НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ УКРАИНЫ « КПИ»

МЕТОДЫ ВЕРИФИКАЦИИ И ТЕСТИРОВАНИЯ ПРОГРАММ И СИСТЕМ

План

1. Верификация и аттестация программ

2. Методы верификации объектно–ориентированных программ

3. Методы тестирования программ

Верификация и аттестация программ

Верификация и аттестация (валидация) – это методы, которые обеспечивают соответственно проверку и анализ правильности выполнения заданных функций и соответствия ПО требованиям заказчика, а также заданным спецификациям ПО. Эти методы обозначены в стандарте ISO/IEC 12207 как самостоятельные процессы ЖЦ и используются, начиная от этапа анализа требований и кончая проверкой правильности функционирования программного кода на заключительном этапе – тестировании.

Верификация – это проверка того, правильно ли система работает в соответствии с ее спецификацией и заданными требованиями заказчика, Этот процесс ЖЦ стандарта ISO/IEC 12207 позволяет сделать заключение о корректности сделанной системы.

Валидация *является методом проверки соответствия спроектированного ПО*
требованиям и потребностям заказчика и предполагает выполнение на этапах ЖЦ
разного рода действий для получения корректных программ и систем:
– планирование процедур проверки и контроля проектных решений с помощью
методик и просмотра хода разработки;
– повышения уровня автоматизации проектирования программ с использованием –
CASE–систем ;
– проверка правильности функционирования программ с помощью методик тестирования на наборах целевых тестов;
– структурирование системы на модули, их спецификации, реализация и использование их как повторных компонентов (reuse);
– адаптация продукта к условиям использования;
– управления проектом.

Основными особенностями методов верификации и валидации

является проверка полноты, непротиворечивости и однозначности спецификаций требований к созданному ПО.

Верификация и валидация предполагают планирование этих процессов в целях распределения ресурсов и сосредоточения проверки на наиболее критичных элементах проекта, а именно:

– компонентов системы;

– интерфейса компонентов системы (программные, технические и информационные) и взаимодействий объектов (протоколов и сообщений) для функционирования в современных распределенных средах;

– средств доступа к БД и файлам, которые обеспечивают защиту от несанкционированного доступа к ним разных пользователей;

– документация на ПО;

– тестов и тестовых процедур;

– специальных средств защиты информации в системе.

По окончании проектирования приведенных элементов соответственно проводится:

– проверка правильности перевода отдельных компонентов в выходной код, а также описаний их интерфейсов, трассировки этих компонентов в соответствии с требованиями заказчика к функциям системы;

– анализ способов доступа к файлам или БД соответственно требований, принципов передачи данных и процедур манипулирования данными;

– проверка средств защиты на удовлетворение требованиям заказчика и правильности реализации.

По завершению процессов верификации и валидации создается комплект материалов отображающий правильность формирования требований, спецификация элементов системы, результатов проведения инспекций и тестирования программ.

Методы верификации объектно–ориентированных программ

Верификация таких программ имеет свою специфику и ее можно рассматривать исходя из композиционного метода, применяя к ним известные методы доказательства правильности композиционных программ.

Выделим несколько этапов для верификации исходного проектирования объектных моделей и программ:

1. Верификация базовых (простых) объектов сводится к верификации структуры, где атрибуты объектов являются данными структуры, а внутренние операции объекта — функциями над этими данными.

2. Верификация объектов, построенных с помощью наследования, агрегации или инкапсуляции, осуществляется исходя из следующих предположений:

– базовые объекты считаются проверенными, если их операции (функции) приняты за теоремы;

– доказательство объектов сводится к этим теоремам;

– доказывается, что все операции, которые применяются над подобъєктами, не выводят
их из множества состояний, для которых они верифицированы.

3. Верификация интерфейсов объектов сводится к доказательству:

– правильности данного интерфейса;

– достаточности параметров интерфейса.

Метод верификации ООП на основе композиционного подхода можно использовать как “вниз” так и “вверх”. Сначала доказывается правильность построенной объектной модели для некоторой предметной области. Верификация ОМ требует реализации на этапах ЖЦ следующих вопросы:

– удаление лишних атрибутов объектов и их интерфейсов в ОМ, внесение необходимых изменений и повторное доказательство правильности объекта;

– выбор типа множества для атрибутов объекта и проверка реализации операций и множество значений этого типа.

