19 апреля 2023 г.

Гаврилов А. Г. Новоселова О. В.

Современные технологии и средства разработки программного обеспечения (второй семестр)

Раздел 7. Основы тестирования программных продуктов

Тема 21
Процесс тестирования. Технологии тестирования. Методы тестирования

Содержание

1. Тестирование требований

1.1. Свойства качественных требований

Требование — описание того, какие функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи. В процессе тестирования требований проверяется их соответствие определённому набору свойств (Рис. 1).

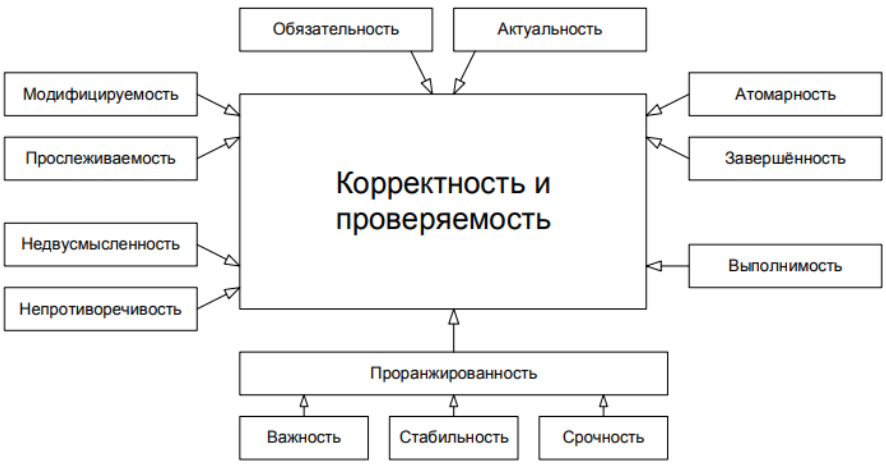

Рис. 1. Свойства качественных требований

Завершённость. Требование является полным и законченным с точки зрения представления в нём всей необходимой информации, ничто не пропущено по соображениям «это и так

всем понятно».

Типичные проблемы с завершённостью:

- Отсутствуют нефункциональные составляющие требования или ссылки на соответствующие нефункциональные требования (например: «пароли должны храниться в зашифрованном виде» — каков алгоритм шифрования?).
- Указана лишь часть некоторого перечисления (например: «экспорт осуществляется в форматы PDF, PNG и т.д.» — что мы должны понимать под «и т.д.»?).
- Приведённые ссылки неоднозначны (например: «см. выше» вместо «см. раздел 123.45.b»).

Способы обнаружения проблем
Применимы почти все техники тестирования требований, но лучше всего помогает задавание вопросов и использование графического представления разрабатываемой системы. Также очень помогает глубокое знание предметной области, позволяющее замечать пропущенные фрагменты информации.

Способы устранения проблем
Как только было выяснено, что чего-то не хватает, нужно получить недостающую информацию и дописать её в требования. Возможно, потребуется небольшая переработка требований.

Атомарность, единичность. Требование является атомарным, если его нельзя разбить на отдельные требования без потери завершённости и оно описывает одну и только одну ситуацию.

Типичные проблемы с атомарностью:

- В одном требовании, фактически, содержится несколько независимых (например: «кнопка “Restart” не должна отображаться при остановленном сервисе, окно “Log” должно вмещать не менее 20-ти записей о последних действиях пользователя» — здесь зачем-то в одном предложении описаны совершенно разные элементы интерфейса в совершенно разных контекстах).
- Требование допускает разночтение в силу грамматических особенностей языка (например: «если пользователь подтверждает заказ и редактирует заказ или откладывает заказ, должен выдаваться запрос на оплату» — здесь описаны три разных случая, и это требование стоит разбить на три отдельных во избежание путаницы). Такое нарушение атомарности часто влечёт за собой возникновение противоречивости.
- В одном требовании объединено описание нескольких независимых ситуаций (например: «когда пользователь входит в систему, ему должно отображаться приветствие; когда пользователь вошёл в систему, должно отображаться имя пользователя; когда пользователь выходит из системы, должно отображаться прощание» — все эти три ситуации заслуживают того, чтобы быть описанными отдельными и куда более детальными требованиями).

