Формализация требований на практике

В. В. Кулямин, Н. В. Пакулин, О. Л. Петренко, А. А. Сортов, А. В. Хорошилов
{kuliamin,npak,olga,sortov,khoroshilov}@ispras.ru

Аннотация

Чтобы сделать хорошую — надежную и правильно работающую — программную систему, необходимо понять, какие задачи она должна будет решать, и реализовать в ее рамках корректные, эффективные и удобные методы их решения. Несмотря на краткость этой мысли и ее кажущуюся простоту, добиться всего этого на практике очень непросто. Одним из перспективных подходов к выполнению второй части этой максимы — реализации «правильных» методов решения для уже поставленных задач — является использование формальных моделей разрабатываемого программного обеспечения. Однако этот подход ничего не говорит о том, как правильно понять и поставить задачу, что критически важно для его успеха, поскольку именно задачи и необходимые свойства их решений должны представляться используемыми в нем моделями. Данная работа посвящена приемам и методам обеспечения адекватного понимания потребностей и нужд пользователей программного обеспечения, а также более широкому кругу вопросов работы с требованиями и их аккуратного отражения в формальных моделях.

Содержание

Введение

Основой успеха при создании надежного и полезного программного обеспечения (ПО) всегда является четкое понимание потребностей его пользователей. В наше время, когда технологии разработки программ все время совершенствуются и наращивают свои возможности, неправильное понимание нужд пользователей создаваемого ПО все еще остается одной из часто встречающихся причин провальных результатов проектов по его разработке. Самое неприятное свойство таких провалов — то, что неудачными могут оказаться результаты даже проекта, вполне успешного с точки зрения его участников и внутренних характеристик, и это станет ясно только в самом его конце.

Небольшую программу еще можно создать на основе понимания, сложившегося у разработчика после выслушивания пожеланий пользователей и заказчиков — таковых у небольшой программы обычно немного. Для сложных же систем этот способ просто не работает из-за гораздо б***о***льшего

объема информации, которую нужно усвоить, и гораздо более серьезных последствий всех принимаемых решений. Он также приводит к серьезным проблемам совместимости программ, разрабатываемых разными людьми и организациями, поскольку они могут по-разному понимать потребности пользователей и иметь разные взгляды на предметную область в целом.

Чтобы обеспечить более адекватный учет нужд пользователей при создании ПО, в рамках процесса его разработки обычно выделяют особую деятельность, называемую ***анализом требований***, и включающую, как минимум, следующие действия.

- Предварительный анализ предметной области. Это выделение ее сущностей, связей между ними, а также типовых задач в ее рамках, позволяющее впоследствии понимать пользователей.
- Сбор пожеланий и выявление действительных нужд всех заинтересованных лиц.
- Формулировка на их основе требований к создаваемой программе.
- Фиксация полученной информации в виде ряда документов и моделей, которые уже используются непосредственно разработчиками программных систем.

Возникающие в результате документы должны содержать информацию двух видов. В первую очередь, они должны описывать предметную область, в которой будет работать создаваемая система. В такое описание входит набор сущностей, с которыми предстоит иметь дело, возможные связи и взаимоотношения между ними, набор задач, которые нужно уметь решать, и критерии корректности их решений. Информация другого рода — это требования именно к данной системе: какие именно задачи должна уметь решать она, и какие специфические критерии правильности решений нужно использовать в том окружении, в котором ей придется работать.

Фиксация всей этой информации в документах и моделях должна помочь перейти от неясных, часто неосознанных, противоречивых и постоянно изменяющихся ***проблем*** и ***потребностей*** пользователей к четко сформулированным, непротиворечивым и однозначно понимаемым ***требованиям***, которые уже можно использовать для разработки программных систем. Причем, независимо от конкретных разработчиков и организаций, участвующих в их создании на основе выделенных требований, итоговые системы должны получаться «одинаковыми», т.е. взаимозаменяемыми при решении любой задачи. Кроме того, часто требуется, чтобы разные части или модули таких систем могли правильно взаимодействовать друг с другом, не замечая различий между элементами, разработанными в рамках одного проекта и в независимых проектах.

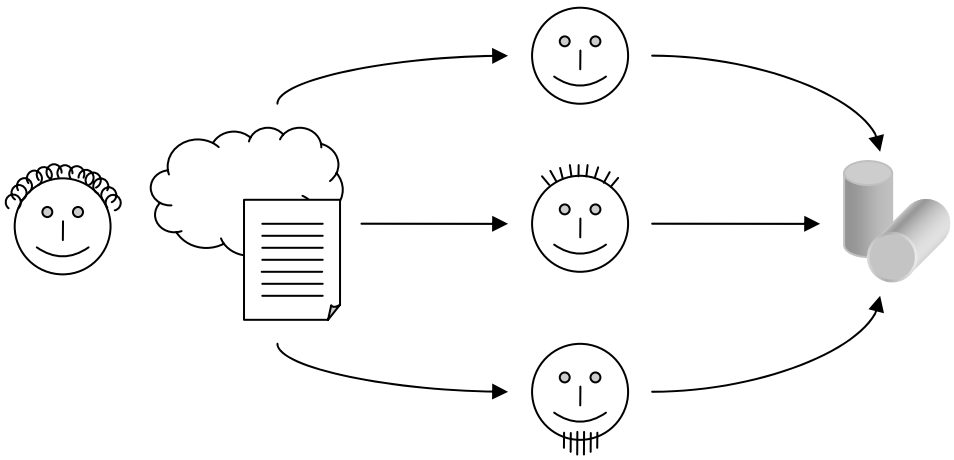


Рисунок 1. Нужно, чтобы на основе требований разные разработчики делали «одинаковые» системы.

Примерами документов, описывающих требования к программным системам, являются как технические задания и спецификации требований, разрабатываемые при создании конкретного ПО, так и программные стандарты, определяющие функциональность широкого круга систем в рамках одной предметной области.

Желательные свойства самих требований и определяющих их документов описываются в рамках международных стандартов, регламентирующих разработку программного обеспечения IEEE 830 [1] и IEEE 1233 [2]. Согласно им правильно сформулированные требования должны обладать, в частности, следующими характеристиками.

- ***Адекватность*** — соответствие сформулированных требований потребностям, ожиданиям и интересам заинтересованных лиц (пользователей, заказчиков, контролирующих организаций и пр.).
- ***Однозначность*** — одинаковость понимания требований экспертами в данной предметной области.
- ***Непротиворечивость*** — согласованность требований друг с другом, отсутствие противоречий и расхождений между ними.
- ***Полнота*** — охваченность требованиями всех ожидаемых аспектов создаваемой системы, всех существенных потребностей пользователей и интересов всех заинтересованных лиц, мнение которых решено было учитывать.

К сожалению, зачастую и спецификации требований к конкретным системам, и даже широко используемые стандарты на программные интерфейсы, не обладают указанными свойствами, что значительно снижает их полезность. В частности, возникает возможность неоднозначного понимания сформулированных в них требований разработчиками и создания на их основе серьезно различающихся и несовместимых систем, что отрицательно сказывается на удовлетворенности пользователей и полезности этих систем для них.

