

Качество ПО и методы его контроля

1.1	Качество ПО	1
1.1.1	Функциональность	9
1.1.2	Надежность (reliability)	10
1.1.3	Удобство использования (usability) или практичность	10
1.1.4	Производительность (efficiency) или эффективность	10
1.1.5	Удобство сопровождения (maintainability)	11
1.1.6	Переносимость (portability)	11
1.2	Методы контроля качества	13
1.2.1	Тестирование	14
1.3	Ошибки в ПО	18

1.1 Качество ПО

Как проверить, что требования заданы достаточно полно и описывают все, что хотелось бы видеть в будущей программной системе? Для этого служит понятие качества ПО. Именно понятие качественного ПО соответствует представлению о том, что программа достаточно успешно справляется со всеми возложенными на нее задачами и не приносит проблем ни конечным пользователям, ни их начальству, ни службе поддержки, ни специалистам по продажам.

Если попросить группу людей высказать своё мнение по поводу того, что такое качественное ПО, можно получить следующие варианты ответов.

- Его легко использовать
- Оно демонстрирует хорошую производительность
- В нем нет ошибок
- Оно не портит пользовательские данные при сбоях
- Его можно использовать на разных платформах
- Оно может работать 24 часа в сутки и 7 дней в неделю
- В него легко добавлять новые возможности
- Оно удовлетворяет потребности пользователей
- Оно хорошо документировано

Все это действительно имеет непосредственное отношение к качеству ПО. Но все эти ответы выделяют характеристики, важные для конкретного пользователя, разработчика или группы таких лиц. Для того, чтобы удовлетворить потребности всех заинтересованных сторон (конечных пользователей, заказчиков, разработчиков, администраторов систем, в которых оно будет работать, регулирующих организаций и пр), для достижения прочного положения разрабатываемого ПО на рынке и повышения потенциала его развития важен учет всей совокупности характеристик ПО, важных для всех заинтересованных лиц.

Приведенные выше ответы показывают, что качество ПО может быть описано большим набором разнородных характеристик. Такой подход к описанию сложных понятий называется холистическим (от греческого *олос*, целое). Он не дает единой концептуальной основы для рассмотрения затрагиваемых вопросов, какую дает целостная система представлений (например, Ньтоновская механика в физике или классическая теория вычислимости на основе машин Тьюринга), но позволяет, по крайней мере, не упустить ничего достаточно важного.

1.1.1 Теория иерархии потребностей

Потребности человека связаны со свойствами человеческой личности. Психика человека крайне сложна, и достаточно полных теорий потребностей человека еще не построено. Тем не менее, сейчас существует ряд теорий, описывающих виды и взаимоотношения потребностей, на основании которых разработчик изделий может действовать достаточно уверенно и добиваться хороших практических результатов.

Одной из наиболее распространенных теорий является теория иерархии потребностей английского ученого Авраама Маслоу (Abraham Maslow), выдвинутая им в 50-е годы нашего века. По Маслоу, существует 5 групп или уровней потребностей:

Основные или физиологические потребности – такие, как потребности в пище, одежде, жилище и т.д., которые определяются биологической природой человека

Потребности в защищенности от “ударов судьбы”, таких, как несчастные случаи, болезни, инвалидность, нищета и др., которые могут нарушить возможность удовлетворения потребностей предыдущего уровня – физиологических потребностей

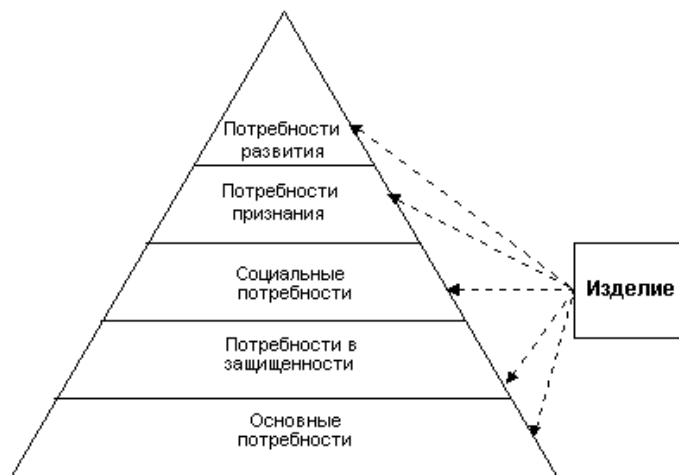
Социальные потребности, то есть потребности в общении, взаимоотношениях с другими людьми. По Маслоу, потребности каждого уровня связаны с возможностью удовлетворения потребностей предыдущего уровня, и социальные потребности вызваны стремлением более полно удовлетворить потребности в защищенности

Потребности признания или потребности “Эго”. Это – потребности в престиже, уважении окружающих, славе и т.д.

Потребности развития – наивысший уровень потребностей- потребности в самосовершенствовании, или потребности развития.

По Маслоу, переход к потребности более высокого уровня происходит, если потребность предыдущего уровня удовлетворена на 100%; современные психологи считают, что этот процент меньше - порядка 70% и даже менее. Иерархия потребностей конкретного человека во многом определяется уровнем развития его психики, она меняется от человека к человеку и различна у одного человека в различные периоды его жизни. С развитием психики человека потребности более высокого уровня становятся более важными по сравнению с потребностями более низкого уровня.

Можно считать, что все эти виды потребностей существуют не только для отдельного человека, но и для коллективов людей, в том числе предприятий и общества в целом.