Способы обнаружения проблем
Обдумывание, обсуждение с коллегами и здравый смысл: если мы считаем, что некий раздел требований перегружен и требует декомпозиции, скорее всего, так и есть.

Способы устранения проблем
Переработка и структурирование требований: разбиение их на разделы, подразделы, пункты, подпункты и т.д

Непротиворечивость, последовательность. Требование не должно содержать внутренних противоречий и противоречий другим требованиям и документам.

Типичные проблемы с непротиворечивостью:

- Противоречия внутри одного требования (например: «после успешного входа в систему пользователя, не имеющего права входить в систему…» — тогда как он успешно вошёл в систему, если не имел такого права?)

- Противоречия между двумя и более требованиями, между таблицей и текстом, рисунком и текстом, требованием и прототипом и т.д. (например: «712.a Кнопка “Close” всегда должна быть красной» и «36452.x Кнопка “Close” всегда должна быть синей» — так всё же красной или синей?)

- Использование неверной терминологии или использование разных терминов для обозначения одного и того же объекта или явления (например: «в случае, если разрешение окна составляет менее 800x600…» — разрешение есть у экрана, у окна есть размер)

Способы обнаружения проблем
Лучше всего обнаружить противоречивость помогает хорошая память, но даже при её наличии незаменимым инструментом является графическое представление разрабатываемой системы, позволяющее представить всю ключевую информацию в виде единой согласованной схемы (на которой противоречия очень заметны).

Способы устранения проблем
После обнаружения противоречия нужно прояснить ситуацию с заказчиком и внести необходимые правки в требования.

Недвусмысленность. Требование должно быть описано без использования жаргона, неочевидных аббревиатур и расплывчатых формулировок, должно допускать только однозначное объективное понимание и быть атомарным в плане невозможности различной трактовки сочетания отдельных фраз.

Типичные проблемы с недвусмысленностью:

- Использование терминов или фраз, допускающих субъективное толкование (например: «приложение должно поддерживать передачу больших объёмов данных» — насколько «больших»?). В таблице 1 представлен краткий перечень слов и выражений, которые можно считать верными признаками двусмысленности:

  Вот утрированный пример требования, звучащего очень красиво, но совершенно нереализуемого и непонятного: «В случае необходимости оптимизации передачи больших файлов система должна эффективно использовать минимум оперативной памяти, если это возможно».

- Использование неочевидных или двусмысленных аббревиатур без расшифровки (например: «доступ к ФС осуществляется посредством системы прозрачного шифрования» и «ФС предоставляет возможность фиксировать сообщения в их текущем состоянии с хранением истории всех изменений» — ФС здесь обозначает файловую систему? Точно? А не какой-нибудь «Фиксатор Сообщений»?)

- Формулировка требований из соображений, что нечто должно быть всем очевидно (например: «Система конвертирует входной файл из формата PDF в выходной файл формата PNG» — и при этом автор считает совершенно очевидным, что имена файлов систма получает из командной строки, а многостраничный PDF конвертируется в несколько PNG-файлов, к именам которых добавляется «page-1», «page-2» и т.д.). Эта проблема перекликается с нарушением корректности.

Таблица 1. Перечень не рекомендуемых слов при составлении требований

– адекватно (adequate)	– нормально (normal)
– легко (easy)	– минимизировать (minimize)
– обеспечивать (provide for)	– максимизировать (maximize)
– как минимум (as a minimum)	– оптимизировать (optimize)
– быть способным (be capable of)	– быстро (rapid)
– эффективно (effectively)	– удобно (user-friendly)
– своевременно (timely)	– просто (simple)
– применимо (as applicable)	– часто (often)
– если возможно (if possible)	– обычно (usual)
– будет определено позже (to be determined, TBD)	– большой (large)
– по мере необходимости (as appropriate)	– гибкий (flexible)
– если это целесообразно (if practical)	– устойчивый (robust)
– но не ограничиваясь (but not limited to)	– по последнему слову техники (state-of-the-art)
– быть способно (capability of)	– улучшенный (improved)
– иметь возможность (capability to)	– результативно (efficient)

Способы обнаружения проблем

Увидеть в требованиях двусмысленность хорошо помогают перечисленные выше слова-индикаторы. Столь же эффективным является продумывание проверок: очень тяжело придумать объективную проверку для требования, допускающего разночтение.