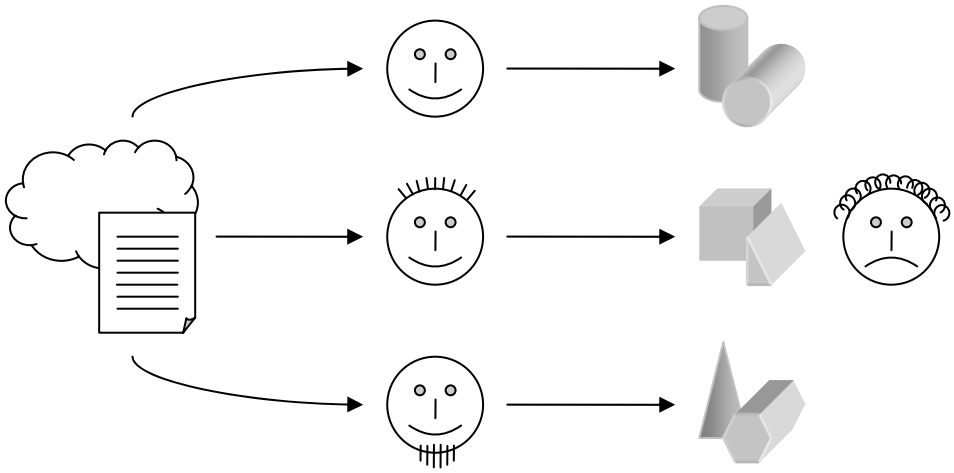


Рисунок 2. Возможные различия в понимании требований разными разработчиками приводят к отличающимся и несовместимым системам.

Например, стандарт на базовые библиотеки операционных систем IEEE 1003.1 (POSIX) [3] при определении поведения математических функций ссылается на стандарт языка C ISO/IEC 9899 [4], который для большинства из них требует только, чтобы данная функция C вычисляла соответствующую математическую функцию. В нем достаточно детально описываются исключительные ситуации: возникающие переполнения, неопределенность значения функции для данного значения параметра и пр., иногда уточняются значения функций в некоторых точках. Но ничего не говорится о точности вычисления функций. Таким образом, пользователи стандартных математических библиотек C и POSIX не имеют данных о точности производимых вычислений. Те из них, кому нужны точные вычисления или хотя бы достоверная информация о возникающих погрешностях, вынуждены пользоваться специализированными библиотеками.

Таким образом, ни анализ требований на основе неясных и противоречивых потребностей пользователей, ни их извлечение из гораздо более четких и согласованных стандартов, не дают строгих гарантий соответствия ПО нуждам людей и даже совместимости различных систем, разработанных на основе, казалось бы, одной и той же информации.

Как же добиться нужного качества формулировки требований? Как обеспечить однозначность их понимания различными людьми? Эту задачу может решить ***формализация*** требований, т.е. представление их в рамках некоторого математического формализма, которое часть указанных выше свойств требований обеспечивает только за счет выбранной формы их представления, а другие позволяет проконтролировать, затратив на это приемлемые усилия.

В частности, само по себе формальное представление некоторой информации всегда *однозначно*. Существуют строгие и выверенные на практике методы проверки формальных систем на *полноту* и *непротиворечивость*.

Однако обеспечение ***адекватности*** результатов формализации остается одной из очень сложных задач. Само по себе использование формализма не дает никаких гарантий их адекватности — формальные системы могут быть никак не связаны с реальной жизнью. Проверка ее также не облегчается использованием формальных методов, поскольку адекватность означает правильность фиксации неформальных знаний и потребностей в рамках избранного формализма, т.е. сама по себе не имеет формального смысла. Поэтому для обеспечения адекватности нужно привлекать дополнительные техники.

Ян Грэхем (Ian Graham) [5,6] приводит следующий пример неадекватной формализации, выполненной в рамках реального проекта. Одно из требований к системе управления самолетом состояло в том, что при посадке тормозной двигатель малой тяги не должен включаться, пока самолет не коснется взлетно-посадочной полосы. При формализации было принято решение, что условие касания полосы эквивалентно вращению колес шасси самолета — ведь при этом колеса должны войти с ней в контакт. Все работало хорошо, пока однажды самолету не пришлось садиться на полосу после сильного ливня, покрытую водой, по которой колеса начали скользить. В результате тормозной двигатель не включился, и самолет выехал за пределы полосы.

Таким образом, успех формализации в огромной степени зависит от ее адекватности. Адекватность же формулировки требований в документе, формальном или неформальном, определяется адекватностью понимания предметной области и нужд пользователей, сложившегося у разработчиков этого документа. Как можно убедиться в адекватности понимания чего-то человеком? На первый взгляд, это очень сложно, поскольку понимание — это внутренняя характеристика его сознания. Вовне оно проявляется только в виде отдельных явлений, каждое из которых само по себе не дает гарантий правильности понимания в целом.

Однако уже много лет люди занимаются преодолением подобных трудностей и часто довольно успешно делают это. Аналогичная задача решается, например, в рамках создания стандартов на программное обеспечение, в которых должны быть зафиксированы требования к функциональности систем, решающих определенный набор задач в заданной предметной области. Качество таких стандартов также в большой мере зависит от адекватности понимания предметной области их авторами. Посмотрим, каким образом обеспечивается эта адекватность.

Группа разработчиков стандарта формирует его черновой вариант на основе имеющегося у них на текущий момент понимания. При этом знание, присутствующее в виде многочисленных документов, описывающих требования к аналогичным системам, а также неясных и противоречивых потребностей разнообразных групп пользователей затрагиваемых стандартом систем, получает совершенно новую форму. Оно излагается в соответствии с определенной заранее структурой стандарта, обычно соответствующей делению предметной области на различные области функциональности и ее реализации с помощью набора элементов программного интерфейса — классов, операций, типов данных, элементов общих данных и пр. Таким образом, происходит ***реструктуризация*** описываемого знания.

Другой важной деятельностью при разработке стандарта является оценка его текста широкой аудиторией из экспертов в данной области, исследователей, работающих в ней и смежных областях, известных специалистов-практиков, имевших дело с системами, аналогичными описываемым данным стандартом, и участвовавших в их разработке. Результатом такой оценки

является набор замечаний и дополнений к тексту стандарта, а также вносимых в него изменений. Это означает, что текст будущего стандарта подвергается ***всесторонней оценке*** со стороны представителей сообщества заинтересованных лиц, а его авторы получают ***обратную связь***, замыкающую информационные потоки между описываемой областью знания и текстом стандарта.

