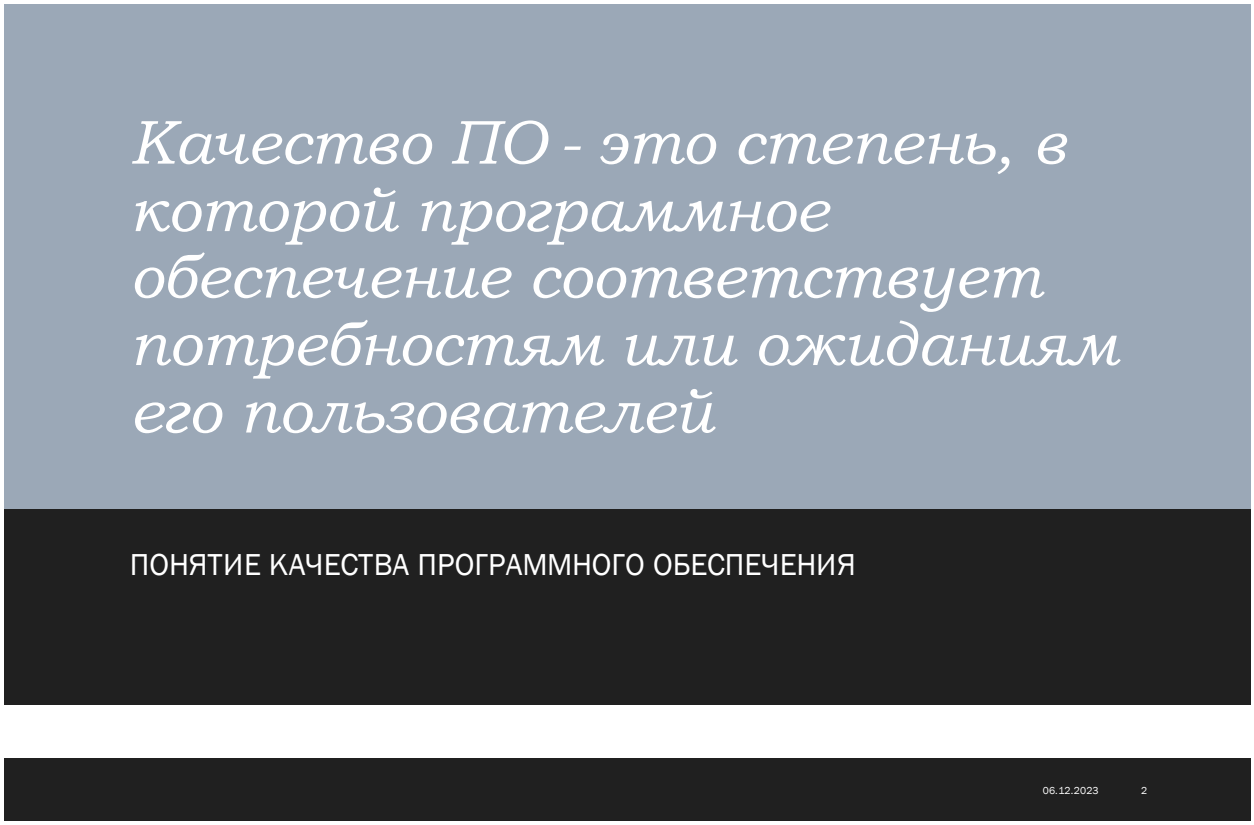
Основы стандартизации и сертификации программных продуктов

Лекция 3
Стандарты качества информационных систем

Качество ПО - это степень, в которой программное обеспечение соответствует потребностям или ожиданиям его пользователей.



Понятие качества ПО

В настоящее время уже нельзя представить современную работу в какой-либо отрасли жизни без компьютерных технологий и программного обеспечения. Следовательно, все большее значение приобретает положительное решение такого вопроса, как обеспечение надлежащего качества программного обеспечения (ПО).

Качество ПО представляет собой степень, в которой программное обеспечение соответствует ожиданиям его пользователей. Кроме того, качеством ПО называют способность программного продукта при заданных условиях удовлетворять заданные или гипотетические потребности.

Качеством ПО обычно представлен весь объем признаков и характеристик программ.

На данный момент общепринято, что обязательными атрибутами качества ПО являются стандарты и критерии, ориентация на которые позволяет заинтересованным субъектам поддерживать уровень качества ПО в заданном диапазоне.

Стандарты качества ПО

Стандарты управления качеством промышленной продукции

Международные стандарты серии ISO 9000 разработаны для управления качеством продукции, их дополняют стандарты серии ISO14000, отражающие экологические требования к производству промышленной продукции.

Очевидно, что управление качеством тесно связано с его контролем. Контроль качества традиционно основан на измерении показателей качества продукции на специальных технологических операциях контроля и выбраковке негодных изделий. Однако есть и другой подход к управлению качеством, который основан на контроле качественных показателей не самих изделий, а проектных процедур и технологических процессов, используемых при создании этих изделий.

Такой подход во многих случаях более эффективен. Он требует меньше затрат, поскольку позволяет обойтись без стопроцентного контроля продукции и благодаря предупреждению появления брака снижает производственные издержки. Именно этот подход положен в основу стандартов ISO 9000, принятых ISO в 1987 г. и проходящих корректировку приблизительно каждые пять лет.

Таким образом, методической основой для управления качеством являются международные стандарты серии ISO 9000. Они определяют и регламентируют инвариантные вопросы создания, развития, применения и сертификации систем качества в промышленности. В них устанавливается форма требований к системе качества в целях демонстрации поставщиком своих возможностей и оценки этих возможностей внешними сторонами.

Основной причиной появления стандартов ISO 9000 была потребность в общем для всех участников международного рынка базисе для контроля и управления качеством товаров.

Американское общество контроля качества определило цели ISO 9000 как помощь в развитии международного обмена товарами и услугами и кооперации в сфере интеллектуальной, научной, технологической и деловой активности.

В стандартах ISO 9000 используется определение качества из стандарта ISO 8402: " Качество - совокупность характеристик продукта, относящихся к его способности удовлетворять установленные или предполагаемые потребности". Аналогичное определение содержится в ГОСТ 15467-79: " Качество продукции - это совокупность свойств продукции, обусловливающих ее пригодность удовлетворять определенные потребности в соответствии с ее назначением". В ISO 9000 вводится понятие системы качества (QS - Quality System), под которой понимают документальную систему с руководствами и описаниями процедур достижения качества. Другими словами, система качества есть совокупность организационной структуры, ответственности, процедур, процессов и ресурсов, обеспечивающая осуществление общего руководства качеством. Система качества обычно представляет собой совокупность трех слоев документов:

- описание политики управления для каждого системного элемента;
- описание процедур управления качеством (что, где, кем и когда должно быть сделано);
- тесты, планы, инструкции и т. п.