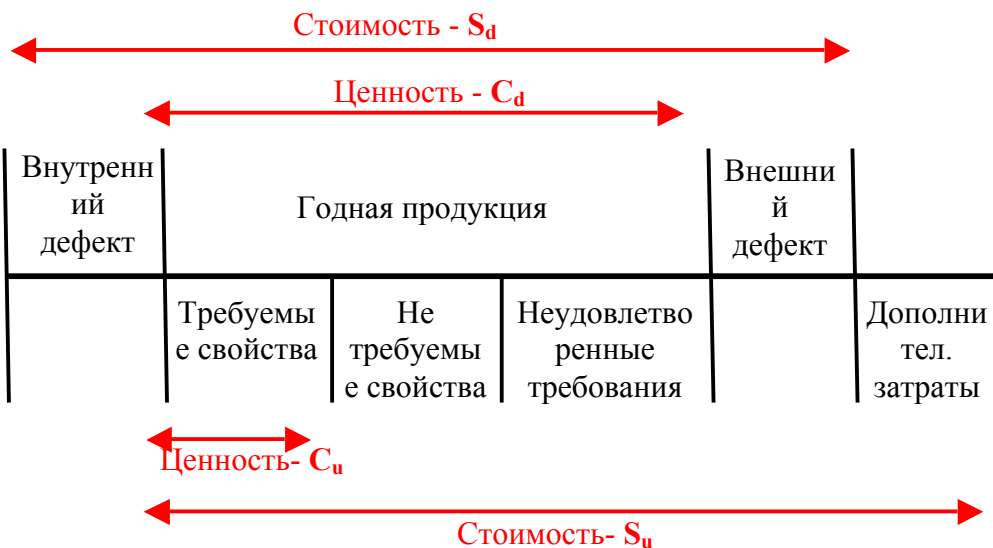


1.1.2 Мера качества: ценность и стоимость

Введенное понятие качества есть «качественный» показатель продукта. Можно ли качество измерить? Наиболее общим является подход, при котором вводятся понятия:

- Ценность изделия - способность удовлетворять потребности.
- Качество изделия - соответствие между свойствами изделия и его ценностью.
- Мера качества - соотношение ценности и стоимости.

Производитель



Потребитель

При этом оказывается, что для производителя и потребителя эти показатели имеют различные значения. Для производителя вся продукция, не содержащая дефектов, которые препятствовали бы продаже этой продукции, имеет ценность. Для потребителя же ценность имеют только те свойства продукции, которые соответствуют его ожиданиям. Важными являются три основные соотношения между ценностью и стоимостью:

- Мера качества для потребителя: $Q_u = C_u / S_u$
- Мера качества для производителя: $Q_d = C_d / S_d$
- Конкурентоспособность продукта: $K = C_u / C_d$

1.1.3 Эволюция методов обеспечения качества

На разных этапах развития товарного производства применялись различные методы обеспечения качества. Выделяют три основные фазы эволюции методов обеспечения качества:

- Фаза отбраковки
- Фаза управления качеством
- Фаза планирования качества

1.1.3.1 Фаза отбраковки

Началась вместе с зарождением ремесленного производства. Отдельные мастера проверяли свою собственную работу Цеховые организации средневековых городов, которые, если выражаться современным языком, сертифицировали мастеров - присуждали звание мастера после серьезных испытаний качества изделия. При этом каждое изделие было индивидуальным.

Следующий шаг этой фазы был связан с применением *стандартов* (калибров). В 70х гг. XIX века в оружейном производстве (заводы Сэмюэля Кольта) родилась идея стандартного качества – изделия собирались не из подогнанных друг к другу деталей, а из случайно выбранных из партии, то есть взаимозаменяемых деталей. Перед сборкой эти детали проверялись с помощью калибров, и негодные отбраковывались. Контроль и отбраковку осуществляли специально обученные контролеры.

В конце XIX века Генри Мартин Леланд ("Кадиллак") впервые применил в автомобильном производстве работу по калибрам и придумал пару "проходной" и "непроходной" калибр. В марте 1908 г. эксперты Британского автомотоклуба отобрали случайным образом 3 экземпляра из экспортной партии автомобилей "Кадиллак", прибывшей в Англию, и разобрал их до последнего винтика. Все детали свалили в кучу, а затем кое-какие детали из этой кучи изъяли и заменили запчастями, позаимствованными опять же наугад в местном агентстве по продаже и обслуживанию автомобилей "Кадиллак". Потом группа механиков, вооруженная только отвертками и гаечными ключами, собрала машины заново и запустила моторы. Две машины завелись с первой попытки, а одна – со второй, и все они отправились на длительную обкатку по только что сданному в эксплуатацию автодрому Бруклэндс. И когда вновь собранные машины подтвердили полную идентичность своих ходовых характеристик параметрам автомобилей заводской сборки, Британский автомотоклуб выдал фирме "Кадиллак" диплом и серебряный кубок с надписью "За стандартизацию". После этого на табличке с гербом фирмы на автомобилях "Кадиллак" появилась надпись "Standard of the world" – образец для подражания для всего мира.

Выходной контроль вместо входного. В начале XX века Форд впервые ввел вместо входного контроля комплектующих на сборке выходной контроль на тех производствах, где эти комплектующие изготавливались, то есть на сборку стали поступать только годные, качественные изделия. Он также создал отдельную службу технического контроля, независимую от производства.

Замена контроля составлял один из элементов производственной системы Форда-Тейлора, которая была разработана Тейлором и внедрена на заводах Форда. Эта система основана на концепции научного менеджмента, включившая системный подход, кадровый менеджмент, идею разделения ответственности между работниками и управленцами в обеспечении качественной и эффективной работы организации, идею научного нормирования труда. В основных чертах просуществовала до настоящего времени и является моделью организации производства большинства современных предприятий. Только в 70-е годы ей на смену стала приходить другая концепция (*производственная система Тойота*).

Концепция фазы отбраковки состоит в том, что потребитель должен получать только годные изделия, т.е. изделия, соответствующие стандартам. Основные усилия должны быть направлены на то, чтобы не годные изделия (брак) были бы отсечены от потребителя.

Результат фазы состоял в том, что численность контролеров стала составлять до 30 - 40% от численности производственных рабочих, иногда и более. Повышение качества всегда сопровождается ростом затрат на его обеспечение. Т.е. цели повышения эффективности

производства и повышения качества изделий являются противоречивыми (не могут быть достигнуты одновременно).

1.1.3.2 Фаза управления качеством

Цель фазы управления качеством – сосредоточить усилия не на том, как обнаружить и изъять негодные изделия до их отгрузки покупателю, а на том, как увеличить выход годных изделий в техпроцессе. В этой фазе выделяют два этапа:

- Управление процессами – переход от контроля к управлению отдельными процессами
- Управление производством – переход от управления отдельными процессами к управлению производством в целом

Старт первого этапа май 1924г. Вестерн Электрик, США. Точкой отсчета считаются работы, выполненные в Отделе технического контроля фирмы Вестерн Электрик, США. В мае 1924 г. сотрудник отдела доктор Шухарт передал своему начальнику короткую записку, которая содержала метод построения диаграмм, известных нынче по всему миру как контрольные карты Шухарта. Контрольные карты основаны на статистических методах оценки стабильности протекания различных технологических процессов. На основе выборок – замеров контрольных показателей процессов (к-ва брака в контрольной партии) – строится среднее значение и допустимые верхнее и нижнее отклонения. Процесс не должен выходить за допустимые значения. Статистические методы, предложенные Шухартом, дали в руки управленцев инструмент, который позволил контролировать качество производства комплекующих.