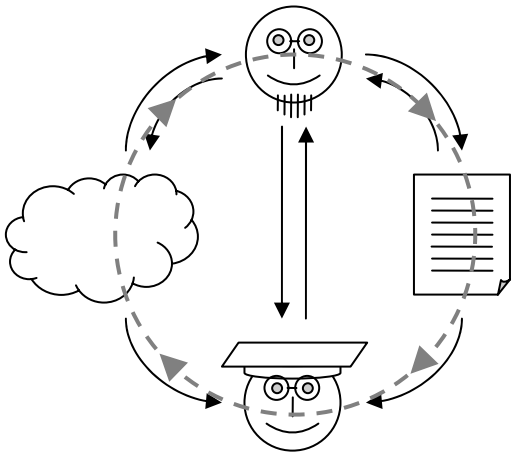


Рисунок 3. Круговорот информации обеспечивает адекватность ее изложения в другой форме.

На основании этого описания можно предположить, что адекватное понимание некоторой области или некоторого документа формируется у человека при выполнении набора действий ***реструктуризация-оценка-обратная связь***, который можно назвать образцом формирования понимания.

- ***Реструктуризация.***
  Человек пытается изложить имеющиеся у него знания в некотором виде, ***отличающемся по структуре*** от того, в котором они представлены исходно. Отличие в структуре изложения важно, поскольку, пытаясь повторить информацию ровно в том виде, как она представлена в исходных документах, человек чаще всего не достигает более глубокого ее понимания.
- ***Оценка.***
  Пытаясь изложить свое текущее понимание в новом виде, человек иногда наталкивается на неясные места, в которых он не знает, что именно нужно написать. При этом он вынужден обращаться к исходному документу, экспертам или другим источникам, содержащим нужную информацию. Это способствует углублению понимания предмета.
  Сами получающиеся документы тоже могут быть подвергнуты анализу экспертами в данной области. По результатам анализа формируется список неясных или некорректных мест в полученном документе, а также выявленных противоречий и дополнений к нему.
- ***Обратная связь.***
  Результаты анализа, проведенного экспертами, должны использоваться человеком для доработки написанных им документов. При этом и его понимание предмета углубляется, и содержание документа становится более полным и корректным.

Применить описанную технику можно не только к неструктурированному знанию, примером которого является набор неоформленных пожеланий будущих пользователей новой системы, но и к уже сформированным документам. Такими документами могут быть описания требований к конкретному ПО, полученные в результате их первичного анализа, или тексты стандартов, которые часто сами по себе достаточно тяжелы для восприятия.

Важным условием применимости описанной техники является возможность представить информацию из рассматриваемой области в нескольких существенно отличающихся друг от друга формах. Чем больше различных форм представления информации человек может использовать, чем более глубоким оказывается его понимание этой информации. Таким образом,

реструктуризация информации может рассматриваться как одна из техник, гарантирующих понимание этой информации и, следовательно, дающих возможность добиться нужной степени адекватности при построении формальных моделей.

Данная статья представляет схему процесса построения формальных моделей требований на основе исходных неясных и противоречивых потребностей пользователей. Предлагаемый процесс основан на итеративном применении реструктуризации знаний для обеспечения адекватности получаемых моделей, а также других характеристик качества таких моделей.

Формальные модели требований были бы просто еще одной кипой бумаг на пути между желаниями пользователей и реальной системой, если бы они не давали ряд преимуществ при разработке качественного ПО. Среди предоставляемых ими возможностей в первую очередь нужно отметить возможность автоматизации выполнения ряда сложных и трудоемких задач разработки, например, следующих.

- Обеспечение более высокого качества системы.
  - Автоматизированная проверка полноты и непротиворечивости набора требований, проверка выполнения нужных пользователям свойств при помощи аналитической верификации или проверки на моделях (model checking).
  - Автоматизированное построение набора тестов, проверяющих работу системы в большом количестве разнообразных ситуаций.
- Обеспечение более высокой динамики разработки.
  - Генерация первых вариантов исходного кода системы из формальных спецификаций.
  - Генерация прототипов, симуляция работы системы для более быстрого получения отзывов пользователей о ней.

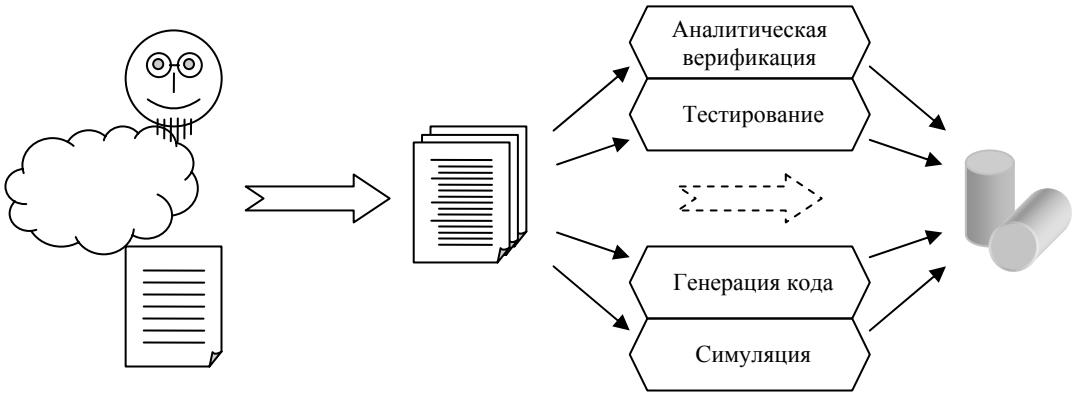


Рисунок 4. Формальные модели позволяют автоматизировать решение ряда задач разработки ПО.

Идеальным решением было бы использовать одну и ту же формальную модель для решения всех этих задач, как это изображено на Рис. 4. Однако пока не существует практически применимых в широких пределах методов, которые позволяли бы решать все перечисленные задачи на основе одной модели. Поэтому приходится использовать различные формализмы для решения разных задач, применяя уже имеющиеся частные, но зато достаточно эффективные методы.

В силу сказанного выше при проведении формализации на нынешнем этапе развития технологий нужно заранее определить ту задачу, которую мы считаем наиболее важной, и использовать формализм, позволяющий наиболее эффективно решить именно эту задачу.

- Для автоматической верификации и проверки на моделях удобна комбинация из расширенного конечного автомата, дающего детализированное представление системы, и временных логик, в которых записываются ее абстрактные свойства.
- Для построения тестов хорошо использовать контрактные спецификации компонентов системы и автоматные модели их тестирования.

- Для генерации кода чаще всего используются схемы структур данных или классов системы и исполнимые модели в виде взаимодействующих автоматов или процессов.
- Для прототипирования и симуляции также хорошо подходят модели на основе взаимодействующих автоматов или машин с абстрактным состоянием.

В дальнейшем мы рассмотрим процесс FOREST, нацеленный на создание формальных моделей, которые удобно использовать для автоматизированной разработки тестов.

Прежде чем перейти к описанию этого процесса, уместно провести обзор ряда стандартов и методов работы с требованиями, чтобы выделить основные задачи работы с требованиями в рамках разработки ПО и на их основе определить необходимые для построения формальных моделей виды деятельности.