Способы устранения проблем

Самый страшный враг двусмысленности – числа и формулы: если что-то можно выразить в формульном или числовом виде (вместо словесного описания), обязательно стоит это сделать. Если это невозможно, стоит хотя бы использовать максимально точные технические термины, отсылки к стандартам и т.п

Выполнимость. Требование должно быть технологически выполнимым и реализуемым в рамках бюджета и сроков разработки проекта.

Типичные проблемы с выполнимостью:

- Так называемое «озолочение» (gold plating) — требования, которые крайне долго и/или дорого реализуются и при этом практически бесполезны для конечных пользователей (например: «настройка параметров для подключения к базе данных должна поддерживать распознавание символов из жестов, полученных с устройств трёхмерного ввода»).
- Технически нереализуемые на современном уровне развития технологий требования (например: «анализ договоров должен выполняться с применением искусственного интеллекта, который будет выносить однозначное корректное заключение о степени выгоды от заключения договора»).
- В принципе нереализуемые требования (например: «система поиска должна заранее предусматривать все возможные варианты поисковых запросов и кэшировать их результаты»).

Способы обнаружения проблем

Увы, здесь есть только один путь: максимально нарабатывать опыт и исходить из него. Невозможно понять, что некоторое требование «стоит» слишком много или вовсе невыполнимо, если нет понимания процесса разработки ПО, понимания предметной области и иных сопутствующих знаний.

Способы устранения проблем

При обнаружении невыполнимости требования не остаётся ничего другого, как подробно обсудить ситуацию с заказчиком и или изменить требование (возможно – отказаться от него), или пересмотреть условия выполнения проекта (сделав выполнение данного требования возможным).

Обязательность, **нужность** и **актуальность** (up-to-date). Если требование не является обязательным к реализации, оно должно быть просто исключено из набора требований. Если требование нужное, но «не очень важное», для указания этого факта используется указание приоритета (см. «проранжированность по…»). Также исключены (или переработаны) должны быть требования, утратившие актуальность.

Типичные проблемы с обязательностью и актуальностью:

- Требование было добавлено «на всякий случай», хотя реальной потребности в нём не было и нет.
- Требованию выставлены неверные значения приоритета по критериям важности и/или срочности.
- Требование устарело, но не было переработано или удалено.

Способы обнаружения проблем

Постоянный (периодический) пересмотр требований (желательно – с участием заказчика) позволяет заметить фрагменты, потерявшие актуальность или ставшие низкоприоритетными.

Способы устранения проблем

Переработка требований (с устранением фрагментов, потерявших актуальность) и переработкой фрагментов, у которых изменился приоритет (часто изменение приоритета ведёт и к изменению формулировки требования).

Прослеживаемость. Прослеживаемость бывает вертикальной (vertical traceability) и горизонтальной (horizontal traceability). Вертикальная позволяет соотносить между собой требования на различных уровнях требований, горизонтальная позволяет соотносить требование с тест-планом, тест-кейсами, архитектурными решениями и т.д. Для обеспечения прослеживаемости часто используются специальные инструменты по управлению требованиями (requirements management tool) и/или матрицы прослеживаемости (traceability matrix).

Типичные проблемы с прослеживаемостью:

- Требования не пронумерованы, не структурированы, не имеют оглавления, не имеют работающих перекрёстных ссылок.
- При разработке требований не были использованы инструменты и техники управления требованиями.
- Набор требований неполный, носит обрывочный характер с явными «пробелами».

Способы обнаружения проблем

Нарушения прослеживаемости становятся заметны в процессе работы с требованиями, как только у нас возникают остающиеся без ответа вопросы вида «откуда взялось это требование?», «где описаны сопутствующие (связанные) требования?», «на что это влияет?».