Правильность спецификаций любого объекта программной системы доказывается независимо от правильности смежных объектов и верификации их интерфейсов. Это является заключительной проверкой ОМ.

Чем отличается валидация от верификации

Стандарт ИСО 9000:2000 определяет эти термины следующим образом:

«**Верификация** — подтверждение на основе представления объективных свидетельств того, что установленные требования были выполнены».

«**Валидация** — подтверждение на основе представления объективных свидетельств того, что требования, предназначенные для конкретного использования или применения, выполнены».
Казалось бы, определения чуть ли не совпадают и уж если не полностью, то в значительной части. И, тем не менее, верификация и валидация — принципиально разные действия.

Чем отличается валидация от верификации

Чтобы было проще понять - пример типичной верификации: тестирование программы или проведение испытания оборудования. Имея определенные требования на руках, мы проводим испытание продукта и фиксируем, соблюдены ли требования. Результат верификации — это ответ на вопрос «Соответствует ли продукт требованиям?».

Но далеко не всегда продукт, соответствующий установленным требованиям, можно применять в конкретной ситуации. Например, лекарство прошло все положенные испытания и поступило в продажу. Значит ли это что оно может быть применено каким-то конкретным больным? Нет, т. к. каждый пациент имеет свои особенности и конкретно для этого лекарство может быть губительным, т.е. кто–то (врач) должен подтвердить: да, этому больному можно принимать это лекарство. То есть врач должен выполнить валидацию: придать законную силу конкретному применению.

Чем отличается валидация от верификации

Пример 2. Предприятие выпускает трубы, предназначенные для закладки в землю, в соответствии с некоторыми ТУ (Техническими условиями). Продукция этим ТУ соответствует, но поступил заказ, предполагающий укладку труб по дну моря. Могут ли трубы, соответствующие имеющимся ТУ, быть применены в данном случае? Именно валидация и дает ответ на этот вопрос.

Нетрудно видеть, что еще одно отличие состоит в том, что верификация производится всегда, а вот необходимость в валидации может и отсутствовать. Она появляется только тогда, когда возникают требования, связанные с конкретным применением продукции. Если фармацевтический завод выпускает лекарства, то он будет проверять лишь их соответствие требованиям, а проблемами применения конкретных лекарств конкретными пациентами заниматься не будет. Или тот же АвтоВАЗ.

Чем отличается валидация от верификации

Вывод: можно констатировать следующее:
верификация — проводится практически всегда, выполняется методом проверки (сличения) характеристик продукции с заданными требованиями, результатом является вывод о соответствии (или несоответствии) продукции,
валидация — проводится при необходимости, выполняется методом анализа заданных условий применения и оценки соответствия характеристик продукции этим требованиям, результатом является вывод о возможности применения продукции для конкретных условий.

Методы тестирования программ

Тестирование – это способ семантической отладки (проверки) программы, заключающийся в обработке последовательности различных контрольных наборов тестов, для которых известен результат. Тестирование основывается на выполнении программы и получении результатов выполнения тестов.

Тесты подбираются так, чтобы они охватили как можно больше различных типов ситуаций обработки элементов программы. Более слабое требование – выполнение хотя бы один раз каждого разветвления программы в определенном направлении.

Исторически первой разновидностью тестирования была отладка.

Отладка означает проверку описания программного объекта в ЯП на наличие в нем ошибок и последующее их устранение. Ошибки обнаруживаются компиляторами при проверки синтаксической правильности. После этого выполняется верификация для установлению правильности кода и валидация на проверку соответствия продукта заданным требованиям.

Следующим шагом является функциональное тестирование для проверки реализованных функций в соответствии с их спецификацией. Тестирование по внешним спецификациям проводится с учетом требований, сформулированных на этапе анализа предметной области.

Методы функционального тестирования подразделяются на статические и динамические.

Статические методы *используются при проведении инспекций и рассмотрении* спецификаций компонентов без их выполнения.

Техника статического анализа заключается в методическом просмотре (или обзоре) и анализе структуры программ, а также в доказательстве правильности.

Статический анализ направлен на анализ документов, разрабатываемых на всех этапах ЖЦ и заключается в инспекции исходного кода и сквозного контроля программы.