Задачи работы с требованиями

Дисциплина, изучающая практически применимые методы работы с требованиями, *инженерия требований*, является очень широкой и разнородной областью, включающей рассмотрение огромного множества вопросов от синтаксиса документов, описывающих требования, до учета социокультурных особенностей пользователей при проведении семинаров и составлении списков вопросов. Поскольку нашей целью является выделение всех основных задач работы с требованиями, остановимся на рассмотрении стандартов IEEE 830 [1] и IEEE 1233 [2], определяющих необходимые характеристики требований, и раздела Свода знаний по программной инженерии (SWEBOK) [7], посвященного работе с требованиями. Стандарт SWEBOK появился не так давно и достаточно хорошо отражает текущее положение дел в этой области.

SWEBOK выделяет в работе с требованиями следующие четыре вида деятельности.

- ***Выделение требований (requirements elicitation)***, нацеленное на выявление всех возможных источников требований и ограничений на работу системы и извлечение требований из этих источников.
- ***Анализ требований (requirements analysis)***, целью которого обнаружение и устранение противоречий и неоднозначностей в требованиях, их уточнение и систематизация.
- ***Описание требований (requirements specification)***. В результате этой деятельности требования должны быть оформлены в виде структурированного набора документов и моделей, который может систематически анализироваться, оцениваться с разных позиций и в итоге должен быть утвержден как официальная формулировка требований к системе.
- ***Валидация требований (requirements validation),*** которая решает задачу оценки понятности сформулированных требований и их характеристик, необходимых, чтобы разрабатывать ПО на их основе, в первую очередь, непротиворечивости и полноты, а также соответствия корпоративным стандартам на техническую документацию.

Стандарты IEEE 830 и IEEE 1233 описывают как раз те характеристики, которые должны иметь требования, чтобы их можно было с успехом использовать для разработки ПО, и которые должны проверяться во время валидации по SWEBOK.

Первый стандарт перечисляет следующие необходимые свойства требований к ПО.

- Корректность, под которой имеется в виду адекватность.
- Однозначность.
- Полнота.
- Непротиворечивость или согласованность.
- Ранжированность по важности для пользователей и стабильности.
- Проверяемость.
- Удобство внесения изменений.
- Прослеживаемость.

Второй стандарт, IEEE 1233 определяет характеристики требований к более общим программно-аппаратным системам, во многом похожие на перечисленные выше.

- Характеристики набора требований в целом.
  - Уникальность каждого требования, отсутствие смысловых пересечений с другими.
  - Четкое описание взаимосвязей.
  - Полнота.
  - Непротиворечивость или согласованность.
  - Четкое описание контекста, в раках которого эти требования имеют смысл.
  - Удобство внесения изменений.
  - Поддержка вариантов и версий.
  - Подходящий для выбранных целей уровень детализации формулировок.
- Характеристики каждого отдельно требования.
  - Абстрактность, отсутствие излишних ограничений на реализацию.
  - Однозначность.
  - Прослеживаемость.
  - Проверяемость.

Систематизировав и уточнив свойства требований, представленные в обоих стандартах, можно переформулировать их следующим образом.

- Свойства, касающиеся связи требований с предметной областью.
  - ***Адекватность*** — соответствие сформулированных требований всем аспектам потребностей и ожиданий пользователей (тех, кто непосредственно будет решать свои задачи с помощью создаваемой системы), а также интересам всех остальных заинтересованных лиц (тех, чьи интересы только затрагиваются ее работой — контролирующие и общественные организации, партнеры и клиенты организации-заказчика и пр.).
    Мы трактуем адекватность в самом широком смысле, шире, чем она понимается в указанных стандартах, как всестороннее соответствие сформулированных требований тому, что действительно хочется иметь в создаваемой системе. В это понятие входит как обычная адекватность — соответствие формулировок требований нуждам пользователей — так и ряд других свойств, часто рассматриваемых в стандартах отдельно от адекватности, например, следующих.
    - Четкое определение контекста использования системы и области действия сформулированных ограничений и правил.
    - Правильно выбранный уровень абстракции, позволяющий определить реальные нужды и не фиксировать возможные детали проектных решений.
    - Учет приоритетности различных требований для пользователей.
    - Учет вероятностей возникновения изменений в требованиях.
    - Внешняя полнота требований, т.е. учет всех потребностей и всех интересов, которые было решено рассматривать.
    - Выделение связей и зависимостей между требованиями, основанных на свойствах предметной области.
- Свойства представления требований самих по себе (внутренние).
  - ***Однозначность*** — одинаковость понимания формулировок требований экспертами в соответствующей предметной области.
  - ***Полнота*** (внутренняя) — охваченность в описании требований всех аспектов и ситуаций, возможных в рамках описанного контекста работы системы.
  - ***Непротиворечивость*** — согласованность описаний требований друг с другом, отсутствие противоречий и расхождений между ними.

- ***Минимальность*** — невыводимость одних требований из других на основе формальной логики, однократная формулировка каждого нужного ограничения и отсутствие смысловых пересечений между разными требованиями.
- ***Систематичность*** — представление в виде системы с четко выделенными атрибутами (идентификаторами, приоритетами, изменчивостью, рисками и пр.) и ясным описанием взаимосвязей и зависимостей между ними.

- Свойства, касающиеся использования требований в процессе разработки (внешние).
  - ***Проверяемость*** — возможность для человека, обладающего определенными навыками, однозначно установить в каждой затрагиваемой требованием ситуации, выполнено оно или нарушено.
  - ***Прослеживаемость*** — возможность установления связей между требованиями и их источникам, с одной стороны, а также с разделами и элементами возникающих при разработке текстов программ, документов и моделей, с другой стороны.
  - ***Модифицируемость*** — возможность удобного и эффективного внесения изменений в сформулированные требования в ходе процесса разработки, включая поддержку различных их версий и конфигураций, а также управление запросами на изменения.

Идеальный процесс работы с требованиями должен обеспечивать все их необходимые свойства. Соответственно, в его состав должны входить деятельности, нацеленные на их обеспечение. Задачи, решаемые в ходе каждой деятельности, должны быть четко определены и отделены от задач других деятельностей.

Ниже представлено описание деятельностей такого процесса, как оно видится авторам. По сравнению со SWEBOK в нем больше внимания уделено разграничению разных задач работы с требованиями.

- ***Выделение требований (requirements elicitation).***
  Эта деятельность нацелена на выявление полного списка требований, их извлечение из всех доступных источников. Она состоит из трех более мелких деятельностей.
  - ***Определение источников требований.*** В его ходе выявляются все заинтересованные лица, документы, эксперты, существующие системы, которые могут служить источниками информации для составления требований к рассматриваемой системе, а также источниками ограничений на ее работу.
  - ***Извлечение требований*** — получение формулировок требований и ограничений из бесед с экспертами или заинтересованными лицами, наблюдением за работой пользователей, анализа документов, отраслевых и корпоративных стандартов, реинжиниринга существующих систем и анализа интерфейсов систем, с которыми придется взаимодействовать, и т.д.
  - ***Согласование требований*** — разрешение противоречий и несогласованности между требованиями, полученными из разных источников. Проводится на основе активного выяснения причин этих противоречий, быть может, более глубоких ограничений, видимых разными участниками по-разному, а также определения приемлемых для заинтересованных лиц компромиссов. Может выполняться при помощи прямого сопоставления разных точек зрения, совместных семинаров, определения приоритетов и пр.