Способы устранения проблем

Переработка требований. Возможно, придётся даже менять структуру набора требований, но всё точно начнётся с расстановки множества перекрёстных ссылок, позволяющих осуществлять быструю и прозрачную навигацию по набору требований.

Модифицируемость. Это свойство характеризует простоту внесения изменений в отдельные требования и в набор требований. Можно говорить о наличии модифицируемости в том случае, если при доработке требований искомую информацию легко найти, а её изменение не приводит к нарушению иных описанных в этом перечне свойств.

Типичные проблемы с модифицируемостью:

- Требования неатомарны (см. «атомарность») и непрослеживаемы (см. «прослеживаемость»), а потому их изменение с высокой вероятностью порождает противоречивость (см. «непротиворечивость»).
- Требования изначально противоречивы (см. «непротиворечивость»). В такой ситуации внесение изменений (не связанных с устранением противоречивости) только усугубляет ситуацию, увеличивая противоречивость и снижая прослеживаемость.
- Требования представлены в неудобной для обработки форме (например, не использованы инструменты управления требованиями, и в итоге команде приходится работать с десятками огромных текстовых документов).

Способы обнаружения проблем

Если при внесении изменений в набор требований, мы сталкиваемся с проблемами, характерными для ситуации потери прослеживаемости, значит – мы обнаружили проблему с модифицируемостью. Также модифицируемость ухудшается при наличии практически любой из рассмотренных в данном разделе проблем с требованиями.

Способы устранения проблем

Переработка требований с первостепенной целью повысить их прослеживаемость. Параллельно можно устранять иные обнаруженные проблемы.

Проранжированность по важности, стабильности, срочности (ranked for importance, stability, priority). Важность характеризует зависимость успеха проекта от успеха реализации требования. Стабильность характеризует вероятность того, что в обозримом будущем в требование не будет внесено никаких изменений. Срочность определяет распределение во времени усилий проектной команды по реализации того или иного требования.

Типичные проблемы с проранжированностью состоят в её отсутствии или неверной реализации и приводят к следующим последствиям.

- Проблемы с проранжированностью по важности повышают риск неверного распределения усилий проектной команды, направления усилий на второстепенные задачи и конечного провала проекта из-за неспособности продукта выполнять ключевые задачи с соблюдением ключевых условий.
- Проблемы с проранжированностью по стабильности повышают риск выполнения бессмысленной работы по совершенствованию, реализации и тестированию требований, которые в самое ближайшее время могут претерпеть кардинальные изменения (вплоть до полной утраты актуальности).
- Проблемы с проранжированностью по срочности повышают риск нарушения желаемой заказчиком последовательности реализации функциональности и ввода этой функциональности в эксплуатацию.

Способы обнаружения проблем
Как и в случае с актуальностью и обязательностью требований, здесь лучшим способом обнаружения недоработок является постоянный (периодический) пересмотр требований (желательно – с участием заказчика), в процессе которого можно обнаружить неверные значения показателей срочности, важности и стабильности обсуждаемых требований.

Способы устранения проблем
Прямо в процессе обсуждения требований с заказчиком (во время пересмотра требований) стоит вносить правки в значения показателей срочности, важности и стабильности обсуждаемых требований.

Корректность (correctness) и **проверяемость** (verifiability). Фактически эти свойства вытекают из соблюдения всех вышеперечисленных (или можно сказать, что они не выполняются, если нарушено хотя бы одно из вышеперечисленных). В дополнение можно отметить, что проверяемость подразумевает возможность создания объективного тест-кейса (тест-кейсов), однозначно показывающего, что требование реализовано верно и поведение приложения в точности соответствует требованию.

К типичным проблемам с корректностью также можно отнести:

- опечатки (особенно опасны опечатки в аббревиатурах, превращающие одну осмысленную аббревиатуру в другую также осмысленную, но не имеющую отношения к некоему контексту; такие опечатки крайне сложно заметить);
- наличие неаргументированных требований к дизайну и архитектуре;
- плохое оформление текста и сопутствующей графической информации, грамматические, пунктуационные и иные ошибки в тексте;
- неверный уровень детализации (например, слишком глубокая детализация требования на уровне бизнес-требований или недостаточная детализация на уровне требований к продукту);
- требования к пользователю, а не к приложению (например: «пользователь должен быть в состоянии отправить сообщение» — увы, мы не можем влиять на состояние пользователя)

Способы обнаружения проблем
Поскольку здесь мы имеем дело с «интегральной» проблемой, обнаруживается она с использованием ранее описанных способов. Отдельных уникальных методик здесь нет.