Инспектирование ПО – это статическая проверка соответствия программы заданным спецификациями, проводится путем анализа различных представлений результатов проектирования (документации, требований, спецификаций, схем или исходного кода программ) на процессах ЖЦ. Просмотры и инспекции результатов проектирования и соответствия их требованиям заказчика обеспечивают более высокое качество создаваемых ПС.

При инспекциях программ рассматриваются документы рабочего проектирования на этапах ЖЦ совместно независимыми экспертами и участниками разработки ПС. На начальном этапе проектирования инспектирование предполагает проверку полноты, целостности, однозначности, непротиворечивости и совместимости документов с исходными требованиями к программной системе. На этапе реализации системы под *инспекцией понимается анализ текстов программ на соблюдение требований* стандартов и принятых руководящих документов технологии программирования.

Эффективность такой проверки заключается в том, что привлекаемые эксперты пытаются взглянуть на проблему "со стороны" и подвергают ее всестороннему критическому анализу.

Эти приемы позволяют на более ранних этапах проектирования обнаружить ошибки или дефекты путем многократного просмотра исходных кодов. Символьное тестирование применяться для проверки отдельных участков программы на входных значения – символах.

Кроме того, разрабатывается множество новых способов автоматизации символьного выполнения программ. Например, автоматизированное средство статического контроля для языково–ориентированной разработки, ряд инструментов автоматизации доказательства корректности. С целью проверки спецификаций параллельных вычислений систем используется автоматизированный аппарат сетей Петри.

Динамические методы тестирования

Динамические методы используются в процессе выполнения программ. Они базируются на графе, который связывает причины ошибок с ожидаемыми реакциями на эти ошибки. В процессе тестирования накапливается информация об ошибках, которая используется при оценке надежности и качества ПС.
Динамическое тестирование ориентировано на проверку корректности ПС на множестве тестов, прогоняемых по ПС, в целях проверки и сбора данных на этапах ЖЦ и проведения измерения отдельных элементов тестирования для оценки характеристик качества, указанных в требованиях посредством выполнения системы на ЭВМ. Оно основывается на систематических, статистических, (вероятностных) и имитационных методах.
Краткая характеристика этих методов.

Систематические методы тестирования делятся на методы, в которых программы рассматриваются как "черный ящик" (используется информация о решаемой задаче), и методы, в которых программа рассматривается как " белый ящик" (используется информация о структуре программы). Этот вид называют тестированием с управлением по данным или управлением по входу–выходу. Цель – выяснение обстоятельств, при которых поведение программы не соответствует ее спецификации. При этом количество обнаруженных ошибок в программе является критерием качества входного тестирования.

Целью динамического тестирования программ по принципу «черного ящика» является выявление одним тестом максимального числа ошибок с использованием небольшого подмножества возможных входных данных.

***Методы «черного ящика»** обеспечивают:*

– эквивалентное разбиение;
– анализ граничных значений;
– применение функциональных диаграмм, которые в объединении с реверсивным анализом дают достаточно полную информацию о функционировании тестируемой программы.

Эквивалентное разбиение состоит в разбиении входной области данных программы на конечное число классов эквивалентности так, чтобы каждый тест, являющийся представителем некоторого класса, был эквивалентен любому другому тесту этого класса.

Классы эквивалентности выделяются путем перебора входных условий и разбиения их на две или более групп. При этом различают два типа классов эквивалентности: правильные, задающие входные данные для программы, и неправильные, основанные на задании ошибочных входных значений.

тестирование по принципу «черного ящика»

Разработка тестов методом эквивалентного разбиения осуществляется в два этапа: выделение классов эквивалентности и построение тестов. При построении тестов, основанных на выборе входных данных, проводится символическое выполнение программы.

Итак, методы тестирования по принципу «черного ящика» используются для тестирования функций, реализованных в программе, путем проверки несоответствия между реальным поведением функций и ожидаемым поведением с учетом спецификаций требований. Во время подготовки к этому тестированию строятся:

таблицы условий, причинно–следственные графы и области разбивки. Кроме того, подготавливаются тестовые наборы, учитывающие параметры и условия среды, которые влияют на поведение функций. Для каждого условия определяется множество значений и ограничений предикатов, которые тестируются.