- ***Систематизация требований (requirements analysis).***
  Целью этой деятельности является построение целостного набора, включающего все выделенные требования, определение всех существенных взаимоотношений и связей между ними. Такая система требований необходима для решения практически всех связанных с ними задач разработки ПО.
  В ходе систематизации приходится постоянно обращаться к источникам требований за прояснением отдельных вопросов, разрешением противоречий и т.п.
  В целом, эта деятельность соответствует анализу требований по SWEBOK, однако термин

«анализ» имеет слишком большую смысловую нагрузку, в частности, может означать как систематизацию, так и оценку свойств требований или их согласование, поэтому мы решили использовать термин, наиболее хорошо отражающий основную решаемую в ходе данной деятельности задачу.

- ***Описание требований (requirements specification).***
  В результате этой деятельности появляются различные представления требований в виде документов или моделей, предназначенные для решения определенных задач в рамках работы с требованиями и разработки ПО вообще.
  Часто для решения отдельной задачи нужно уметь строить специальное, удобное именно для ее решения представление требований.
  Описание требований в идеале строится на основе системы, полученной в результате предыдущей деятельности. На практике же их тяжело отличить друг от друга — в ходе систематизации всегда создается одно или несколько представлений требований.
- ***Валидация требований (requirements validation).***
  Эта деятельность направлена на проверку адекватности требований в широком смысле. В ее ходе, как и в ходе выделения требований, происходит взаимодействие с внешним миром — источниками требований, в том числе заинтересованными лицами.
  Выявленные в ходе валидации неадекватные требования должны быть уточнены или исправлены в ходе дополнительных работ по систематизации.
  Обычно термин «валидация» употребляется для названия деятельности, связанной именно с проверкой внешней правильности, адекватности чего-то: документов, программных решений, программно-аппаратных систем. Для проверки внутренней корректности — однозначности, непротиворечивости и пр. — используется термин «верификация». Эти две деятельности, несмотря на, казалось бы, общую задачу — проверку чего-то, — очень сильно отличаются по техникам их выполнения. Исходя из этих соображений, мы разделили валидацию требований по SWEBOK на их валидацию в указанном выше смысле и верификацию.

Рисунок 5. Схема потоков данных в идеальном методе работы с требованиями.

- ***Верификация требований (requirements verification).***
  Цель этой деятельности — проверка перечисленных выше внутренних свойств набора требований, т.е. его однозначности, полноты, непротиворечивости, минимальности и систематичности.
  Найденные дефекты должны быть устранены с помощью работ по систематизации.

Очень часто верификация проводится уже при систематизации требований — тяжело свести требования в единую систему, если они двусмысленны или рассогласованы. Однако бывает необходимо проводить и отдельную проверку того, что требования сформулированы аккуратно.

- ***Обеспечение внешних свойств требований (requirements usability assurance).***
  Эта деятельность служит для обеспечения трех внешних свойств требований.
  На практике большая часть работ, необходимых для решения этих задач выполняется в процессе систематизации требований.
  - ***Обеспечение прослеживаемости*** — определяет процедуры прослеживания требований в документах, моделях и коде, вырабатываемых при создании системы. Основано, чаще всего, на выделении элементарных требований и присвоении им идентификаторов.
  - ***Обеспечение проверяемости*** — включает установление возможности проверки требований и разработку процедур их проверки, которые могут быть представлены как выполнение тестов, проведение инспекций, проведение формальной верификации части требований и пр. Нужно также определять критерии полноты проводимых проверок, чтобы выполняющие их и руководители проекта четко осознавали, что именно проверено, а что еще нет — в этом плане эта деятельность опирается на прослеживаемость требований.
  - ***Обеспечение модифицируемости***. Определяет процедуры внесения изменений в требования, их согласования и утверждения, а также процедуры определения версий и конфигураций набора требований. Основывается на систематичности требований и их прослеживаемости, наличии уникальных идентификаторов и внесении всех документов, содержащих различные представления требований в среду конфигурационного управления.

На Рис. 5 показана схема обмена информацией между деятельностями представленного процесса. Поскольку эта схема похожа на букву ‘Ф’, будем называть его *Ф-процессом*.

Анализ ряда методов работы с требованиями

Посмотрим, как выделенные задачи работы с требованиями решаются в рамках ряда широко используемых методов разработки ПО. В качестве таких методов рассмотрим Унифицированный процесс разработки Rational (RUP), метод, лежащий в основе инструмента управления требованиями DOORS, метод работы с требованиями CORE, метод разработки RAISE и методология нечетких систем (Soft System Methodology, SSM). Эти методы выбраны для анализа, поскольку они достаточно сильно отличаются по базовым идеям и используемым в них техникам.

- *Унифицированный процесс разработки Rational (RUP)* [8,9] является одним из самых широко используемых полномасштабных процессов разработки ПО.
  Он разработан в Rational Software, ставшей впоследствии подразделением IBM, на основе модели процесса разработки ПО, первоначально созданной Айваром Джекобсоном (Ivar Jackobson) в компании Ericsson в конце 70-х годов XX века, а затем развитой им же в полномасштабный процесс разработки под названием Objectory в конце 80-х.
  Базовая идея RUP состоит в том, что процесс разработки ПО должен быть приспособлен к эффективной работе в постоянно меняющемся окружении и производить ПО, решающее проблемы пользователей, даже если они менялись в ходе разработки. Достигать этого предлагается за счет отслеживания происходящих изменений и их постепенного включения в контекст разработки с помощью следующих основных техник.
  - Все деятельности *итеративны* — они решают поставленные задачи не сразу, а постепенно, в несколько итераций. При этом возникает возможность учесть изменения в окружении и в понимании задач, происходящие во время разработки.

- Основой разрабатываемой системы, стержнем, на который нанизываются ее подверженные частым изменениям элементы и который позволяет ей, несмотря на эти изменения, сохранять целостность, является *архитектура ПО*.
- Техникой описания требований, достаточно удобной, чтобы фиксировать все происходящие изменения, считаются *варианты использования*. Каждый вариант использования задает набор сценариев взаимодействия системы с ее окружением, предназначенных для решения какой-то одной задачи, важной для пользователей.
- Практически все более частные техники, используемые в процессе разработки ПО, связаны с созданием и использованием *моделей* различных аспектов создаваемой системы. Модели используются для того, чтобы оценивать последствия предлагаемых решений и возникающих изменений и связанные с ними риски до воплощения этих решений и принятия изменений в полном объеме. Это позволяет значительно снизить затраты на выбор решения, удовлетворяющего нужды пользователей в максимальной степени.

