поэтому можно сказать, что *тестирование всегда проводится на основе некоторых моделей*, быть может, не представленных явно.

Тестирование на основе моделей, которому посвящен этот курс, отличается только тем, что *все используемые при разработке, выборе и выполнении тестов модели формулируются явно*. Наиболее важную роль среди моделей, используемых при тестировании, играют модели требований, описывающие желательное поведение системы, и критерии полноты тестирования, определяющие набор разрабатываемых или выбираемых для выполнения тестов.

Ошибки в программном обеспечении

Наиболее наглядными результатами тестирования являются обнаруженные в тестируемой системе ошибки, то есть расхождения между ее реальным поведением и требованиями.

В литературе по программной инженерии термин «ошибка» используется в нескольких различных значениях.

- Иногда, хотя и достаточно редко, так называют произвольный *дефект* программной системы, будь то сбой, полностью разрушающий данные системы, неточно прорисованная буква на кнопке графического интерфейса пользователя или просто нестандартное форматирование исходного кода. В англоязычной литературе в этом смысле чаще всего употребляется термин *defect*.
- Часто ошибкой или *сбоем* называют наблюдаемое нарушение требований, проявляющееся при некотором сценарии работы рассматриваемой системы. В английском языке этому значению соответствует термин *failure*.
- Ошибка в коде программы, вызывающая наблюдаемое нарушение требований, и состоящая в неправильном использовании какой-то конструкции языка программирования, употреблении лишней конструкции или в пропуске необходимой. В англоязычной литературе ошибка такого рода называется *fault*.

  Ошибка такого рода определена нечетко, в отличие от предыдущего, — неправильная работа некоторого кода часто может быть исправлена несколькими разными способами. Если есть несколько конструкций, исправление каждой из которых удаляет эту ошибку, тяжело определить, какая именно из них ошибочна.
- Ошибка аналитика, архитектора или программиста, заключающаяся в неправильном понимании определенного требования или ограничения, в том, что какое-то требование забыто, или, наоборот, используется лишнее требование. По-английски такая ошибка называется обычно *error*.

  Иногда в англоязычной литературе термин *error* употребляется еще и в другом смысле — так называют некорректные, не соответствующие требованиям данные, возвращаемые системой в ответ на какой-либо запрос, или некорректные данные, возникающие в ходе внутренних вычислений в системе.

Основной смысл терминов failure, fault и error достаточно тесно связан с основными источниками ошибок, которых тоже три.

- *Неправильное понимание задач.*
  Очень часто люди не понимают или понимают неправильно то, что им пытаются сказать другие. Разработчики ПО тоже не всегда понимают, что именно нужно сделать. Дополнительным источником трудностей может служить отсутствие четкого понимания задач у самих пользователей и заказчиков — чаще всего они лишь приблизительно могут сформулировать проблему или попросить сделать несколько не то, что им действительно нужно.
  Выход из таких ситуаций один — нужно проводить тщательный анализ предметной области, уточнять каждое сформулированное требование, анализировать его причины и связи с другими требованиями и ограничениями.
- *Неправильное решение задач.*
  Даже правильно поняв, что нужно сделать, разработчики часто выбирают неправильный подход к тому, как это делать. Выбираемые решения могут обеспечивать лишь некоторые из требуемых свойств, могут решать поставленную задачу лишь для одного класса ситуаций из нескольких возможных, они могут подходить для данной задачи в теории, но плохо работать на практике, в конкретных обстоятельствах и в том окружении, в которых должна будет работать создаваемая система.
  Помочь в выборе правильного решения может сопоставление альтернативных решений и тщательный анализ их на предмет соответствия всем требованиям, поддержание постоянной связи с пользователями и заказчиками, предоставление им информации о предлагаемых решениях, демонстрация прототипов, анализ пригодности выбираемых решений для работы именно в том контексте, в котором они будут использоваться.
- *Неправильный перенос решений в код.*
  Имея правильное решение правильно понятой задачи, люди, тем не менее, способны сделать достаточно много ошибок при его воплощении. Корректному представлению решений в коде могут помешать как обычные опечатки, так и забывчивость программиста или его нежелание отказаться от привычных приемов, которые не дают возможности аккуратно записать именно то, что нужно.
  С ошибками такого рода можно справиться при помощи инспектирования кода, взаимного контроля, при котором разработчики внимательно читают код друг друга, опережающей разработки модульных тестов и тестирования.

Ошибки в программном обеспечении иногда приводят к серьезным инцидентам и значительным убыткам для организаций, использующим такое ПО. Гораздо чаще возникают более мелкие ошибки, не приводящие к серьезным последствиям. Зависимость между количеством ошибок и размером их последствий подчиняется закону Ципфа (Zipf) [9] — количество случающихся сбоев примерно обратно пропорционально величине наносимого ими ущерба.