Для первого этапа было характерно создание аудиторских служб по качеству, которые в отличие от отделов технического контроля занимались не разбраковкой продукции, а путем контроля небольших выборок из партий изделий проверяла работоспособность системы обеспечения качества на производстве.

Внедрение таких служб контроля качества отдельных процессов значительно повысило эффективность производства, но есть предел, определяемый системой. Каждый производственный процесс имеет определенный предел выхода годных изделий, и это предел определяется не процессом самим по себе, а системой, то есть всей совокупностью деятельности предприятия, организации труда, управления, в которой этот процесс протекает.

Второй этап управления качеством (фаза менеджмента качества) связан с повышением качества путем управления предприятием. Составными элементами этого этапа являются:

- Совершенствование системы в целом, а не только отдельных производственных процессов.
- Непосредственное участие высшего руководства компаний в проблемах качества.
- Обучение всех сотрудников компаний сверху донизу основным методам обеспечения качества.
- Упор на мотивацию сотрудников на высококачественный труд .
- Концепция «0 дефектов на всех участках».

Начало этапа – 1950 г., Япония. В 1950 доктор Эдвардс Деминг получил приглашение от японского союза ученых и инженеров принять участие в программе восстановления японской промышленности. Там он и предложил программу менеджмента качества из 14 пунктов, разработал принцип постоянного улучшения качества. За 12 лекций доктор Деминг встретился с сотнями ведущих менеджеров японских фирм. Основная идея программы:

"Основа качества продукции – качество труда и качественный менеджмент на всех уровнях, то есть такая организация работы коллективов людей, когда каждый работник получает удовольствие от своей работы". Программа была активно воспринята и произвела революцию в японской промышленности.

Основные идеи этой программы были разработаны и пытались применяться в США, но там вначале не нашли должного отклика. Только в Японии они нашли благодатную почву. После прочтения лекций в Японии, Деминг предрек: «Через несколько лет мир содрогнется от обилия качественных японских товаров».

В 1957 г. Фейгенбаум опубликовал статью, в которой изложил принципы тотального управления качеством и параллельного (одновременного) инжиниринга – принципы TQM – Total Quality Management. Эти принципы лежат в основе современных систем управления качеством.

Второй этап – менеджмент качества отражает современное представление о системе управления качеством. Противоречие между повышением качества и ростом эффективности производства в его прежних формах было преодолено – применение новых идей управления позволило одновременно повышать качество и снижать затраты на производство. Потребитель практически во всех странах стал получать товары и услуги высочайшего качества по доступной цене – идея “общества потребления” воплотилась в жизнь.

Сложилась концепция стандартизованного качества – качество определяет производитель, а покупатель берет товар или не берет. Внутренним противоречием этой фазы является вопрос: что делать при ошибке определения запросов, когда годные товары не находят спроса?

1.1.3.3 Фаза планирования качества

Цель – планирование запросов. Эта фаза стала зарождаться в середине 60х гг. как развитие идей предыдущей фазы в направлении более полного удовлетворения запросов потребителей.

Предпосылками возникновения фазы планирования качества являются:

- Развитие мирового рынка товаров и услуг.
- Резкое обострение конкуренции на этом рынке.
- Политика государственной защиты интересов потребителей .
- Развитие теории надежности изделий.
- Внедрение вычислительной техники и САПР.

Основы концепции новой фазы:

- Большая часть дефектов изделий закладывается на стадии разработки из-за недостаточного качества проектных работ.
- Математическое моделирование свойств и процессов. Перенос центра тяжести работ по созданию изделия с натурных испытаний опытных образцов или партий на математическое моделирование свойств изделий, а также моделирование процессов производства изделий, что позволяет обнаружить и устранить конструкторские и технологические дефекты еще до начала стадии производства.
- Снижение цены – высокое качество необходимо предоставить потребителю за приемлемую цену, которая постоянно снижается, т.к. конкуренция на рынках очень высока.

В настоящее время эта фаза только зарождается, и ее концепция еще окончательно не сформировалась.

1.1.4 Качество программного обеспечения

Качество программного обеспечения определяется в стандарте ISO 9126 как вся совокупность его характеристик, относящихся к возможности удовлетворять высказанные или подразумеваемые потребности всех заинтересованных лиц.

Стандарт ISO 9126 дает следующее представление качества.

При рассмотрении качества ПО различаются понятия **внутреннего качества**, связанного с характеристиками ПО самого по себе, без учета его поведения, **внешнего качества**, характеризующего ПО с точки зрения его поведения, и **качество ПО при использовании** в различных контекстах — то качество, которое ощущается пользователями при конкретных сценариях работы ПО. Для всех этих взглядов на качество введены метрики, позволяющие оценить его. Кроме того, при создании качественного ПО существенно качество технологических процессов его разработки. Взаимоотношения между этими аспектами качества по схеме, принятой в ISO 9126, показано на Рис. 1.

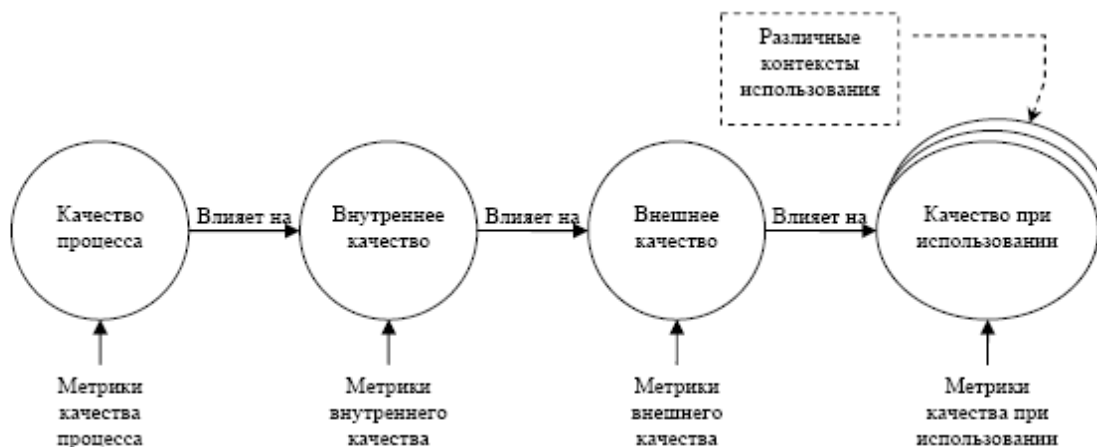


Рисунок Ошибка! Текст указанного стиля в документе отсутствует..1

Общие принципы обеспечения качества процессов производства во всех отраслях экономики регулируются набором стандартов ISO 9000. Наиболее важные для разработки ПО стандарты в его составе следующие.