- *Метод DOORS* лежит в основе использования инструмента управления требованиями *DOORS* [10,11] — динамической объектно-ориентированной системы управления требованиями (Dynamic Object-Oriented Requirements System).
  Этот метод вместе с самим инструментом был создан Ричардом Стивенсом (Richard Stevens) в 80-х годах XX века на основе опыта работы в Европейском космическом агентстве. Для распространения этого инструмента он основал компанию QSS, впоследствии приобретенную Telelogic.
  В основе этого метода лежит представление о том, что требования к сложной системе очень сложны, сильно взаимосвязаны, разнородны и в своем исходном виде не могут быть использованы для разработки. Чтобы стать ее частью, они должны быть подвергнуты поэтапному уточнению, проводимому несколько раз, а также формулироваться совместно с правилами их проверки. При этом крайне важно сохранить всю историю проведенных уточнений и построенных тестов, явно зафиксировать все привлекаемые для этого дополнительные свойства и ограничения окружения, чтобы быть в состоянии анализировать влияние отдельных допущений, возможных изменений требований и покрытие требований в ходе разработки и верификации системы.
  Основной техникой, используемой в DOORS, является представление требований в виде структурированного набора текстовых и графических документов, моделей и других объектов, имеющих расширяемый набор атрибутов и связанных размеченными связями. Б*о*льшая часть работ с требованиями выполняется с помощью анализа и модификации этих атрибутов и связей.

- *Метод работы с требованиями CORE* (управляемое описание требований, Controlled Requirements Expression) [12,13] был разработан в конце 70-х годов XX века в одной из компаний, выполнявших проектные работы для Британского аэрокосмического агентства. Это один из первых методов, предназначенных именно для работы с требованиями в рамках разработки сложных систем.
  Основная его идея — необходимость учета различных точек зрения на систему при работе с требованиями и привлечения специальных техник для их согласования. Сначала все точки зрения разделяются на два больших класса: точка зрения людей, связанных с разработкой, развитием и поддержкой работоспособности системы, и точка зрения все остальных людей, затрагиваемых системой. Это разделение позволяет рассматривать проблемы, которые система решает, отдельно от используемых ею способов их решения.
  Техники, используемые в рамках CORE, основаны на иерархической декомпозиции интерфейсов и функций системы с одновременным прослеживанием того, как эти функции задействуются в различных сценариях ее работы (*системных транзакциях*).

- *Методология нечетких систем* (Soft System Methodology, SSM) [14,15,16] является методом анализа нечетких задач, используемым как для работы с требованиями к сложным

системам, так и для анализа их функционирования или управления изменениями в рамках такой системы. Она была создана Питером Чекландом (Peter Checkland) и Брайном Вильсоном (Brian Wilson) с коллегами на факультете системной инженерии университета Ланкастера на основе опыта совместных работ с British Aircraft в конце 70-х годов, участвовавшей, в частности, в создании самолета Конкорд.
Основой SSM является представление о наличии *нечетких систем* — систем, границы, задачи и само существование которых существенно зависят от наблюдателя. Например, большинство социальных систем нечетки — их границы и решаемые ими задачи не определены однозначно и видятся по-разному с разных точек зрения. Такие системы не имеют *четких задач* (hard problems), которые можно однозначно сформулировать и для которых имеются четкие критерии того, получено их решение или нет. Вместо этого в них возникают *проблемные ситуации*, которые характеризуются множественностью и неоднозначностью проблем, множественностью возможных выходов и отсутствием ясного представления о возможном решении.
Техники, используемые для анализа нечетких систем, основаны на построении субъективной целостной картины ситуации с определением конфликтующих интересов и целей. В дальнейшем это описание ситуации используется для формулировки различных точек зрения и соответствующих им критериев адекватности создаваемых решений.

- *Метод разработки RAISE* (строгий подход к промышленной программной инженерии, Rigorous Approach to Industrial Software Engineering) [17,18]. Это полностью формальный метод разработки ПО, созданный в рамках европейского проекта ESPRIT I в конце 80-х годов XX века для использования в индустриальных проектах. У истоков его стояли Дайнес Бьорнер (Dines Bjoerner) и Крис Джордж (Cris George). Он имеет много общего с другими формальными методами разработки ПО — Z [19], VDM [20], B [21] — и рассматривается здесь как их представитель.
Основная идея метода RAISE — необходимость полной формализации требований для возможности их адекватного включения в процесс разработки. Базовыми техниками RAISE являются определение формальных контрактов компонентов разрабатываемого ПО и их пошаговое уточнение при разработке. При этом формальные спецификации требований сначала представляются в наиболее абстрактной форме, а затем постепенно уточняются до такого состояния, когда на их основе можно создать (или автоматически сгенерировать) полностью исполнимый код.

Чтобы проанализировать эти методы, нужно четко понимать различие между общими понятиями *техники* и *метода*.

Под ***техникой*** мы понимаем деятельность, состоящую в выполнении определенной последовательности действий, нацеленных на решение одной вполне конкретной задачи. При этом часто ни одно из действий, предпринимаемых в рамках работы в соответствии с данной техникой, не имеет смысла выполнять отдельно.

Методом называется набор идей и техник, объединенных этими идеями и образующих единую структуру обработки информации, нацеленную на решение ряда связанных задач из некоторой области. Техники, используемые в рамках метода, можно применять отдельно или в рамках других методов для решения тех же задач. Все пять перечисленных выше подходов определяют методы работы с требованиями.

Процессом мы будем называть такой метод, который, во-первых, включает инструментальную поддержку ряда используемых в нем техник, и, во-вторых, предназначен для решения не просто группы, а всех значимых задач в некоторой четко определенной области. Из приводимой ниже таблицы следует, что процессами являются только первые два из пяти рассматриваемых методов.

Сравнение перечисленных методов приведено в Таблице 1. В ней они рассматриваются с точки общих зрения задач работы с требованиями, определенных в схеме Ф-процесса. Для каждой задачи перечисляются применяемые в рамках данных методов техники решения этой задачи. Эти

техники либо прямо указываются в литературе, описывающей соответствующие методы, либо взяты из имеющейся практики их применения. В описаниях различных методов одинаковые или очень близкие по смыслу совершаемых в них действий техники часто называются по-разному, поэтому при сравнении мы постарались привести их названия к некоторой общей системе, позволяющей соотнести рассматриваемые методы друг с другом.

Таблица 1 показывает, что только метод работы с требованиями в рамках RUP и метод, лежащий в основе DOORS, определяют полное семейство техник, способное решить все задачи работы с требованиями. Методы CORE и SSM фокусируют внимание на выделении, описании и уточнении требований, и ничего не говорят об их верификации, обеспечении проверяемости и модифицируемости. Метод RAISE, наоборот, делает акцент на верификации требований, не определяя детально способов их выделения.