Далее рассматривается несколько примеров ошибок в программном обеспечении, имевших серьезные последствия.

- Ошибка в системе управления космическим аппаратом Mariner 1 [10].
  Эта ошибка привела к уничтожению одного из первых кораблей, направлявшегося к Венере, через несколько минут после запуска 22 июля 1962 года.
  В ходе полета антенна связи вышла из строя, связь со службой управления была потеряна, и управление полетом взял на себя бортовой компьютер. Однако в одной из формул для расчета положения было забыто усреднение скорости по нескольким последовательным измеренным значениям — в результате небольшие перепады скорости стали рассматриваться системой как серьезные, она стала предпринимать «корректирующие» действия, в результате чего корабль сошел с курса и был уничтожен.

- Ошибка в программном обеспечении, управляющем аппаратом радиационной терапии Therac-25 [11].
  За 1985-1987 годы зафиксировано 6 инцидентов, связанных с этой ошибкой. В трех из них причиной смерти пациентов были признаны именно инциденты, приводившие к их повышенному облучению.
  Аппарат имел два режима облучения — мягкое облучение электронами и рентгеновское облучение. Во втором случае с источника электронных лучей снимался фильтр, который ослаблял их интенсивность, но между пациентом и источником излучения устанавливался специальный экран, падая на который мощные электронные лучи вызывали рентгеновское излучение.
  Ошибка проявлялась, когда оператор сначала включал первый режим, а потом слишком быстро переключал аппарат на второй. При этом ослабляющий фильтр снимался, а экран не устанавливался, и пациент подвергался очень интенсивному облучению электронными лучами. Кроме того, оператору при этом сообщалось, что пациент не получил никакой дозы, что не позволяло адекватно среагировать на происходящее.
  Ошибка возникала лишь иногда и была связана с несинхронизованным выполнением модулей, управлявших различными элементами аппарата. При эксплуатационном тестировании она не была обнаружена, поскольку операторы тогда еще не научились переключать режимы достаточно быстро.
- Ошибка в системе управления космическим аппаратом Фобос 1 [12].
  Привела к потере связи с кораблем, уже находившимся на пути к Марсу, 2 сентября 1988 года. Корабль перестал ориентироваться в пространстве, не смог сориентировать солнечные батареи и израсходовал аккумуляторы, поскольку были отключены навигационные приборы для определения положения относительно Солнца.
  Команда отключения приборов была в тестовой подпрограмме, использовавшейся на Земле для проверки работоспособности отдельных систем, удалить этот код не успели перед вылетом, поскольку он был записан в памяти, предназначенной только для чтения, а для ее замены требовалось существенное время. Казалось, однако, что в ходе полета эта программа никогда не будет вызвана. Но при передаче команд по корректировке курса 29 августа 1988 года была допущена ошибка — пропущен один символ, что привело к выполнению этой тестовой программы.
  Другой корабль этой серии, Фобос 2, был также потерян из-за какой-то ошибки в системе управления 27 марта 1989 года, уже на орбите вокруг Марса.
- 25 февраля 1991 года во время Первой войны в Персидском заливе американская система ПВО Patriot не смогла сбить иракскую ракету Скад, которая в результате попала в барак американской армии, убив 28 человек и ранив около ста [13].
  Причиной промаха Patriot, как выяснилось, было накопление ошибок округления за время работы системы. Время в ней измерялось в десятых долях секунды, а числа были представлены в 24-битном двоичном формате с плавающей точкой. При представлении 1/10 как двоичной дроби с 24-мя цифрами возникает небольшая ошибка.
  В рассматриваемом случае система Patriot работала без перезагрузки около 100 часов. За это время накопление погрешности определения времени дало ошибку около 1/3 секунды. Поскольку ракета Скад летит со скоростью 1700 м/с, ошибка в 1/10 секунды при расчете ее траектории уже не дает возможности ее сбить.
- Многочисленные ошибки в системе управления двигателями и навигационной системе считаются наиболее вероятной причиной катастрофы вертолета Chinook ZD 576 [14], произошедшей 2 июня 1994 года на мысе Кинтайр. В этой катастрофе погибли 25 экспертов и высокопоставленных сотрудников отдела разведки Великобритании в Северной Ирландии, что на значительное время парализовало работу этого отдела.
- Ошибка в системе управления ракеты Ариан-5 привела к ее уничтожению при первом запуске этой ракеты 4 июня 1996 года [15].