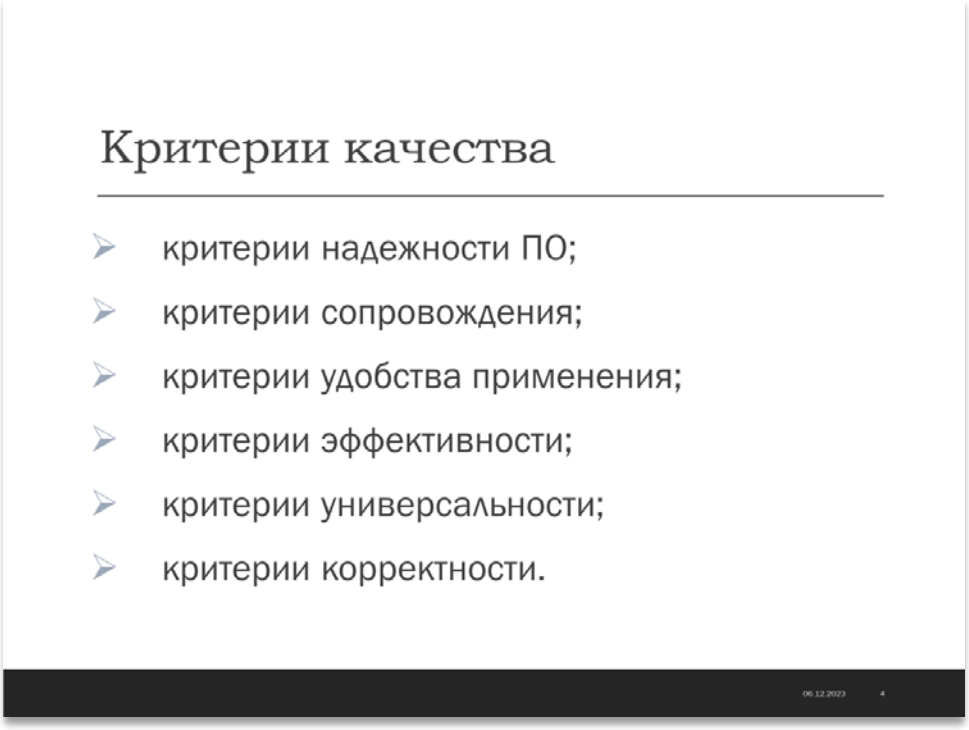
Сертификация предприятий по стандартам ISO 9001выполняется органом по сертификации.

Наличие сертификата качества - одно из важных условий для успеха коммерческой деятельности предприятий.

ГОСТ Р ИСО 9004-2019. Менеджмент качества. Качество организации. Руководство по достижению устойчивого успеха организации

ГОСТ Р ИСО 9001-2015 "Системы качества. Модель обеспечения качества при проектировании, разработке, производстве, монтаже и обслуживании";

Критерии качества ПО



Качество ПО может быть охарактеризовано при помощи нескольких критериев, которые представлены определенными показателями. Чаще всего используется следующая номенклатура критериев качества ПО:

- критерии надежности ПО;
- критерии сопровождения;
- критерии удобства применения;
- критерии эффективности;
- критерии универсальности;
- критерии корректности.

Критерием надежности ПО является устойчивость функционирования. Она заключается в способности ПО продолжить свою работу после возникновения отклонений, которые могут быть вызваны техническими сбоями, ошибками во входных данных или в обслуживании.

Кроме надёжности рассматривают работоспособность ПО, т. е. функционирование программы в заданных режимах и объемах обрабатываемой информации в соответствии с программными документами при отсутствии сбоев технических средств.

Сопровождение ПО определяется четырьмя критериями:

- во-первых, структурность, которая означает организацию в единое целое всех взаимосвязанных частей программы с использованием трех основных логических структур: «повторение», «выбор», «последовательность»;

- во-вторых, простота конструкции, в соответствии с которой модульная структура программы выстраивается наиболее рациональным с точки зрения восприятия и понимания образом;
- в-третьих, наглядность, когда исходные модули ПО предстают в наиболее воспринимаемом виде;
- в-четвертых, повторяемость, которая означает степень использования типовых проектных решений или компонентов, образующих ПО.

Удобство применения ПО характеризуется через легкость его освоения. Она соответствует представлению программных документов и программы в том виде, который способствует пониманию логики функционирования программы в целом и ее частей по отдельности.

Доступность эксплуатационных программных документов говорит о понятности, наглядности и полноте описания процесса взаимодействия пользователя с программой в эксплуатационных программных документах. Эксплуатация и обслуживание могут быть удобными, если процесс обработки данных и форм представления результатов соответствует характеру решаемых задач.

Главными критериями эффективности ПО в настоящее время считаются уровень автоматизации функций обработки данных, временная эффективность (т.е. способность ПО осуществлять заданные функции в течение некоторого времени по заданным требованиям) и ресурсоемкость (минимальное использование вычислительных и трудовых ресурсов для обеспечения функционирования ПО).

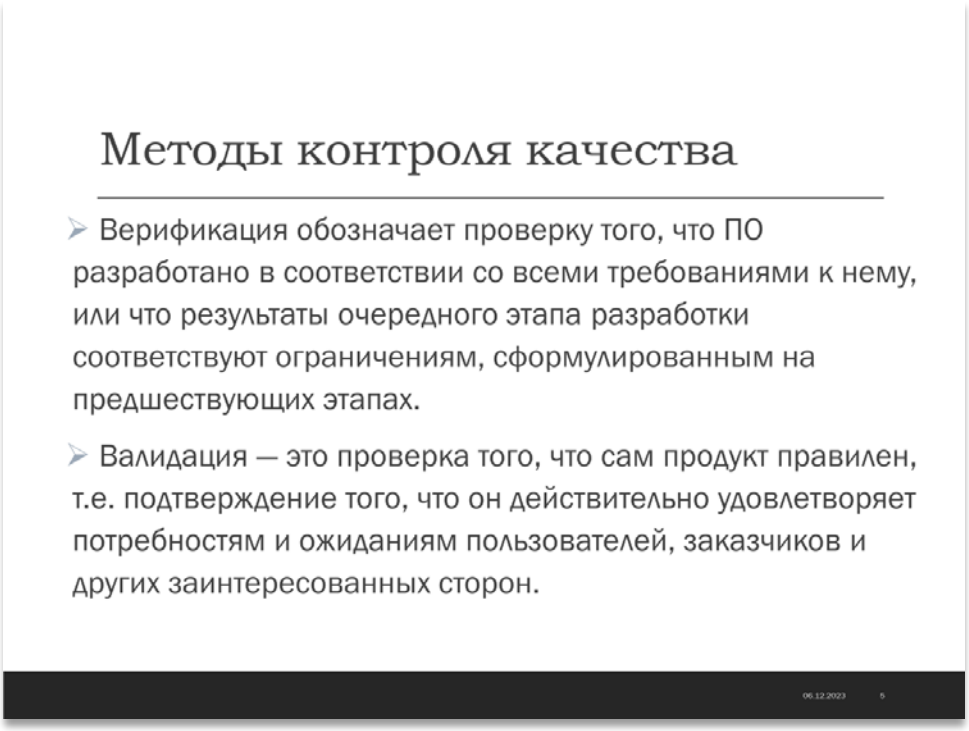
Критерии универсальности ПО заключаются в следующем:

- гибкость - использование ПО в различных областях применения;
- мобильность - использование ПО без необходимости дополнительного привлечения ресурсов;
- модифицируемость - возможность внесения доработок в ПО во время его эксплуатации.

Критерий корректности подразумевает логическую корректность. Она свидетельствует о функциональном и программном соответствии процесса обработки данных при выполнении задания общесистемным требованиям. Заданные функции ПО должны быть полностью реализованы и достаточно описаны в программной документации.

Согласованность ПО выражена через однозначное, непротиворечивое описание и использование тождественных элементов ПО в различных частях программных документов и текста программы.

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 эти вопросы можно получить с помощью процессов верификации и валидации.

Верификация обозначает проверку того, что ПО разработано в соответствии со всеми требованиями к нему, или что результаты очередного этапа разработки соответствуют ограничениям, сформулированным на предшествующих этапах.

Валидация — это проверка того, что сам продукт правилен, т.е. подтверждение того, что он действительно удовлетворяет потребностям и ожиданиям пользователей, заказчиков и других заинтересованных сторон.

Эффективность верификации и валидации, как и эффективность разработки ПО в целом, зависит от полноты 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. Мы будем рассматривать подобные методы на протяжении всего курса.

Методы контроля качества позволяют 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:

- Методы и техники, связанные с 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.

- Методы и техники определения 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), используемое для оценки качества принимаемых решений.

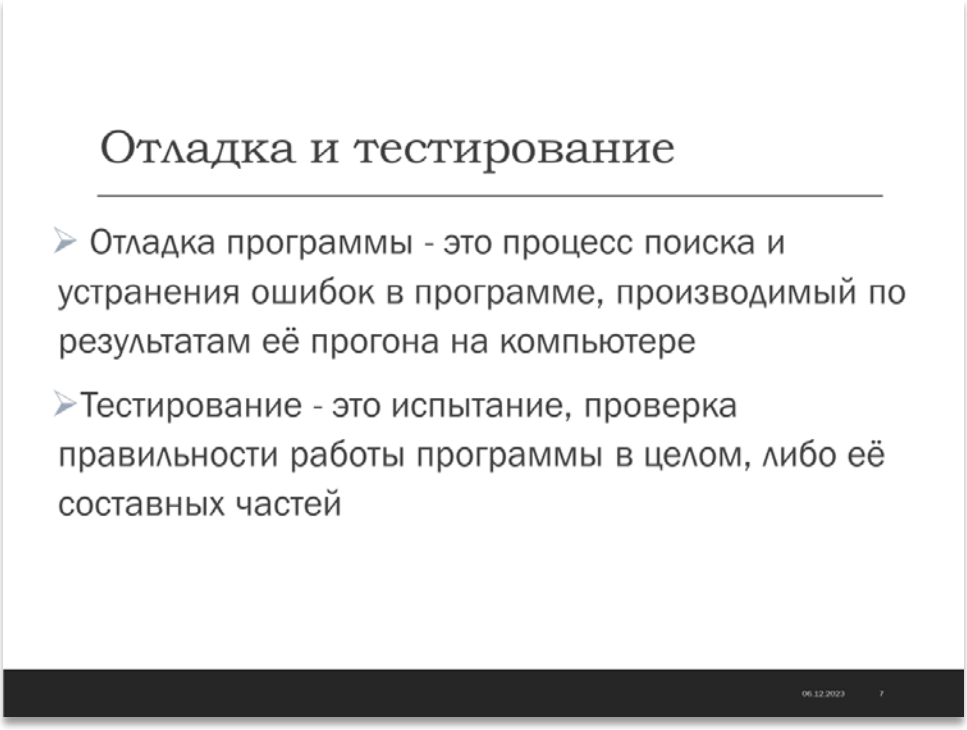
- Методы и техники, нацеленные на выявление нарушений формализованных правил построения исходного кода ПО, проектных моделей и документации.

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 выполнении, методы проверки документации на согласованность и соответствие стандартам.

- Методы и техники обычного или формализованного анализа 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.

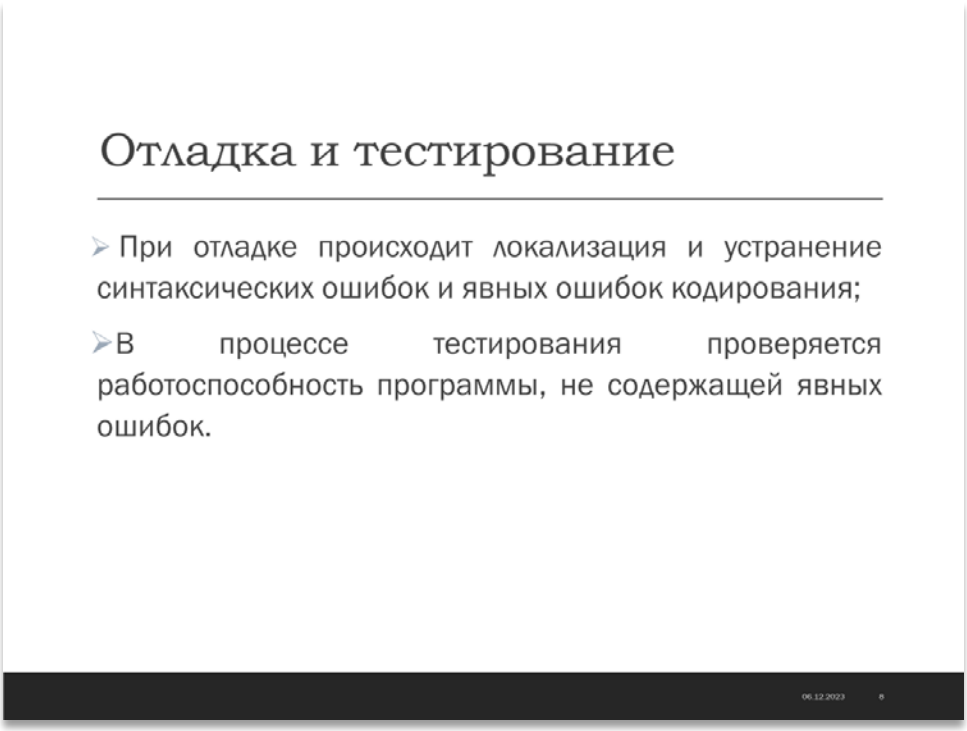
Для чего нужны отладка и тестирование?