- **ISO 9000:2000 Quality management systems — Fundamentals and vocabulary** [5].
Системы управления качеством — Основы и словарь. (Аналог ГОСТ Р-2001).
- **ISO 9001:2000 Quality management systems — Requirements. Models for quality assurance in design, development, production, installation, and servicing** [6].
Системы управления качеством — Требования. Модели для обеспечения качества при проектировании, разработке, коммерциализации, установке и обслуживании. Определяет общие правила обеспечения качества результатов во всех процессах жизненного цикла. (Аналог ГОСТ Р-2001).

Этот стандарт выделяет следующие процессы

- Управление качеством
- Управление ресурсами
- Развитие системы управления
- Исследования рынка

- Проектирование продуктов
- Приобретения
- Производство
- Оказание услуг
- Защита продуктов
- Оценка потребностей заказчиков
- Поддержка коммуникаций с заказчиками
- Поддержка внутренних коммуникаций
- Управление документацией
- Ведение записей о деятельности
- Планирование
- Обучение персонала
- Внутренние аудиты
- Оценки управления
- Мониторинг и измерения
- Управление несоответствиями
- Постоянное совершенствование
- Управление и развитие системы в целом

Для каждого процесса требуется иметь планы развития процесса, состоящие как минимум из следующих разделов.

- Проектирование процесса
- Документирование процесса
- Реализация процесса
- Поддержка процесса
- Мониторинг процесса
- Управление процессом
- Усовершенствование процесса

Помимо поддержки и развития системы процессов, нацеленных на удовлетворение нужд заказчиков и пользователей, ISO 9001 требует

Определить, документировать и развивать собственную систему качества на основе измеримых показателей.

Использовать эту систему качества в качестве средства управления процессами, нацеливая их на большее удовлетворение нужд заказчиков, планируя и постоянно отслеживая качество результатов всех видов деятельности, в том числе и самого управления.

Обеспечить использование качественных ресурсов, качественного (компетентного, профессионального) персонала, качественной инфраструктуры и качественного окружения.

Постоянно контролировать соблюдение требований к качеству на практике, во всех процессах проектирования, производства, предоставления услуг и при приобретениях.

Предусмотреть процесс устранения дефектов, определить и контролировать качество результатов этого процесса.

Ранее использовавшиеся стандарты ISO 9002:1994 Quality systems — Model for quality assurance in production, installation and servicing и ISO 9003:1994 Quality systems — Model for quality assurance in final inspection and test в 2000 году были заменены соответствующими им частями ISO 9001.

• ISO 9004:2000 Quality management systems — Guidelines for performance improvements [7]. Системы управления качеством. Руководство по улучшению деятельности. (Аналог ГОСТ Р-2001).

• ISO/IEC 90003:2004 Software engineering — Guidelines for the application of ISO 9001:2000 to computer software [8]. Руководящие положения по применению стандарта ISO 9001 при разработке, поставке и обслуживании программного обеспечения. Этот стандарт конкретизирует положения ISO 9001 для разработки программных систем, с упором на обеспечение качества при процессе проектирования. Он также определяет некоторый набор техник и процедур, которые рекомендуется применять для контроля и обеспечения качества разрабатываемых программ.

Стандарт ISO 9126 [1-4] предлагает использовать для описания внутреннего и внешнего качества ПО многоуровневую модель. На верхнем уровне выделено 6 основных характеристик качества ПО. Каждая характеристика описывается при помощи нескольких входящих в нее атрибутов. Для каждого атрибута определяется набор метрик, позволяющих оценить этот атрибут. Набор характеристик и атрибутов качества согласно ISO 9126 показан на Рис. 2.



Рисунок Ошибка! Текст указанного стиля в документе отсутствует..2

1.1.5 Функциональность

Способность ПО в определенных условиях решать задачи, нужные пользователям. Определяет, что именно делает ПО, какие задачи оно решает.

Функциональная пригодность (suitability). Способность решать нужный набор задач.

Точность (accuracy). Способность выдавать нужные результаты.

Способность к взаимодействию (interoperability). Способность взаимодействовать с нужным набором других систем.

Соответствие стандартам и правилам (compliance). Соответствие ПО имеющимся промышленным стандартам, нормативным и законодательным актам, другим регулирующим нормам.

Защищенность (security). Способность предотвращать неавторизованный, т.е. без указания лица, пытающегося его осуществить, и не разрешенный доступ к данным и программам.

1.1.6 Надежность (reliability)

Способность ПО поддерживать определенную работоспособность в заданных условиях.

Зрелость, завершенность (maturity). Величина, обратная к частоте отказов ПО.

Устойчивость к отказам (fault tolerance) Способность поддерживать заданный уровень работоспособности при отказах и нарушениях правил взаимодействия с окружением.

Способность к восстановлению (recoverability). Способность восстанавливать определенный уровень работоспособности и целостность данных после отказа, необходимые для этого время и ресурсы.

Соответствие стандартам надежности (reliability compliance). Этот атрибут добавлен в 2001 году.

1.1.7 Удобство использования (usability) или практичность

Способность ПО быть удобным в обучении и использовании, а также привлекательным для пользователей.

Понятность (understandability). Показатель, обратный к усилиям, затрачиваемым пользователями, чтобы воспринять набор понятий, на которых основано ПО, и их применимость для решения своих задач.

Удобство обучения (learnability). Показатель, обратный к усилиям, затрачиваемым пользователями чтобы научиться работе с ПО.

Удобство работы (operability). Показатель, обратный к усилиям, предпринимаемым пользователями, чтобы решать свои задачи с помощью ПО.

Привлекательность (attractiveness). Способность ПО быть привлекательным для пользователей. Этот атрибут добавлен в 2001.

Соответствие стандартам удобства использования (usability compliance). Этот атрибут добавлен в 2001.

1.1.8 Производительность (efficiency) или эффективность

Способность ПО при заданных условиях обеспечивать необходимую работоспособность по отношению к выделяемым для этого ресурсам. Можно определить ее и как отношение получаемых с помощью ПО результатов к затрачиваемым на это ресурсам.