Долгое время эта ошибка, приведшая к убыткам в размере 500 миллионов долларов США, считалась самой дорогостоящей ошибкой в программной системе.
Ошибка состояла в том, что без изменений использовался модуль расчета траектории из системы управления ракетой Ариан-4. В нем горизонтальная составляющая скорости ракеты представлялась 16-битным числом. Ариан-5 могла выдерживать более значительные ускорения и большую кривизну траектории, из-за чего в ходе полета значение горизонтальной скорости вышло за пределы представимых 16-ю битами чисел. Специальной процедуры обработки такой ситуации не было, поэтому возникшее исключение обрабатывалось модулем обработки общих сбоев, который остановил данный процесс и запустил новый с теми же исходными данными, что вновь привело к той же ошибке. В результате система не смогла вычислить правильное текущее положение ракеты и стала использовать ранее полученные данные. Это привело к попытке «скорректировать» курс и «болтанию» ракеты, после чего она была уничтожена.

- Ошибка в системе управления космическим аппаратом Mars Climate Orbiter [16].
  Привела к его выходу на слишком низкую орбиту вокруг Марса 23 сентября 1999 года и к последовавшему за этим разрушению.
  Необходимые корректировки к движению корабля рассчитывались специальной программой на Земле и после передавались в виде команд двигателям аппарата. Ошибка состояла в том, что управляющая программа на Земле использовала значения импульсов в фунтах силы на секунду, а бортовая система передавала ей значения, измеренные в Ньютонах на секунду. В результате были использованы неправильные команды корректировки.
- Наконец, одной из причин сбоя в электроснабжении северо-востока Северной Америки, произошедшего 14 августа 2003 года, на несколько часов оставившего без электричества около 50 миллионов человек и приведшего к потерям на сумму около 6 миллиардов долларов США, была ошибка в программной системе оповещения о сбоях на электростанции [17].

Можно отметить, что в большинстве примеров ошибок, имевших тяжелые последствия, нельзя однозначно приписать всю вину за случившееся ровно одному недочету или дефекту, одному месту в коде. К тяжелым последствиям чаще всего приводят ошибки системного характера, затрагивающие многие элементы системы и различные аспекты ее устройства. Это значит, что при анализе такого происшествия обычно выявляется множество частных ошибок, нарушений действующих правил, недочетов в инструкциях и требованиях, которые совместно привели к создавшейся ситуации.

Даже если ограничиться рассмотрением только ПО, часто одно проявление ошибки (failure) может выявить несколько дефектов, находящихся в разных местах. Такие ошибки возникают, как показывает практика, в тех ситуациях, в которых поведение системы неоднозначно или недостаточно четко определяется требованиями (а иногда и вообще никак не определяется, что является признаком неполного понимания задачи). Поэтому разработчики различных модулей ПО имеют возможность по-разному интерпретировать те части требований, которые относятся непосредственно к их модулям, а также иметь разные мнения по поводу области ответственности каждого из взаимодействующих модулей в данной ситуации. Если различия в их понимании не выявляются достаточно рано, при разработке системы, то становятся «минами замедленного действия» в ее коде, приводя в дальнейшем к сбоям системы.

При подготовке и проведении тестирования эти закономерности распределения ошибок помогают находить их быстрее и с меньшими трудозатратами.

Литература

[1] ISO/IEC 9126-1:2001. Software engineering — Software product quality — Part 1: Quality model, 2001.

[2] ISO/IEC TR 9126-2:2003 Software engineering — Product quality — Part 2: External metrics, 2003.

[3] ISO/IEC TR 9126-3:2003 Software engineering — Product quality — Part 3: Internal metrics, 2003.

[4] ISO/IEC TR 9126-4:2004 Software engineering — Product quality — Part 4: Quality in use metrics, 2003.

[5] IEEE 830-1998. Recommended Practice for Software Requirements Specifications. New York: IEEE, 1998.

[6] IEEE 1233-1998. Guide for Developing System Requirements Specifications. New York: IEEE, 1998.

[7] ANSI/IEEE 610.12-1990. Glossary of Software Engineering Terminology. NY:IEEE, 1987.

[8] IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004.

[9] Статья Wikipedia о законе Ципфа http://en.wikipedia.org/wiki/Zipf%27s_law.

[10] http://nssdc.gsfc.nasa.gov/nmc/tmp/MARIN1.html.

[11] N. Levenson, C. S. Turner. An Investigation of the Therac-25 Accidents. IEEE Computer, 26(7):18-41, July 1993.

[12] R. Z. Sagdeev, A. V. Zakharov. Brief history of the Phobos mission. Nature 341:581-585, 1989.

[13] G. N. Lewis, S. Fetter, L. Gronlund. Casualties and Damage from Scud Attacks in the 1991 Gulf War, 1993. http://web.mit.edu/ssp/Publications/working_papers/wp93-2.pdf.

[14] http://www.publications.parliament.uk/pa/ld200102/ldselect/ldchin/25/2501.htm.

[15] http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html.

[16] Mars Climate Orbiter Mishap Investigation Board Phase I Report, 1999. ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf.

[17] http://www.nyiso.com/public/webdocs/newsroom/press_releases/2005/blackout_rpt_final.pdf.