Отладка программы — это процесс поиска и устранения ошибок в программе, производимый по результатам её прогона на компьютере.

Тестирование — это испытание, проверка правильности работы программы в целом, либо её составных частей.

Отладка и тестирование (англ. test - испытание) — это два четко различимых и непохожих друг на друга этапа:



- при отладке происходит локализация и устранение синтаксических ошибок и явных ошибок кодирования;

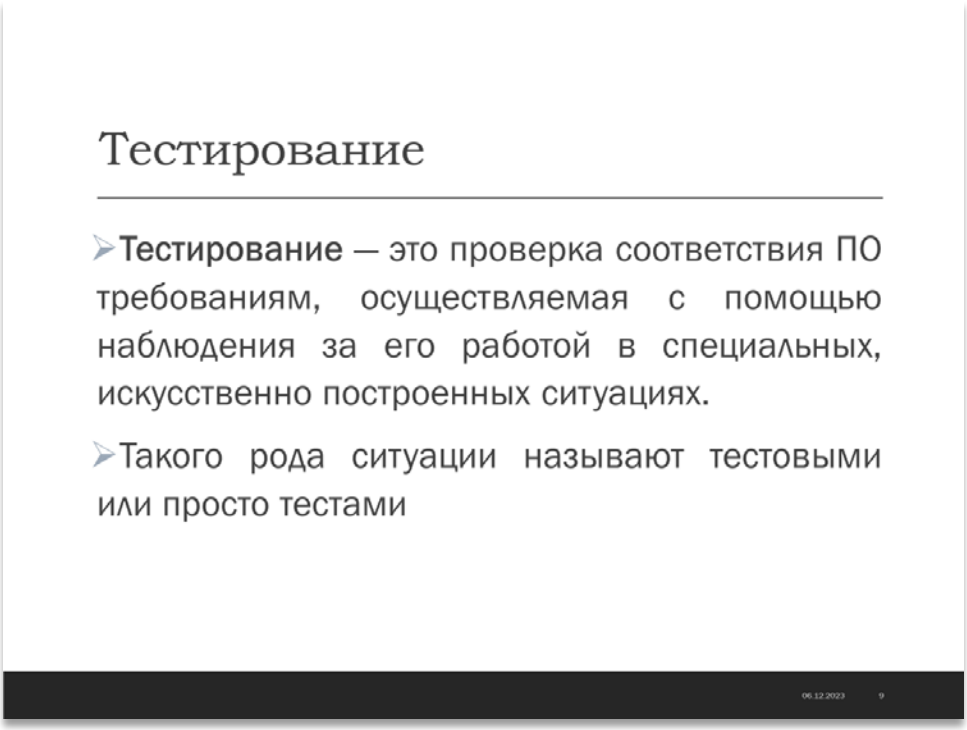
- в процессе же тестирования проверяется работоспособность программы, не содержащей явных ошибок.

Тестирование устанавливает факт наличия ошибок, а отладка выясняет ее причину.

Английский термин debugging ("отладка") буквально означает "вылавливание жучков". Термин появился в 1945 г., когда один из первых компьютеров - "Марк-1" прекратил работу из-за того, что в его электрические цепи попал мотылек и заблокировал своими останками одно из тысяч реле машины.

Далее мы несколько подробнее рассмотрим тестирование и проверку на моделях как примеры методов контроля качества.

Тестирование



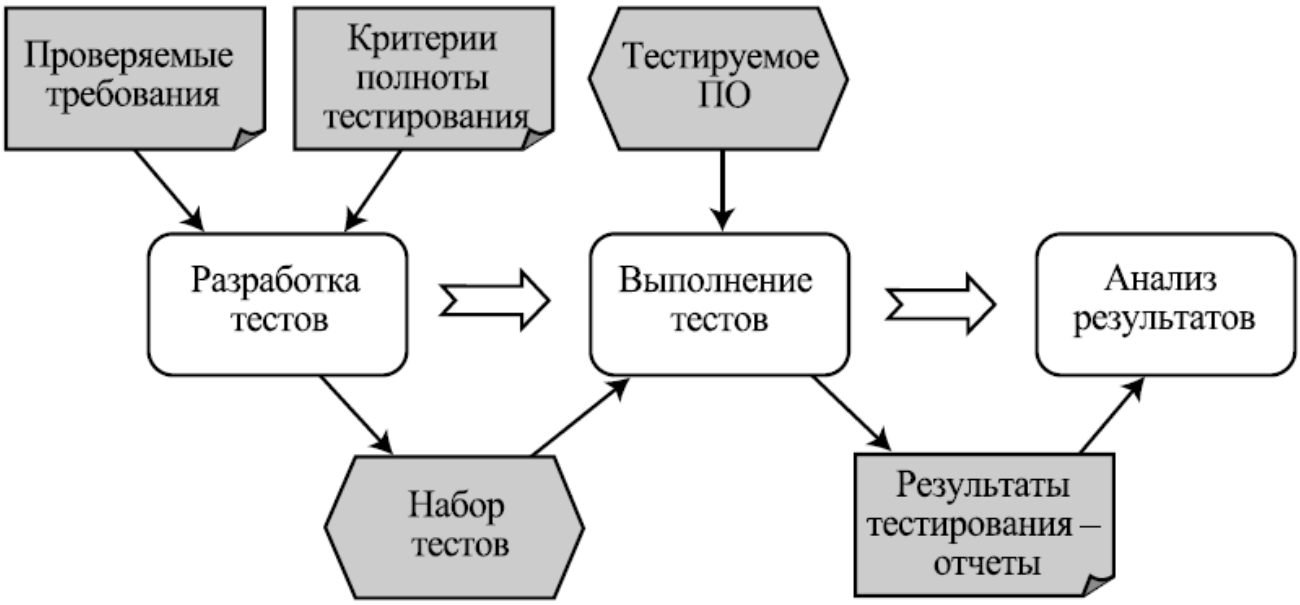
Тестирование — это проверка соответствия ПО требованиям, осуществляемая с помощью наблюдения за его работой в специальных, искусственно построенных ситуациях. Такого рода ситуации называют тестовыми или просто тестами.