		RUP	**DOORS**	**CORE**	**SSM**	**RAISE**
Выделение требований	Определение источников	Анализ окружения и целей системы	Список ролей, анализ окружения	Анализ документов и целей системы	Анализ окружения и целей системы	—
	Извлечение требований	Семинары, раскадровка, ролевые игры, мозговые штурмы, интервью, моделирование сценариев работы, моделирование структуры данных, моделирование деятельностей	Моделирование сценариев, анализ документации, анализ существующих систем, прототипирование, интервью, опросы, семинары, мозговые штурмы	Анализ документов, интервью, моделирование точек зрения	Интервью, когнитивные техники, моделирование точек зрения, наблюдение, семинары, ролевые игры, прототипирование	Интроспекция, анализ документов
	Согласование требований	Семинары, мозговые штурмы, определение приоритетов, моделирование сценариев работы	Моделирование сценариев, семинары, определение статуса согласования	Моделирование точек зрения, уточнение требований	Моделирование точек зрения	—
Систематизация требований		Построение модели вариантов использования и концептуальной модели, уточнение моделей	Шаблоны требований, многоаспектная классификация, атрибутирование, моделирование сценариев, анализ размеченных связей	Иерархическая модель точек зрения, модель системных функций, модель системных транзакций	Построение целостной картины ситуации, выделение точек зрения, построение концептуальной модели	Система формальных моделей
Описание требований		Модель вариантов использования и концептуальная модель	Структурированный текст с размеченными связями между его элементами	Иерархическая модель точек зрения, модель системных функций, модель системных транзакций	Целостная картина ситуации, концептуальная модель	Система формальных моделей
Валидация требований		Прототипирование, рецензирование пользователями и экспертами, раскадровка	Прототипирование, рецензирование пользователями и экспертами	Рецензирование, интроспекция, интервью	Интервью, когнитивные техники, прослеживание следствий, прототипирование	Рецензирование, анализ сценариев

		RUP	DOORS	CORE	SSM	RAISE
Верификация требований	Проверка однозначности	Рецензирование аналитиком, автоматизированный статический анализ	Рецензирование аналитиком	Рецензирование аналитиком, уточнение	—	Рецензирование аналитиком, автоматизированный статический анализ
	Проверка полноты	Рецензирование аналитиком	Рецензирование аналитиком	Рецензирование аналитиком	—	Рецензирование аналитиком, формальный анализ
	Проверка непротиворечивости	Рецензирование аналитиком, автоматизированный статический анализ	Рецензирование аналитиком	Рецензирование аналитиком	—	Рецензирование аналитиком, (автоматизированный) формальный анализ
	Проверка минимальности	Рецензирование аналитиком	Анализ размеченных связей, рецензирование аналитиком	—	—	Рецензирование аналитиком, формальный анализ
	Проверка систематичности	Рецензирование аналитиком	Анализ размеченных связей	Рецензирование аналитиком	Рецензирование аналитиком	Рецензирование аналитиком
Обеспечение внешних свойств требований	Обеспечение прослеживаемости	Идентификация вариантов использования, матрица прослеживания	Идентификация, уточнение требований, анализ размеченных связей	Иерархическая декомпозиция точек зрения	Декомпозиция задач на целостной картине	Матрица прослеживания
	Обеспечение проверяемости	Матрица прослеживания, разработка тестовых вариантов	Определение стратегии проверки и критериев соответствия, определение статуса проверки	—	—	—
	Обеспечение модифицируемости	Конфигурационное управление, БД запросов на изменение	Конфигурационное управление, БД запросов на изменение, анализ размеченных связей, определение статуса изменений	—	—	Конфигурационное управление

Таблица 1. Техники работы с требованиями, используемые в методах RUP, DOORS, CORE, SSM и RAISE.

Техники выделения требований

Каждая деятельность в рамках Ф-процесса может быть выполнена с помощью разнообразных техник, что продемонстрировано и в приведенной выше таблице. Эта таблица, однако, содержит далеко не все такие техники. В качестве примера набора возможных техник для выполнения одного вида деятельности приведем обзор *техник выделения требований*. Основная задача таких техник — выявить потребности и интересы всех связанных с разрабатываемой системой лиц. Все они так или иначе связаны с более сложной областью *выделения знаний*.

Техник выделения требований известно достаточно много. Многие из них применяются в рамках нескольких методов. Иногда использующий некоторую специфическую технику метод упоминается в литературе под видом этой техники. Сами же методы чаще всего включают также техники систематизации и описания требований, реже — техники их валидации и верификации.

Все приведенные ниже техники могут использоваться как собственно для выделения требований, так и для проверки адекватности уже сформированных требований. Применение их в этих двух ситуациях часто отличается только отсутствием или наличием исходных документов и моделей, содержащих уже определенные требования пользователей, заказчиков и пр.

Техники выделения требований можно классифицировать следующим образом.

- *Аналитические техники.* В их основе лежит анализ каких-либо документов, знаний, систем или деятельностей.
    - ***Размышление (introspection)*** [22] — аналитик размышляет над возможными функциями системы и выбирает те, которые выглядят достаточно привлекательными и полезными.
    - ***Анализ документов и моделей (review, analysis)*** — анализируются нормативные документы, документы, описывающие реальность или пожелания и проблемы, модели, описывающие предметную область. Из них выделяются потребности, возможные функции и критерии их оценки.
    - *Анализ имеющихся аналогичных систем.*
        - ***Анализ выборки данных (sampling)*** — анализируются большие массивы имеющихся данных (отчетов, внутренних данных системы, входной информации и пр.), выделяются сущности, их атрибуты и связи.
        - ***Анализ функций*** — анализируется работа одной или нескольких имеющихся систем, выделяются выполняемые ими операции и возможные правила их выполнения.
- *Техники, основанные на общении.* В их основе лежит общение между аналитиком и другими людьми, нацеленное на выявление потребностей заинтересованных лиц и ограничений предметной области.
    - *Индивидуальные техники.* В них используется только индивидуальное общение аналитика с другими людьми.
        - ***Интервью (interview)*** — заинтересованное лицо отвечает на вопросы аналитика, часть их которых подготовлена заранее, другая часть возникает в процессе диалога.
        - ***Опрос (survey)*** — ряд заинтересованных лиц отвечает на вопросы из заранее подготовленного списка (вопросника, questionnaire).
    - *Групповые техники.* В них используется групповое общение, повышающее отдачу каждого сеанса, но и приводящее к возникновению конфликтов, различных позиций и пр., и поэтому нуждающееся в контроле и управлении.
        - ***Групповое интервью (group interview)*** или ***фокус-группа (focus group)*** — группа экспертов собирается и обсуждает совместно с аналитиком возможные функции и требования. Обсуждение направляется вопросами аналитика по заранее подготовленному сценарию.
        - ***Мозговой штурм (brainstorming session)*** — несколько человек, включая аналитиков и заинтересованных лиц, некоторое время выдумывают различные возможности, которые могли бы быть полезны, не ограничивая воображение. Есть ведущий, который следит за соблюдением правил: давать всем высказаться, не перебивать, не оценивать идеи, соблюдать регламент. После этого группа экспертов сортирует и оценивает полученные идеи.
        - ***Семинар (workshop)*** [5] — группа лиц, включающая аналитиков, экспертов, пользователей, по несколько часов в течение 1-5 дней обсуждает проблемы, потребности, возможные функции, пытаясь составить их полный и согласованный список. У семинара есть ведущий, который направляет его ход в соответствии с заранее определенным планом, сохраняя нейтральную позицию при возникновении разногласий и помогая их разрешить.
        - ***Метод Дельфи (Delphi method)*** [23,24]— опрос группы экспертов на одну тему. Каждый из них сначала готовит документ с изложением своего мнения. Затем каждый из полученных документов передается для анализа и замечаний другому эксперту. Такая передача может повторяться несколько раз.