Метод «белого ящика» *позволяет исследовать внутреннюю структуру программы,* причем обнаружение всех ошибок в программе является критерием исчерпывающего тестирования маршрутов ее потоков (графа) передач управления, среди которых рассматриваются:

(а) критерий покрытия операторов – набор тестов в совокупности должен
обеспечить прохождение каждого оператора не менее одного раза;

(б) критерий тестирования ветвей (известный как покрытие решений или покрытие переходов) – набор тестов в совокупности должен обеспечить прохождение каждой ветви (каждого выхода оператора) по крайней мере один раз.

Критерий (б) соответствует простому структурному тесту и наиболее распространен на практике. Для удовлетворения этого критерия необходимо построить систему путей, содержащую все ветви программы. Нахождение такого оптимального покрытия в некоторых случаях осуществляется просто, а в других является более сложной задачей.

Тестирование по принципу «белого ящика» ориентировано на проверку прохождения всех путей программ посредством применения путевого и имитационного тестирования

Путевое тестирование *применяется на уровне модулей и графовой модели* программы путем выбора тестовых ситуаций, подготовки данных и включает тестирование:

– операторов, которые должны быть выполнены хотя бы один раз, без учета ошибок, которые могут остаться в программе из–за большого количества логических путей и необходимости прохождения подмножеств этих путей;

– путей по заданному графу потоков управления для выявления разных маршрутов передачи управления с помощью путевых предикатов, для вычисления которого создается набор тестовых данных, гарантирующих прохождение всех путей. Однако, все пути выполнить невозможно, поэтому остаются и не выявленные ошибки, которые могут проявиться в процессе эксплуатации;

– блоков, разделяющих программы на отдельные части–блоки, которые выполняются один раз или многократно при нахождении путей в программе, включающих совокупность блоков реализации одной функции либо нахождения входного множества данных, которое будет использоваться для выполнения указанного пути.

«Белый ящик» базируется на структуре программы, а «черный ящик», когда о структуре ничего неизвестно. Для выполнения тестирования с помощью этих «ящиков» известными считаются выполняемые функции, входы (входные данные) и выходы (выходные данные), а также логика обработки, представленные в документации.

Функциональное тестирование

Целью функционального тестирования является обнаружение несоответствий между реальным поведением реализованных функций и ожидаемым поведением в соответствии со спецификацией и исходными требованиями. Функциональные тесты должны охватывать все реализованные функции с учетом наиболее вероятных типов ошибок. Тестовые сценарии, объединяющие отдельные тесты, ориентированы на проверку качества решения функциональных задач. Функциональные тесты создаются по внешним спецификациям функций, проектной информации и по тексту на ЯП, относятся к функциональным его характеристикам и применяются на этапе комплексного тестирования и испытаний для определения полноты реализации функциональных задач и их соответствия исходным требованиям.

В задачи функционального тестирования входят:

– идентификация множества функциональных требований;

– идентификация внешних функций и построение последовательностей функций в соответствии с их использованием в ПС;
– идентификация множества входных данных каждой функции и определение областей их изменения;
– построение тестовых наборов и сценариев тестирования функций;
– выявление и представление всех функциональных требований с помощью тестовых наборов и проведение тестирования ошибок в программе и при взаимодействии со средой.

Тесты, создаваемые по проектной информации, связаны со структурами данных, алгоритмами, интерфейсами между отдельными компонентами и применяются для тестирования отдельных компонентов и интерфейсов между ними. Основная цель – обеспечение полноты и согласованности реализованных функций и интерфейсов между ними.

Комбинированный метод "черного ящика" и "прозрачного ящика" основан на разбиении входной области функции на подобласти обнаружения ошибок. Подобласть содержит однородные элементы, которые все обрабатываются корректно либо некорректно. Для тестирования подобласти производится выполнение программы на одном из элементов этой области.

Предпосылками функционального тестирования являются:

– корректное формирование требований и ограничений к качеству ПО;

– корректное описание модели функционирования ПО в среде его эксплуатации
заказчиком;

– адекватность модели ПО заданному классу.

Организационные аспекты процесса тестирования

Под организацией проведения тестирования понимается:
– выделение объектов тестирования,
– проведение классификации ошибок для рассматриваемого класса тестируемых программ,
– подготовка тестов, их выполнение и поиск разного рода ошибок и отказов в компонентах и в системе в целом;
– служба проведения и управление процессом тестирования.

Спасибо за внимание!