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) заметил, что тестирование

позволяет точно определить, что в программе есть ошибка, но 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. Этот критерий может утверждать, что все равно в какой из ситуаций, A или B, проверять правильность работы ПО, или, если программа правильно работает в ситуации A, то, скорее всего, в ситуации B 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, случившихся во время тестирования, — достигнутым тестовым покрытием.

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



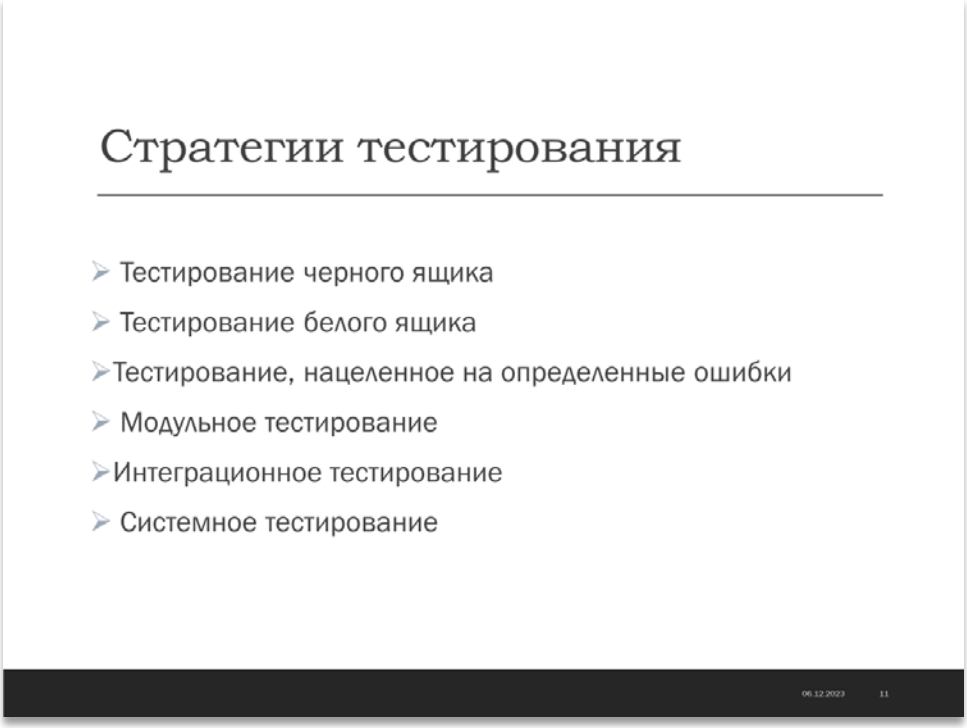
Тестировать можно соблюдение любых требований, соответствие которым выявляется во время работы ПО.

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 — тестирование 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 — при большом количестве пользователей, интенсивном обмене данными с другими системами, большом объеме передаваемых или используемых данных и пр.



На основе исходных данных, используемых для построения тестов, тестирование делят на следующие виды, ещё их принято называть стратегиями:

- Тестирование черного ящика, нацеленное на проверку требований. Тесты для него и критерий полноты тестирования строятся на основе требований и ограничений, четко зафиксированных в спецификациях, стандартах, внутренних нормативных документах. Часто такое тестирование называется тестированием на соответствие (conformance testing). Частным случаем его является функциональное тестирование — тесты для него, а также используемые критерии полноты проведенного тестирования определяют на основе требований к функциональности.

Еще одним примером 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.

- Тестирование белого ящика, оно же структурное тестирование — 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 связи требований с определенными ограничениями на значения внутренних данных системы (например, на значения параметров вызовов, результатов и локальных переменных).

- 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 проверить. 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 не проходит успешно через данный набор тестов, тем полнее и качественнее проводимое с его 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.

- 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.

- 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.

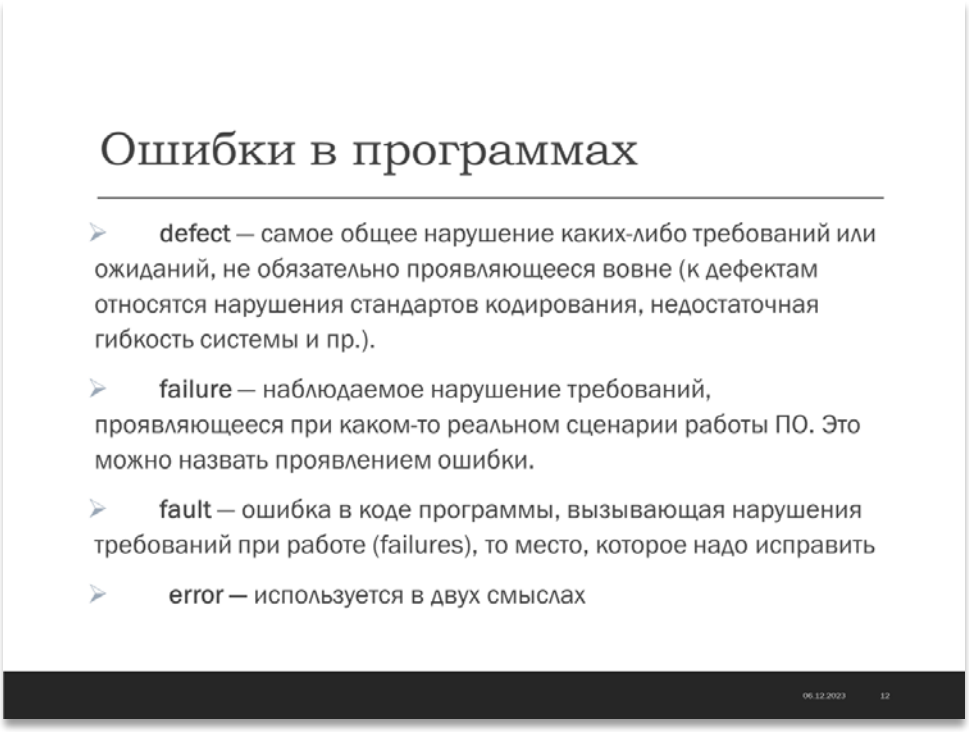
- Системное тестирование (system testing) предназначено для проверки правильности работы системы в целом, ее способности правильно решать поставленные пользователями задачи в различных ситуациях.

Системное тестирование выполняется через внешние интерфейсы ПО и 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 Web-приложений (WebUI).

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 вполне возможно.

Основной недостаток тестирования состоит в том, что проводить его можно, только когда проверяемый элемент программы уже разработан. Снизить влияние этого ограничения можно, подготавливая тесты (а это — наиболее трудоемкая часть тестирования) на основе требований заранее, когда исходного кода еще нет.