Временная эффективность (time behaviour). Способность ПО выдавать ожидаемые результаты, а также обеспечивать передачу необходимого объема данных за отведенное время.

Эффективность использования ресурсов (resource utilisation). Способность решать нужные задачи с использованием определенных объемов ресурсов определенных видов. Имеются в виду такие ресурсы, как оперативная и долговременная память, сетевые соединения, устройства ввода и вывода, и пр.

Соответствие стандартам производительности (efficiency compliance). Этот атрибут добавлен в 2001.

1.1.9 Удобство сопровождения (maintainability)

Удобство проведения всех видов деятельности, связанных с сопровождением программ.

Анализируемость (analyzability) или удобство проведения анализа. Удобство проведения анализа ошибок, дефектов и недостатков, а также удобство анализа на предмет необходимых изменений и их возможных эффектов.

Удобство внесения изменений (changeability). Показатель, обратный к трудозатратам на проведение необходимых изменений.

Стабильность (stability). Показатель, обратный к риску возникновения неожиданных эффектов при внесении необходимых изменений.

Удобство проверки (testability). Показатель, обратный к трудозатратам на проведение тестирования и других видов проверки того, что внесенные изменения привели к нужным эффектам.

Соответствие стандартам удобства сопровождения (maintainability compliance). Этот атрибут добавлен в 2001.

1.1.10 Переносимость (portability)

Способность ПО сохранять работоспособность при переносе из одного окружения в другое, включая организационные, аппаратные и программные аспекты окружения. Иногда эта характеристика называется в русскоязычной литературе мобильностью. Однако термин «мобильность» стоит резервировать для перевода «mobility» — способности ПО и компьютерной системы в целом сохранять работоспособность при ее физическом перемещении в пространстве.

Адаптируемость (adaptability). Способность ПО приспосабливаться к различным окружениям без проведения для этого действий, помимо заранее предусмотренных.

Удобство установки (installability). Способность ПО быть установленным или развернутым в определенном окружении.

Способность к сосуществованию (coexistence). Способность ПО сосуществовать с другими программами в общем окружении, деля с ним ресурсы.

Удобство замены (replaceability) другого ПО данным. Способность ПО использоваться вместо другого ПО для решения тех же самых задач в заданном окружении.

Соответствие стандартам переносимости (portability compliance). Этот атрибут добавлен в 2001.

Для описания качества ПО при использовании стандарт ISO 9126-4 предлагает следующий набор характеристик.

- **Эффективность (effectiveness).** Это способность ПО предоставлять пользователям возможность решать их задачи с необходимой точностью при использовании в заданном контексте.
- **Продуктивность (productivity).** Способность ПО предоставлять пользователям определенные результаты в рамках ожидаемых затрат ресурсов.
- **Безопасность (safety).** Способность ПО обеспечивать необходимо низкий уровень риска нанесения ущерба жизни и здоровью людей, бизнесу, собственности или окружающей среде.
- **Удовлетворенность (satisfaction).** Способность ПО приносить удовлетворение пользователям при использовании в заданном контексте.

Помимо перечисленных характеристик и атрибутов качества стандарт ISO 9126:2001 определяет наборы метрик для оценки каждого атрибута. Приведем следующие примеры таких метрик.

- Полнота реализации функций — процент реализованных функций по отношению к перечисленным в требованиях. Используется для измерения функциональной пригодности.
- Корректность реализации функций — правильность их реализации по отношению к требованиям. Используется для измерения функциональной пригодности.
- Отношение числа обнаруженных дефектов к прогнозируемому. Используется для определения зрелости.
- Отношение числа проведенных тестов к общему их числу. Используется для определения зрелости.
- Отношение числа доступных проектных документов к указанному в их списке. Используется для измерения удобства проведения анализа.
- Наглядность и полнота документации. Используется для оценки понятности.

Перечисленные характеристики и атрибуты качества ПО позволяют систематически описывать требования к нему, определяя, какие свойства ПО по данной характеристике хотят видеть заинтересованные стороны. Таким образом, требования должны определять следующее.

- Что ПО должно делать, например: Позволять клиенту оформить заказы и обеспечить их доставку; Обеспечивать контроль качества строительства и отслеживать проблемные места; Поддерживать нужные характеристики автоматизированного процесса производства, предотвращая аварии и оптимальным образом используя имеющиеся ресурсы.
- Насколько оно должно быть надежно, например: Работать 7 дней в неделю и 24 часа в сутки; Допускается неработоспособность в течение не более 3 часов в год. Никакие введенные пользователями данные при отказе не должны теряться.
- Насколько им должно быть удобно пользоваться, например: Покупатель должен легко находить нужный ему товар; Инженер по специальности «строительство мостов» должен в течение одного дня разобраться в 80% функций системы.
- Насколько оно должно быть эффективно, например: Поддерживать обслуживание до 10000 запросов в секунду; Время отклика на запрос при максимальной загрузке не должно превышать 3 с; Время реакции на изменение параметров процесса производства не должно превышать 0.1 с; На обработку одного запроса не должно тратиться более 1 МВ оперативной памяти.
- Насколько удобно должно быть его сопровождение, например: Добавление в систему нового вида запросов не должно требовать более 3 человеко-дней; Добавление поддержки нового процесса производства не должно занимать более 24 человеко-месяцев.
- Насколько оно должно быть переносимо и заменяемо, например: ПО должно работать на операционных системах Linux, Windows XP и MacOS X; ПО должно работать с документами в форматах MS Word 97 и HTML; ПО должно сохранять файлы отчетов в форматах MS Word 2000, MS Excel 2000, HTML, RTF и в виде обычного текста. ПО должно сопрягаться с существующей системой записи данных о заказах.

Приведенные атрибуты качества закреплены в стандартах, но это не значит, что они полностью исчерпывают понятие качества ПО. Так, в стандарте ISO 9126 полностью отсутствуют характеристики, связанные с **мобильностью ПО (mobility)**, т.е. способностью программы работать при физических перемещениях машины, на которой она работает. Вместо надежности многие исследователи предпочитают рассматривать более общее понятие **добротности (dependability)**, описывающее способность ПО поддерживать определенные показатели качества по основным характеристикам (функциональности, производительности, удобству использования) с заданными вероятностями выхода за их рамки и заданными рисками возможных нарушений. Кроме того, активно исследуются понятия удобства использования, безопасности и защищенности ПО — они кажутся большинству специалистов гораздо более сложными, чем это описывается данным стандартом.