Способы устранения проблем
Внесение в требования необходимых изменений – от элементарной правки обнаруженной опечатки, до глобальной переработки всего набора требований.

1.2. Техники тестирования требований

Тестирование документации и требований относится к разряду нефункционального тестирования. Основные техники такого тестирования в контексте требований таковы.

1. Взаимный просмотр (peer review). Взаимный просмотр («рецензирование») является одной из наиболее активно используемых техник тестирования требований и может быть представлен в одной из трёх следующих форм (по мере нарастания его сложности и цены):

- *Беглый просмотр* (walkthrough) может выражаться как в показе автором своей работы коллегам с целью создания общего понимания и получения обратной связи, так и в простом обмене результатами работы между двумя и более авторами с тем, чтобы коллега высказал свои вопросы и замечания. Это самый быстрый, дешёвый и часто используемый вид просмотра. Для запоминания: аналог беглого просмотра — это ситуация, когда

вы в школе с одноклассниками проверяли перед сдачей сочинения друг друга, чтобы найти описки и ошибки.

- *Технический просмотр* (technical review) выполняется группой специалистов. В идеальной ситуации каждый специалист должен представлять свою область знаний. Тестируемый продукт не может считаться достаточно качественным, пока хотя бы у одного просматривающего остаются замечания. Для запоминания: аналог технического просмотра — это ситуация, когда некий договор визирует юридический отдел, бухгалтерия и т.д.

- *Формальная инспекция* (inspection) представляет собой структурированный, систематизированный и документируемый подход к анализу документации. Для его выполнения привлекается большое количество специалистов, само выполнение занимает достаточно много времени, и потому этот вариант просмотра используется достаточно редко (как правило, при получении на сопровождение и доработку проекта, созданием которого ранее занималась другая компания). Для запоминания: аналог формальной инспекции — это ситуация генеральной уборки квартиры (включая содержимое всех шкафов, холодильника, кладовки и т.д.).

2. Вопросы. Следующей очевидной техникой тестирования и повышения качества требований является (повторное) использование техник выявления требований, а также (как отдельный вид деятельности) — задавание вопросов (табл. 2). Если хоть что-то в требованиях вызывает у вас непонимание или подозрение — задавайте вопросы. Можно спросить представителей заказчика, можно обратиться к справочной информации. По многим вопросам можно обратиться к более опытным коллегам при условии, что у них имеется соответствующая информация, ранее полученная от заказчика. Главное, чтобы ваш вопрос был сформулирован таким образом, чтобы полученный ответ позволил улучшить требования.

3. Тест-кейсы и чек-листы. Мы помним, что хорошее требование является проверяемым, а значит, должны существовать объективные способы определения того, верно ли реализовано требование. Продумывание чек-листов или даже полноценных тест-кейсов в процессе анализа требований позволяет нам определить, насколько требование проверяемо. Если вы можете быстро придумать несколько пунктов чек-листа, это ещё не признак того, что с требованием всё хорошо (например, оно может противоречить каким-то другим требованиям). Но если никаких идей по тестированию требования в голову не приходит — это тревожный знак.

Рекомендуется для начала убедиться, что вы понимаете требование (в том числе прочесть соседние требования, задать вопросы коллегам и т.д.). Также можно пока отложить работу с данным конкретным требованием и вернуться к нему позднее — возможно, анализ других требований позволит вам лучше понять и это конкретное. Но если ничто не помогает — скорее всего, с требованием что-то не так.

Справедливости ради надо отметить, что на начальном этапе проработки требований такие случаи встречаются очень часто — требования сформированы очень поверхностно, расплывчато и явно нуждаются в доработке, т.е. здесь нет необходимости проводить сложный анализ, чтобы констатировать непроверяемость требования.