Ошибки в программах



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 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.

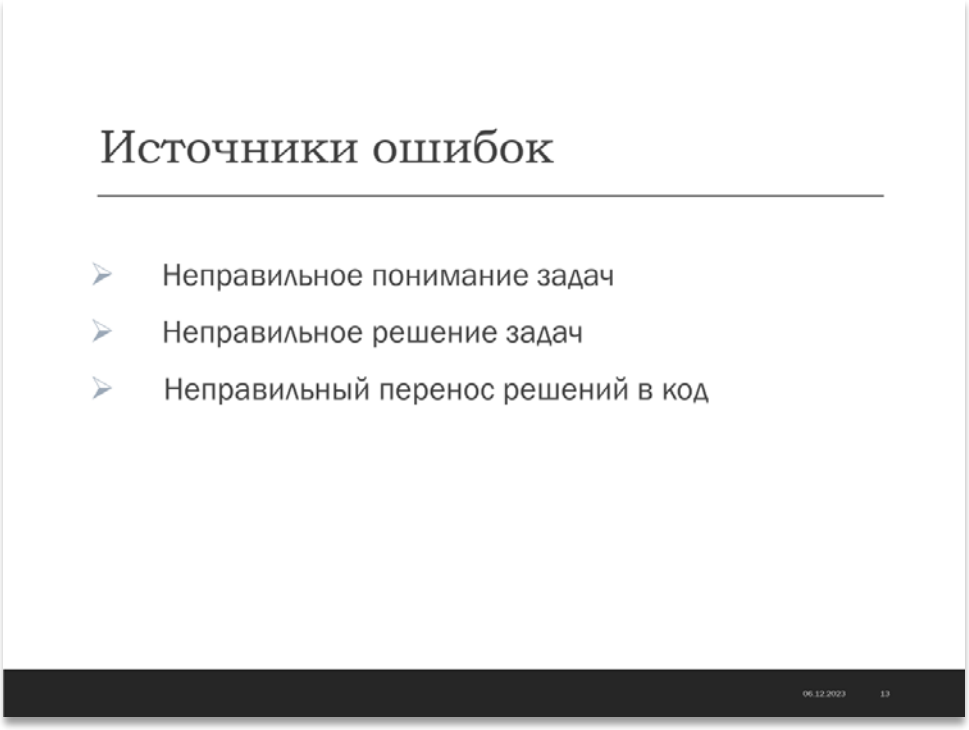
Хотя это понятие используется довольно часто, оно, вообще говоря, не вполне четкое, поскольку для устранения нарушения можно исправить программу в нескольких местах. Что именно надо исправлять, зависит от дополнительных условий, выполнение которых мы хотим при этом обеспечить, хотя в некоторых ситуациях 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.

Эти понятия некоторым образом связаны с основными источниками ошибок. Поскольку при разработке программ необходимо сначала понять задачу, затем придумать ее решение и закодировать его в виде программы, то, соответственно, основных источников ошибок три:



- Неправильное понимание задач.

Очень часто люди не понимают, что им пытаются сказать другие. Так же и разработчики ПО не всегда понимают, что именно нужно сделать. Другим источником непонимания служит отсутствие его у самих пользователей и заказчиков — достаточно часто они могут просить сделать несколько не то, что им действительно нужно.

Для предотвращения неправильного понимания задач программной системы служит анализ предметной области.

- Неправильное решение задач.

Зачастую, даже правильно поняв, что именно нужно сделать, разработчики выбирают неправильный подход к тому, как это делать. Выбираемые решения могут обеспечивать лишь некоторые из требуемых свойств, они могут хорошо подходить для данной задачи в теории, но плохо работать на практике, в конкретных обстоятельствах, в которых должно будет работать ПО.

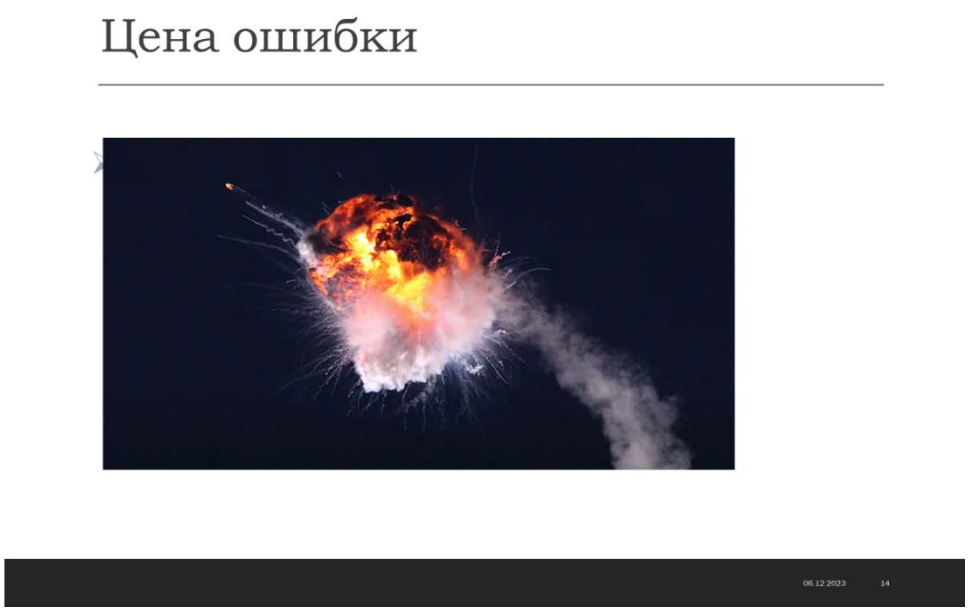
Помочь в выборе правильного решения может сопоставление альтернативных решений и тщательный анализ их на предмет соответствия всем требованиям, поддержание постоянной связи с пользователями и заказчиками, предоставление им необходимой информации о выбранных решениях, демонстрация прототипов, анализ пригодности выбираемых решений для работы 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 воплощении этих решений. 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 принятое решение.

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*.



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 (см. [13]), стоившая около $500 000 000. 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 [14], это место можно отдать спровоцировавшей его ошибке в системе управления электростанцией. Широко известны также примеры ошибок в системах управления космическими аппаратами, приведшие к их потере или разрушению. Менее известны, но не менее трагичны, ошибки в ПО, управлявшем медицинским и военным оборудованием, некоторые из которых привели к гибели людей.

Стоит отметить, что в большинстве примеров ошибок, имевших тяжелые последствия, нельзя однозначно приписать всю вину за случившееся ровно одному недочету, одному месту в коде. Ошибки очень часто "охотятся стаями". К тяжелым последствиям чаще всего приводят ошибки системного характера, затрагивающие многие аспекты и элементы системы в целом. 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 — признак 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.