1.2 Методы контроля качества

Как контролировать качество системы? Как точно узнать, что программа делает именно то, что нужно, и ничего другого? Как определить, что она достаточно надежна, переносима, удобна в использовании? Ответы на этот вопрос можно получить с помощью процессов верификации и валидации.

- **Верификация** обозначает проверку того, что продукт делался правильно, т.е. проверку того, что он разрабатывался в соответствии со всеми требованиями по отношению к процессу и этапам разработки. К верификации относятся все проверки соответствия результатов некоторого этапа разработки требованиям, выдвинутым к ним на предыдущем этапе.
- **Валидация** — это проверка того, что сам продукт правилен, т.е. подтверждение того, что он действительно удовлетворяет требованиям и ожиданиям пользователей, заказчиков и других заинтересованных сторон.

Эффективность верификации и валидации, как и эффективность разработки ПО в целом зависит от точности и корректности формулировки требований к программному продукту.

Основой любой системы обеспечения качества являются методы его обеспечения и контроля. **Методы обеспечения качества** [9] представляют собой техники, гарантирующие достижение определенных показателей качества при их применении.

Методы контроля качества предназначены для того, чтобы убедиться, что определенные характеристики качества ПО достигнуты. Сами по себе они не могут помочь их достижению, они лишь помогают определить, удалось ли получить в результате то, что хотелось, или нет, а также найти ошибки, дефекты и отклонения от требований. Методы контроля качества ПО можно классифицировать следующим образом.

1. Методы и техники, связанные выяснением свойств ПО во время его работы. Это, прежде всего, все виды тестирования, а также профилирование и измерение количественных показателей качества, которые можно определить по результатам работы ПО — эффективности по времени и другим ресурсам, надежности, доступности и пр.
2. Методы и техники, связанные с определением показателей качества на основе симуляции работы ПО с помощью моделей разного рода. К этому виду относятся проверка на моделях (model checking), а также прототипирование (макетирование), использованное для оценки качества принимаемых решений.
3. Методы и техники, предназначенные для выявления нарушений формализованных правил построения исходного кода ПО, проектных моделей и документации. К методам такого рода относится инспектирование кода, заключающееся в целенаправленном поиске

определенных дефектов и нарушений требований в коде на основе набора шаблонов, автоматизированные методы поиска ошибок в коде, не основанные на его интерпретации, методы проверки документации на согласованность и соответствие стандартам.

4. Методы и техники, связанные с обычным или формализованным анализом проектной документации и исходного кода для выявления их свойств. К этой группе относятся многочисленные методы анализа архитектуры ПО, о которых пойдет речь в следующей лекции, методы формального доказательства свойств ПО и формального анализа эффективности применяемых алгоритмов.

1.2.1 Тестирование

Тестирование — это проверка соответствия ПО требованиям, осуществляемая с помощью наблюдения за его работой в специальных, искусственно построенных ситуациях. Такого рода ситуации называют тестовыми или просто **тестами**.

Тестирование — конечная процедура. Набор ситуаций, в которых будет проверяться тестируемое ПО всегда конечен. Более того, он должен быть настолько мал, чтобы тестирование можно было провести в рамках проекта разработки ПО, не слишком увеличивая его бюджет и сроки. Это означает, что при тестировании всегда проверяется очень небольшая доля всех возможных ситуаций. По этому поводу Дейкстра (Dijkstra) сказал, что тестирование позволяет точно определить, что ошибка есть в программе, но никогда не позволяет утверждать, что там нет ошибок.

Тем не менее, тестирование может использоваться для достаточно уверенного вынесения оценок о качестве ПО. Для этого необходимо иметь **критерии полноты тестирования**, описывающие важность различных ситуаций для оценки качества, а также эквивалентности и зависимости между ними (т.е. все равно в какой из ситуаций, А или В, проверять правильность работы ПО, или, если программа правильно работает в ситуации А, то, скорее всего, в ситуации В все тоже будет правильно). Часто критерий полноты тестирования задается при помощи определения эквивалентности ситуаций, дающей конечный набор классов ситуаций. В этом случае считают, что все равно, какую из ситуаций одного класса использовать в качестве теста. Такой критерий называют **критерием тестового покрытия**, а процент классов эквивалентности ситуаций, покрытых во время тестирования — достигнутым **тестовым покрытием**.

Таким образом, основные задачи тестирования — построить такой набор ситуаций, который был бы достаточно репрезентативен и позволял бы завершить тестирование с достаточной степенью уверенности в правильности проверяемого ПО вообще, и убедиться, что в конкретной ситуации ПО работает правильно, в соответствии с требованиями.

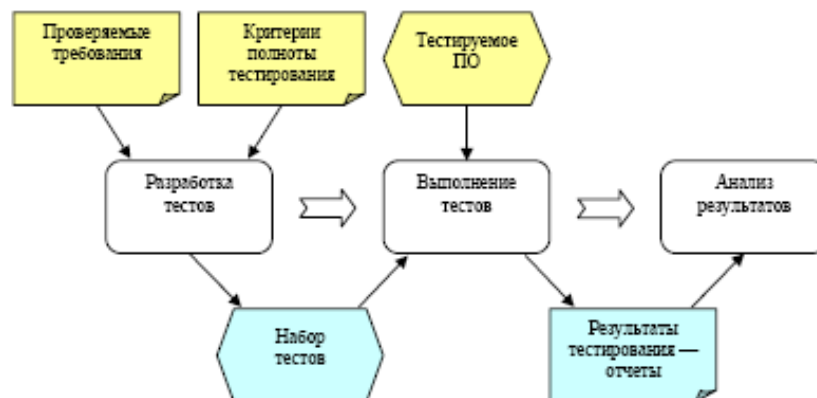


Рисунок Ошибка! Текст указанного стиля в документе отсутствует.3 – Схема процесса тестирования

Тестирование — наиболее широко применяемый метод контроля качества. Для оценки многих атрибутов качества не существует других эффективных способов, кроме тестирования.

Организация тестирования ПО регулируется следующими стандартами.

- IEEE 829-1998 Standard for Software Test Documentation. Описывает виды документов, служащих для подготовки тестов.
- IEEE 1008-1987 (R1993, R2002) Standard for Software Unit Testing. Описывает организацию модульного тестирования.
- ISO/IEC 12119:1994 (аналог AS/NZS 4366:1996 и ГОСТ Р-2000, также принят IEEE под номером IEEE 1465-1998) Information Technology. Software packages — Quality requirements and testing.