На стадии же, когда требования уже хорошо сформулированы и протестированы, вы можете продолжать использовать эту технику, совмещая разработку тест-кейсов и дополнительное тестирование требований.

4. Исследование поведения системы. Эта техника логически вытекает из предыдущей (продумывания тест-кейсов и чек-листов), но отличается тем, что здесь тестированию подвергается, как правило, не одно требование, а целый набор. Тестировщик мысленно моделирует процесс работы пользователя с системой, созданной по тестируемым требованиям, и ищет неоднозначные или вовсе неописанные варианты поведения системы. Этот подход сложен, требует достаточной квалификации тестировщика, но способен выявить нетривиальные недоработки, которые почти невозможно заметить, тестируя требования по отдельности.

Таблица 2. Примеры хороших и плохих вопросов при уточнении требований

Плохое требование	**Плохие вопросы**	**Хорошие вопросы**
«Приложение должно быстро запускаться».	«Насколько быстро?» (На это вы рискуете получить ответы в стиле «очень быстро», «максимально быстро», «нууу… просто быстро»). «А если не получится быстро?» (Этим вы рискуете просто удивить или даже разозлить заказчика.) «Всегда?» («Да, всегда». Хм, а вы ожидали другого ответа?)	«Каково максимально допустимое время запуска приложения, на каком оборудовании и при какой загруженности этого оборудования операционной системой и другими приложениями? На достижение каких целей влияет скорость запуска приложения? Допускается ли фоновая загрузка отдельных компонентов приложения? Что является критерием того, что приложение закончило запуск?»
Опционально должен поддерживаться экспорт документов в формат PDF».	«Любых документов?» (Ответы «да, любых» или «нет, только открытых» вам всё равно не помогут.) «В PDF какой версии должен производиться экспорт?» (Сам по себе вопрос хорош, но он не даёт понять, что имелось в виду под «опционально».) «Зачем?» («Нужно!» Именно так хочется ответить, если вопрос не раскрыт полностью.)	«Насколько возможность экспорта в PDF важна? Как часто, кем и с какой целью она будет использоваться? Является ли PDF единственным допустимым форматом для этих целей или есть альтернативы? Допускается ли использование внешних утилит (например, виртуальных PDF-принтеров) для экспорта документов в PDF?»
«Если дата события не указана, она выбирается автоматически»	«А если указана?» (То она указана. Логично, не так ли?) «А если дату невозможно выбрать автоматически?» (Сам вопрос интересен, но без пояснения причин невозможности звучит как издёвка.) «А если у события нет даты?» (Тут автор вопроса, скорее всего, хотел уточнить, обязательно ли это поле для заполнения. Но из самого требования видно, что обязательно: если оно не заполнено человеком, его должен заполнить компьютер.)	«Возможно, имелось в виду, что дата генерируется автоматически, а не выбирается? Если да, то по какому алгоритму она генерируется? Если нет, то из какого набора выбирается дата и как генерируется этот набор? P.S. Возможно, стоит использовать текущую дату?»

5. Рисунки (графическое представление). Чтобы увидеть общую картину требований целиком, очень удобно использовать рисунки, схемы, диаграммы, интеллект-карты108 и т.д. Графическое представление удобно одновременно своей наглядностью и краткостью (например, UML-схема базы данных, занимающая один экран, может быть описана несколькими десятками страниц текста). На рисунке очень легко заметить, что какие-то элементы «не стыкуются», что где-то чего-то не хватает и т.д. Если вы для графического представления требований будете

использовать общепринятую нотацию (например, уже упомянутый UML), вы получите дополнительные преимущества: вашу схему смогут без труда понимать и дорабатывать коллеги, а в итоге может получиться хорошее дополнение к текстовой форме представления требований.

6. Прототипирование. Можно сказать, что прототипирование часто является следствием создания графического представления и анализа поведения системы. С использованием специальных инструментов можно очень быстро сделать наброски пользовательских интерфейсов, оценить применимость тех или иных решений и даже создать не просто «прототип ради прототипа», а заготовку для дальнейшей разработки, если окажется, что реализованное в прототипе (возможно, с небольшими доработками) устраивает заказчика.