Например, анализ катастрофы Ариан-5 показал следующее.

- 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 данного 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.

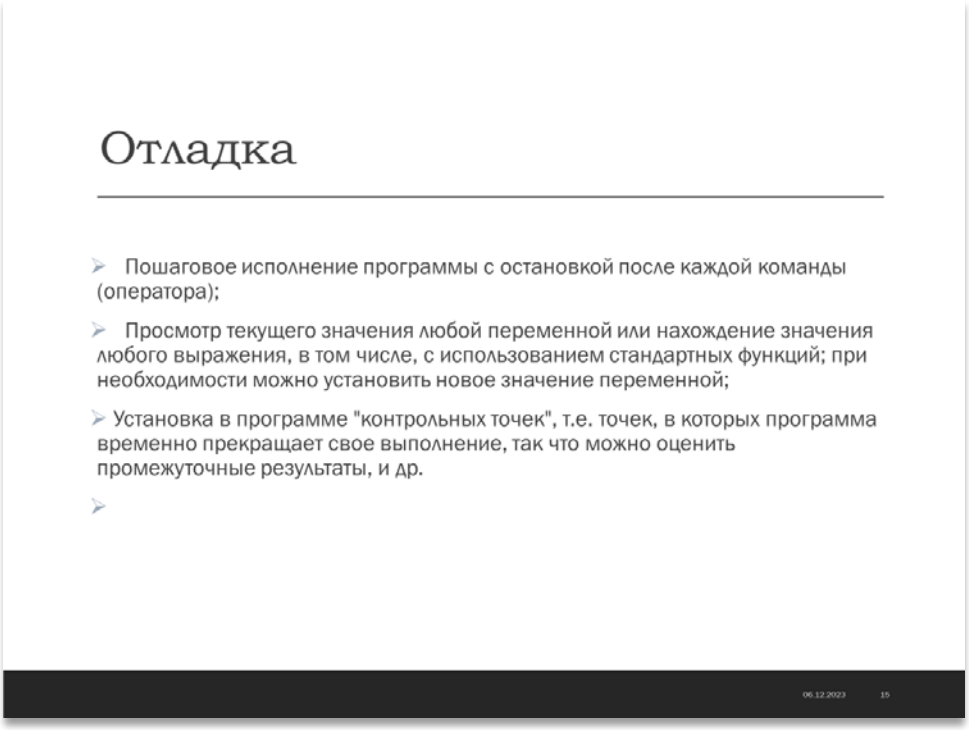
- Когда такая ситуация действительно случилась, т.е. горизонтальная скорость ракеты превысила определенное значение, она не была обработана соответствующим образом, и в результате ею вынужден был

заняться модуль, обеспечивающим 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.

Отладка

В чем заключается отладка? В современных программных системах (Turbo Basic, Turbo Pascal, Turbo C и др.) отладка осуществляется часто с использованием специальных программных средств, называемых отладчиками. Эти средства позволяют исследовать внутреннее поведение программы.



Программа-отладчик обычно обеспечивает следующие возможности:

- пошаговое исполнение программы с остановкой после каждой команды (оператора);
- просмотр текущего значения любой переменной или нахождение значения любого выражения, в том числе, с использованием стандартных функций; при необходимости можно установить новое значение переменной;
- установку в программе "контрольных точек", т.е. точек, в которых программа временно прекращает свое выполнение, так что можно оценить промежуточные результаты, и др.

При отладке программ важно помнить следующее:

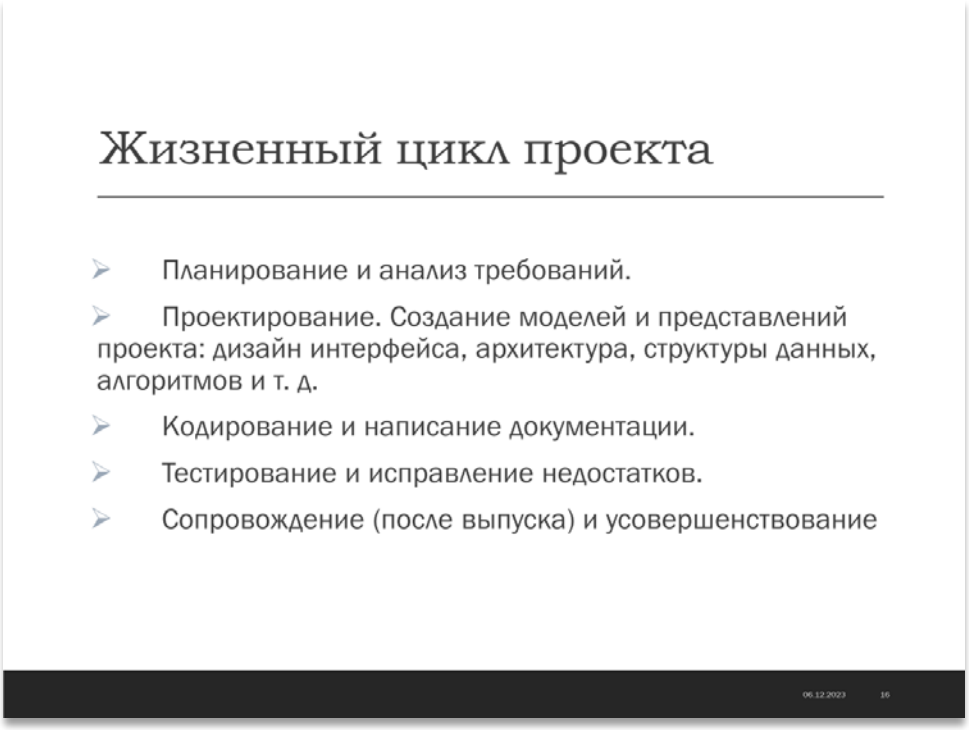
в начале процесса отладки надо использовать простые тестовые данные;

возникающие затруднения следует четко разделять и устранять строго поочередно;

не нужно считать причиной ошибок машину, так как современные машины и трансляторы обладают чрезвычайно высокой надежностью.

Тестирование на этапах жизненного цикла проекта

Рассмотрим классический жизненный цикл проекта:



· Планирование и анализ требований.
· Проектирование. Создание моделей и представлений проекта: дизайн интерфейса, архитектура, структуры данных, алгоритмов и т. д.
· Кодирование и написание документации.

· Тестирование и исправление недостатков.
· Сопровождение (после выпуска) и усовершенствование.

На каждом этапе жизненного цикла должны выполняться верификация и валидация проекта.