Тестировать можно соблюдение любых требований, соответствие которым выявляется во время работы ПО. Из характеристик качества по ISO 9126 этим свойством не обладают только атрибуты удобства сопровождения. Поэтому выделяют виды тестирования, связанные с проверкой определенных характеристик и атрибутов качества — тестирования функциональности, надежности, удобства использования, переносимости и производительности, а также тестирование защищенности, функциональной пригодности и пр. Кроме того, особо выделяют **нагрузочное** или **стрессовое тестирование**, проверяющее работоспособность ПО и показатели его производительности в условиях повышенных нагрузок — большом количестве пользователей, интенсивном обмене данными с другими системами, большим объемом передаваемых или используемых данных, и пр.

На основе исходных данных, используемых для построения тестов, тестирование делят на следующие виды.

1. **Тестирование на соответствие (conformance testing)** — тесты для него и критерии полноты тестирования строятся на основе каких-то достаточно четко зафиксированных требований (в спецификациях, стандартах, внутренних нормативных документах). Частным случаем является **функциональное тестирование**, оно же **тестирование черного ящика** — тесты для него, а также используемые критерии полноты проведенного тестирования определяют на основе требований к функциональности. Частным случаем тестирования на соответствие является **аттестационное** или **квалификационное тестирование**, по результатам которого программная система получает (или не получает) официальный документ, подтверждающий ее соответствие определенным требованиям и стандартам.
2. **Структурное тестирование**, оно же **тестирование белого ящика** — тесты создаются на основе знаний о структуре самой системы и о том, как она работает. Критерии полноты основаны на проценте элементов кода, которые отработали в ходе выполнения тестов. Для оценки степени соответствия требованиям могут привлекаться дополнительные знания о прослеживании требований в определенные ограничения на значения внутренних данных системы (например, на значения параметров вызовов, результатов и локальных переменных).
3. **Тестирование, нацеленное на определенные ошибки**. Тесты для такого тестирования строятся так, чтобы гарантированно выявлять определенные виды ошибок. Полнота тестирования определяется на основе количества проверенных ситуаций по отношению к общему числу ситуаций, которые мы пытались достичь. К этому виду относится, например, **тестирование на отказ (smoke testing)**, в ходе которого просто пытаются вывести систему из строя, давая ей на вход как обычные данные, так и некорректные, с нарочно внесенными ошибками. Другим примером служит метод оценки полноты тестирования при помощи набора мутантов — программ, совпадающих с

тестируемой всюду, кроме нескольких мест, где специально внесены некоторые ошибки. Чем больше мутантов тесты находят, тем полнее проводимое с их помощью тестирование.

Еще одна классификация видов тестирования основана на том уровне, на который оно нацелено. Эти же разновидности тестирования можно связать с фазой жизненного цикла, на которой они выполняются.

1. **Модульное тестирование (unit testing)** предназначено для проверки правильности отдельных модулей, вне зависимости от их окружения. При этом проверяется, что, если модуль получает на вход данные, удовлетворяющие определенным критериям корректности, то и результаты его корректны. Для описания критериев корректности входных и выходных данных часто используют **программные контракты** — **предусловия**, описывающие для каждой операции, на каких входных данных она предназначена работать, **постусловия**, описывающие для каждой операции, как должны соотноситься входные данные с возвращаемыми ею результатами, и **инварианты**, определяющие критерии целостности внутренних данных модуля. Модульное тестирование является важной составной частью **отладочного тестирования**, выполняемого разработчиками для отладки написанного ими кода.

2. **Интеграционное тестирование (integration testing)** предназначено для проверки правильности взаимодействия модулей некоторого набора друг с другом. При этом проверяется, что в ходе совместной работы модули обмениваются данными и вызовами операций, не нарушая взаимных ограничений на такое взаимодействие, например, предусловий вызываемых операций. Интеграционное тестирование также используется при отладке, но на более позднем этапе разработки.

3. **Системное тестирование (system testing)** предназначено для проверки правильности работы системы в целом, ее способности правильно решать поставленные пользователями задачи в различных ситуациях. Системное тестирование тесно связано с **тестированием пользовательского интерфейса** (или через пользовательский интерфейс), проводимым при помощи имитации действий пользователей над элементами этого интерфейса. Частными случаями этого вида тестирования являются **тестирование графического пользовательского интерфейса** (Graphical User Interface, GUI) и **пользовательского интерфейса Интернет-приложений** (Web UI). Если интеграционное и модульное тестирование чаще всего проводят, воздействуя на компоненты системы при помощи операций предоставляемого ими программного интерфейса (Application Programming Interface, API), то на системном уровне без использования пользовательского интерфейса не обойтись, хотя тестирование через API в этом случае также часто возможно.

4. **Проверка свойств на моделях (model checking)** [10] — проверка соответствия ПО требованиям при помощи формализации проверяемых свойств, построения формальных моделей проверяемого ПО (чаще всего в виде автоматов различных видов) и автоматической проверки выполнения этих свойств на построенных моделях. Проверка свойств на моделях позволяет проверять достаточно сложные свойства автоматически, при минимальном участии человека. Однако она оставляет открытым вопрос о том, насколько выявленные свойства модели можно переносить на само ПО.



Рисунок Ошибка! Текст указанного стиля в документе отсутствует..4 – Схема процесса проверки свойств ПО на моделях

Обычно при помощи проверки свойств на моделях анализируют два вида свойств использованных при построении ПО алгоритмов. **Свойства безопасности (safety properties)** утверждают, что нечто нежелательное никогда не случится в ходе работы ПО. **Свойства живости (liveness properties)** утверждают, наоборот, что нечто желательное при любом развитии событий произойдет в ходе его работы.

Примером свойства первого типа служит отсутствие взаимных блокировок (deadlocks). Взаимная блокировка возникает, если каждый из группы параллельно работающих в проверяемом ПО процессов или потоков ожидает прибытия данных или снятия блокировки от одного из других, а тот не может этого сделать, потому что не может продолжить выполнение, ожидая того же от первого или от третьего процесса, и т.д.

Примером свойства живости служит гарантированная доставка сообщения, обеспечиваемая некоторыми протоколами — как бы ни развивались события, если сетевое соединение между машинами будет работать, посланное с одной стороны (процессом на первой машине) сообщение будет доставлено другой стороне (процессу на второй машине).