2. Тестирование на основе классов эквивалентности и граничных значений

Тестирование на основе классов эквивалентности — техника тестирования, направленная на сокращение количества разрабатываемых и выполняемых тест-кейсов при сохранении достаточного тестового покрытия. Суть техники состоит в выявлении наборов эквивалентных тест-кейсов (каждый из которых проверяет одно и то же поведение приложения) и выборе из таких наборов небольшого подмножества тест-кейсов, с наибольшей вероятностью обнаруживающих проблему.

Класс эквивалентности — набор данных, обрабатываемых одинаковым образом и приводящих к одинаковому результату.

Граничное условие — значение, находящееся на границе классов эквивалентности. Иногда под классом эквивалентности понимают набор тест-кейсов, полное выполнение которого является избыточным. Это определение не противоречит предыдущему, т.к. показывает ту же ситуацию, но с другой точки зрения.

В качестве пояснения идеи рассмотрим тривиальный пример. Допустим, нам нужно протестировать функцию, которая определяет, корректное или некорректное имя ввёл пользователь при регистрации.

Требования к имени пользователя таковы:

- От трёх до двадцати символов включительно.
- Допускаются цифры, знак подчёркивания, буквы английского алфавита в верхнем и нижнем регистрах.

Если попытаться решить задачу «в лоб», нам для позитивного тестирования придётся перебрать все комбинации допустимых символов длиной [3, 20] (это 18-разрядное 63-ричное число, т.е. 2.4441614509104E+32). А негативных тест-кейсов здесь и вовсе будет бесконечное количество, ведь мы можем проверить строку длиной в 21 символ, 100, 10000, миллион, миллиард и т.д.

Представим графически классы эквивалентности относительно требований к длине (рис. 2).

Поскольку для длины строки невозможны дробные и отрицательные значения, мы видим три недостижимых области, которые можно исключить, и получаем окончательный вариант (рис. 3).

Мы получили три класса эквивалентности:

- [0, 2] — недопустимая длина;
- [3, 20] — допустимая длина;
- [21, ∞] — недопустимая длина.

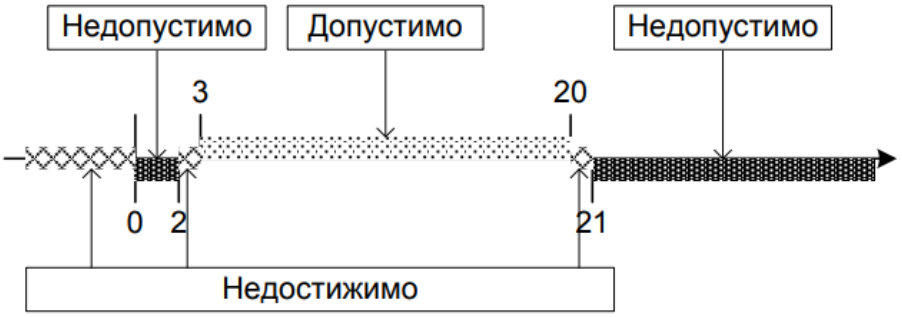

Рис. 2. Классы эквивалентности для значений длины имени пользователя

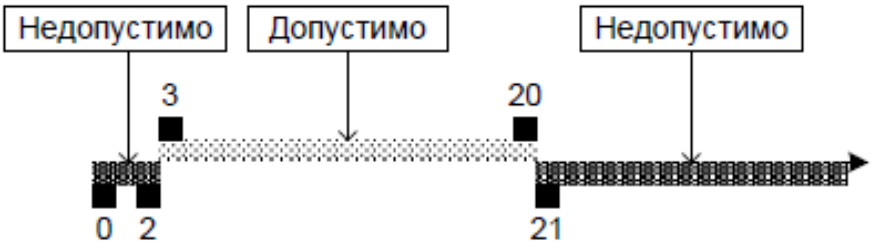

Рис. 3. Классы эквивалентности для значений длины имени пользователя

Список литературы

[1] Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. — 3-е изд. — Минск: Четыре четверти, 2020. — 312 с. ISBN 978-985-581-362-1.