Верификация (Verification) — это процесс оценки системы или её компонентов с целью определения удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа.

Валидация (Validation) — это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе. Тестирование как инструмент верификации и валидации является постоянным процессом и проводится на всех этапах жизненного цикла проекта.

В ходе тестирования на текущем этапе необходимо достичь следующих целей:

· Повысить вероятность того, что разрабатываемое ПО будет работать правильно при любых обстоятельствах.
· Повысить вероятность того, что разрабатываемое ПО будет соответствовать всем описанным требованиям.
· Предоставить актуальную информацию о состоянии продукта на данный момент.

1.1. Планирование и анализ требований

На этапе планирования выполняется организация работы, согласование тематики проекта с заказчиком, выработка идей проекта, сбор и анализ требований, определение функциональных характеристик продукта. Результатом этапа являются план реализации проекта и техническое задание.

В ходе планирования и анализа требований “тестируются” идеи. На данном этапе к анализу могут быть привлечены специалисты и руководители разных направлений, в том числе отдела маркетинга, отдела разработки и др. Специалисты по тестированию в этой работе практически никогда не участвуют.

Специалисты по тестированию знакомятся с проектными документами и собирают различного рода информацию, которая может оказать помощь в оценке результатов этапа и дальнейшем планировании тестирования. Для сбора информации используются следующие способы:

· сравнительный анализ: выполняется сравнение целей и задач проекта с аналогичными или похожими проектами для установления взаимосвязей и выявления потенциальных проблем;
· дискуссионные группы: выполняется анализ и обсуждение предлагаемых идей с целью уточнения и детализации;
· обследование объекта: выполняется изучение бизнес-процессов с целью выявления скрытой информации.

Логическим завершением каждой из этой процедур является пересмотр существующих планов.

В ходе анализа требований группа тестирования должна достичь следующих целей.

· Адекватность требований. В ходе анализа требований может выясниться, что заказчик хочет получить совершенно другой продукт. Группа тестирования должна удостовериться, что заявленные требования соответствуют ожиданиям заказчика.

· Полнота требований. Выяснение дополнительной информации у заказчика на последующих этапах проекта может привести к дополнительным задержкам. Кроме этого, не выясненные детали могут привести к кардинальному усложнению проекта.

· Совместимость требований. Функции разрабатываемого программного обеспечения могут оказаться несовместимыми по логическим (противоречивость функций) или психологическим (концептуальные различия) компонентам.

· Выполнимость требований. Группа тестирования должна выяснить возможность нормальной эксплуатации продукта. Заявленные требования могут подразумевать более высокие требования к аппаратному обеспечению, памяти, пропускной способности каналов связи и т. д.

· Разумность требований. Качество продукта (его производительность, надежность и нетребовательность к ресурсам) и стоимость и сроки его разработки являются двумя противоречивыми требованиями. Поэтому расстановка приоритетов при планировании является ключевой составляющей конечного успешного продукта.

· Подверженность тестированию. Позволяет определить сответствие документации и требований.

В ходе проектирования командой разработчиков создается набор моделей — представлений будущей программной системы. К ключевым моделям можно отнести следующие:

• внешний дизайн (представление системы с точки зрения конечного пользователя);

• проектирование программной архитектуры (декомпозиция, подсистемы, модули, интерфейсы);

• проектирование организации данных (потоки данных, преобразования, представления);

• описания алгоритмов (параметры, действия, результат);

• прототипы.

Группа проектирования более активно участвует в проекте и занимается проверкой проектных идей. Как правило, анализ и обсуждение проектных решений выполняется на совещаниях аналитиков. В результате формируются новые проектные решения или модифицируются текущие.

• Эффективность проекта. Насколько проект отвечает задаче эффективного создания тестируемого продукта.
• Соответствие требованиям. Все требования, выявленные на этапе планирования, формализованы в проекте.
• Полнота проекта. Насколько подробно проект описывает модули, данные, передачу данных между модулями, реализацию модулей.
• Реалистичность проекта. Насколько проект позволяет удо-влетворить системные требования и ресурсы (как аппаратные, так и программные), соответствует ли скорость обработки запросов ожиданиям пользователей, насколько удачно выбор средств разработки поможет в создании продукта и др.
• Поддержка сопровождения. Как подробно описаны ситуации возникновения ошибок.

1.3. **Кодирование и написание документации**

В ходе выполнения этапа разработчики создают код и программную документацию.

Задачей группы тестирования является разработка тестов. Тесты создаются на основе внутренней структуры кода или алгоритма (белый ящик, White box testing) или функциональности объекта тестирования (черный ящик, Black box testing). Разработка тестов белым ящиком позволяет:

- проверить логику работы кода;
- выполнить полный охват кода;
- проверить потоки данных;
- отследить целостность данных;
- проверить внутренние граничные точки.

Разработка тестов черным ящиком позволяет:

- проверить работу сложных объектов;
- проверить работу на некорректных данных;
- тестировать с точки зрения пользователя;
- создавать тесты параллельно с кодом.

По аналогии с проектированием тестирование большого сложного объекта (целостное) принесет меньше информации, чем тестирование составляющих объект модулей (модульное) и их взаимосвязей (ин-теграционное). С другой стороны, модульное тестирование требует дополнительных затрат на создание условий запуска и имитацию дея-тельности смежных модулей. Однако затраты на создание окружения тестируемого модуля, как правило, однократные, а окружение может быть использовано как для автоматизации тестирования, так и при сопровождении проекта, например, при демонстрации работы компо-нент.

Таким образом, задача группы тестирования заключается в определении объектов тестирования, их взаимосвязей, подготовке окружения и набора тестов.

Для оценки результатов работы группы тестирования используют метрики покрытия (Coverage criteria) или полноты. Метрики покрытия позволяют оценить охват объекта тестирования тестами и выявить слабые места, где покрытие тестами минимально. Для оценки покрытия можно воспользоваться следующими метриками.

- Критерий охвата функций (Function coverage) — каждая функция вызывается хотя бы раз.
- Критерий охвата строк (Statement coverage) — самый слабый, каждая строка должна выполниться.
- Критерий охвата ветвлений (Decision / Branch coverage) — более основательный, каждое ветвление проверяется по всем направлениям.
- Критерий охвата условий (Condition coverage) — более стро¬гий, проверка всех составляющих логического условия (каждое атомарное булево выражение приняло значения и «истина», и «ложь»).
- Критерий охвата параметров (Parameter Value coverage) — проверка всех значений параметров метода.
- Критерий охвата путей (Path coverage) — все возможные пути в коде были пройдены.
- Критерий охвата циклов (Loop coverage) — все циклы исполнялись 0, 1, ..., N раз.