В классическом подходе к проверке на моделях проверяемые свойства формализуются в виде формул так называемых временных логик. Их общей чертой является наличие операторов «всегда в будущем» и «когда-то в будущем». Заметим, что второй оператор может быть выражен с помощью первого и отрицания — то, что некоторое свойство когда-то будет выполнено, эквивалентно тому, что отрицание этого свойства не будет выполнено всегда. Свойства безопасности легко записываются в виде «всегда будет выполнено отрицание нежелательного свойства», а свойства живости — в виде «когда-то обязательно будет выполнено желаемое».

Проверяемая программа в классическом подходе моделируется при помощи конечного автомата. Проверка, выполняемая автоматически, состоит в том, что для всех состояний этого автомата проверяется нужное свойство. Если оно оказывается выполненным, выдается сообщение об успешности проверки, если нет — выдается трасса, последовательность выполнения отдельных шагов программы, моделируемых переходами автомата, приводящая из начального состояния в такое, в котором нужное свойство нарушается. Эта трасса используется для анализа происходящего и исправления либо программы, либо модели, если ошибка находится в ней.

Основная проблема этого подхода — огромное количество состояний в моделях, достаточно хорошо отражающих поведение реальных программ. Для борьбы с комбинаторным взрывом состояний используются многочисленные методы оптимизации

представления автомата и оптимизации выделения и поиска существенных для выполнения проверяемого свойства состояний.

1.3 Ошибки в ПО

Ошибками в ПО, вообще говоря, являются все возможные несоответствия между демонстрируемыми характеристиками его качества и сформулированными или подразумеваемыми требованиями и ожиданиями пользователей.

В англоязычной литературе используется несколько терминов, часто одинаково переводящихся как «ошибка» на русский язык.

- **defect** — самое общее нарушение каких-либо требований или ожиданий, не обязательно проявляющееся вовне (к дефектам относятся и нарушения стандартов кодирования, недостаточная гибкость системы и пр.).
- **failure** — наблюдаемое нарушение требований, проявляющееся при каком-то реальном сценарии работы ПО. Это можно назвать проявлением ошибки.
- **fault** — ошибка в коде программы, вызывающая нарушения требований при работе (failures), то место, которое надо исправить. Хотя это понятие используется довольно часто, оно, вообще говоря, не вполне четкое, поскольку для устранения нарушения можно исправить программу в нескольких местах. Что именно надо исправлять, зависит от дополнительных условий, выполнение которых мы хотим при этом обеспечить, хотя в некоторых ситуациях наложение дополнительных ограничений не устраняет неоднозначность.
- **error** — используется в двух смыслах. Первый — это ошибка в ментальной модели программиста, ошибка в его рассуждениях о программе, которая заставляет его делать ошибки в коде (faults). Это, собственно, ошибка, которую сделал человек в своем понимании свойств программы. Второй смысл — это некорректные значения данных (выходных или внутренних), которые возникают при ошибках в работе программы.

Первое место в неформальном состязании за место «самой дорого обошедшейся ошибки в ПО» долгое время удерживала ошибка, приведшая к неудаче первого запуска ракеты Ариан-5 4 июня 1996 года, стоившая около \$500 М. После произошедшего 14 августа 2003 года обширного отключения электричества на северо-востоке Северной Америки, стоившего экономике США и Канады от 4 до 10 миллиардов долларов, это место закрепилось за вызвавшей его ошибкой в системе управления электростанцией. Широко известны также примеры ошибок в системах управления космическими аппаратами, приведшие к их потере или разрушению.

Стоит отметить, что в большинстве примеров ошибок, имевших тяжелые последствия, нельзя совершенно однозначно приписать всю вину за случившееся ровно одному недочету, одному месту в коде. Ошибки очень часто «охотятся стаями». К тяжелым последствиям приводят чаще всего ошибки системного характера. 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, поведение в рамках которых неоднозначно или недостаточно четко определяется требованиями (а иногда и вообще никак не определяется). Поэтому разработчики различных модулей ПО имеют возможность по-

разному интерпретировать те части требований, которые относятся непосредственно и 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.

Например, анализ катастрофы Ариан 5 показал следующее [13].

- Ариан 5 была способна летать при более высоких значениях ускорений и скоростей, чем это могла делать ракета предыдущей серии, Ариан 4. Однако большое количество процедур контроля и управления движением по траектории в коде управляющей системы было унаследовано от Ариан 4. Большинство таких процедур не были специально проверены на работоспособность в новой ситуации, как в силу большого размера кода, который надо было проанализировать, так и потому, что этот код раньше не вызывал проблем, а соотнести его со специфическими характеристиками полета ракет вовремя никто не сумел.
- В одной из таких процедур производилась обработка горизонтальной скорости ракеты. При выходе этой величины за границы, допустимые для Ариан 4, создавалась исключительная ситуация переполнения. Надо отметить, что обработка нескольких достаточно однородных величин производилась по-разному — семь переменных могли вызвать исключительную ситуацию этого вида, обработка четырех из них была защищена от этого, а три оставшихся, включая горизонтальную скорость, оставлены без защиты. Аргументом для этого послужило выдвинутое при разработке требование поддерживать загрузку процессора не выше 80%. «Нагружающие» процессор защитные действия для этих переменных не были использованы, поскольку предполагалось, что эти величины будут находиться в нужных пределах в силу физических ограничений на параметры движения ракеты. Обоснований для поддержки именно такой загрузки процессора и того, что отсутствие обработки переполнения выбранных величин будет способствовать этому, найдено не было.
- Когда такая ситуация действительно случилась, она не была обработана соответствующим образом, и в результате ею вынужден был заняться модуль, обеспечивающий отказоустойчивость программной системы в целом.
- Этот модуль, в силу отсутствия у него какой-либо возможности обрабатывать такие ошибки специальным образом, применил обычный прием — остановил процесс, в котором возникла ошибка, и запустил другой процесс с теми же исходными данными. Как легко догадаться, эта же ошибка повторилась и во втором процессе.
- Не в силах получить какие-либо осмысленные данные о текущем состоянии полета, система управления использовала ранее полученные, которые уже не соответствовали действительности. При этом были ошибочно включены боковые двигатели «для корректировки траектории», ракета начала болтаться, угол между нею и траекторией движения стал увеличиваться и достиг 20 градусов. В результате она стала испытывать чрезмерные аэродинамические нагрузки и была автоматически уничтожена.