- ***Ролевая игра (role playing game)*** — участники группы разыгрывают возможные сценарии работы, каждый играет свою роль. Выявляемые в ходе игры проблемы, потребности, ограничения фиксируются.

- *Демонстрационные техники.* Они используют демонстрацию одному или группе экспертов моделей системы или возможных сценариев работы системы для получения от них откликов и замечаний, которые обычно тут же фиксируются.
  - ***Раскадровка (storyboard)*** — сценарий работы в некоторой ситуации расписывается по отдельным действиям и представляется пользователям и экспертам. Представляться он может в виде описания действий, набора рисунков, анимации, и пр. Собираются замечания и поправки, высказанные по поводу такого сценария.
  - ***Демонстрация прототипа (prototype demo)*** — демонстрируется работа прототипа или отдельных модулей системы в определенной ситуации. Собираются замечания пользователей и экспертов.
  - Техники, использующие в качестве основы общения некоторые ***модели (model based techniques)***.
    Таких техник много, любой вид моделей, особенно графических или симуляционных, можно использовать для общения с экспертами и пользователями и фиксации их замечаний к такой модели.
    В качестве моделей часто используются описания сценариев работы системы, варианты использования, диаграммы потоков данных, концептуальные модели данных системы.
    Отдельно можно упомянуть модели, фиксирующие выделяемые цели операций системы и связанные с ними техники (goal-driven, см. [25,26]), а также модели, фиксирующие различные точки зрения (viewpoint-based, см. [27-29]).
- *Когнитивные техники.* Это специализированные техники, основанные на некоторых положениях когнитивной психологии. Исторически они возникли как приемы выявления структур мышления у людей с психическими отклонениями. После достижения ряда успехов в этой области техники такого рода стали применятся и для выявления мыслей и знаний в более широком контексте, в частности, для выделения требований.
  - ***Анализ задач (task analysis)*** [30-35] — анализируются задачи, решаемые пользователем, и операции, предпринимаемые им для их решения. Целью анализа является определение информации, необходимой для выполнения каждой операции, и взаимосвязей между различными используемыми в процессе решения сущностями. Обычно проводится аналитиком, присутствующим при работе пользователя и задающим вопросу по ходу его работы.
  - ***Анализ протоколов (protocol analysis)*** [36-39] — пользователь или эксперт проговаривает и вслух все выполняемые действия при конкретном или абстрактном сценарии. При этом приводятся их обоснования и возможные варианты.
  - ***Устойчивые решетки*** или ***решетки Келли (repertory grids)*** [40,41]. Выбирается набор объектов предметной области, для которых каждый эксперт из группы придумывает систему атрибутов и их значений. Потом эксперты пытаются найти наиболее близкие атрибуты в своей системе к атрибутам в системе другого эксперта, а также оценить значения атрибутов в чужих системах. Есть еще несколько разновидностей этой техники [6].
  - ***Сортировка карточек (card sorting)*** [42]. На карточки заносятся названия объектов предметной области. Обычно один объект присутствует на одной карточке, но в некоторых случаях используется модификация — один объект

может быть на нескольких карточках. После этого каждому из нескольких экспертов выдают полный набор карточек и просят разделить карточки на группы по каким-то признакам. Затем каждый формулирует обоснование своей группировки.

- ***Лестница (laddering)*** [43,43] — используется иерархическая система вопросов-зондов (probes) для выявления знаний опрашиваемого о предметной области.
- ***Масштабирование близости (proximity scaling)*** [45,46]. Определяется набор объектов предметной области, список которых выдается экспертам. Те должны определить набор характеристик для их описания и степень близости этих объектов друг к другу по выбранному набору характеристик.

- *Контекстуальные техники* [22,47,48]. В их основе лежат методы этнометодологии и культурологии. В этих техниках используется наблюдение за действиями пользователей и экспертов в том окружении, в котором надо будет работать будущей системе. При этом пытаются учитывать национальный и культурный контексты их поведения.
  Учет этих факторов позволяет как задавать правильные вопросы (на один и тот же по форме вопрос представители разных культур могут дать совершенно разные ответы при очень схожих обстоятельствах), так и правильно интерпретировать получаемые ответы.
  В качестве вспомогательных элементов в таких техниках используется следующее.
  - Анализ разговоров и речи во время интервью или совещаний.
  - Анализ невербального поведения.
  - Запись поведения человека на видео и последующий его детальный анализ.
  - Запись и анализ физиологических параметров (пульс, давление, электропроводность кожи и пр.).

Процесс формализации требований FOREST

В данном разделе описан процесс работы с требованиями *FOREST*, который обобщает использование требований в проектах группы UniTESK. Основная цель этого процесса — построение формального описания требований к программной системе, адекватно отражающего их исходное содержание. В данный процесс включено также использование формальной модели требований в качестве основы для построения тестов. Название процесса FOREST расшифровывается как FOrmal REquirements Specification and Testing и представляет основные его задачи:

- построить формальную спецификацию требований к некоторой системе;
- разработать на ее основе тестовый набор для тестирования соответствия этим требованиям.

Процесс FOREST применялся на практике во многих проектах:

- в проекте по разработке тестовых наборов для сетевого протокола Интернет IPv6 [49];
- при разработке прототипа тестового набора для протокола контроля прав доступа к мультимедийной информации IPMP-2 [50];
- при тестировании платформы интеграции в компании-операторе мобильной связи;
- при разработке тестового набора для части базовой библиотеки Java;
- в проекте по разработке набора тестов для проверки соответствия базовому стандарту Linux (Linux Standard Base, LSB [51,52]).

В этих проектах он показал себя достаточно зрелым и эффективным методом работы с требованиями.

Процесс FOREST используется преимущественно в проектах по разработке тестовых наборов. Это обусловило ряд его особенностей, которые выделяют его среди других процессов и методов работы с требованиями.

- В качестве источников требований используются хорошо структурированные документы, такие как международные и отраслевые стандарты, рекомендации международных организаций, технические задания.
  Такой выбор источников обусловлен тем, что процесс предназначен для формализации требований, что невозможно сделать, если они не представлены в некотором однозначном виде, и не достигнуто согласие об их понимании всеми заинтересованными сторонами.
- В процессе FOREST особый упор сделан на прослеживаемость требований. Во всех представлениях требований, которые разрабатываются в ходе выполнения работ процесса, устанавливаются указания на исходные требования.
  Прослеживаемость требований необходима для проверки корректности формальной спецификации, а так же для определения покрытия требований тестами.
- В качестве методологической основы для формальной спецификации используются программные контракты. Такая форма представления требований позволяет эффективно использовать спецификации в тестировании.

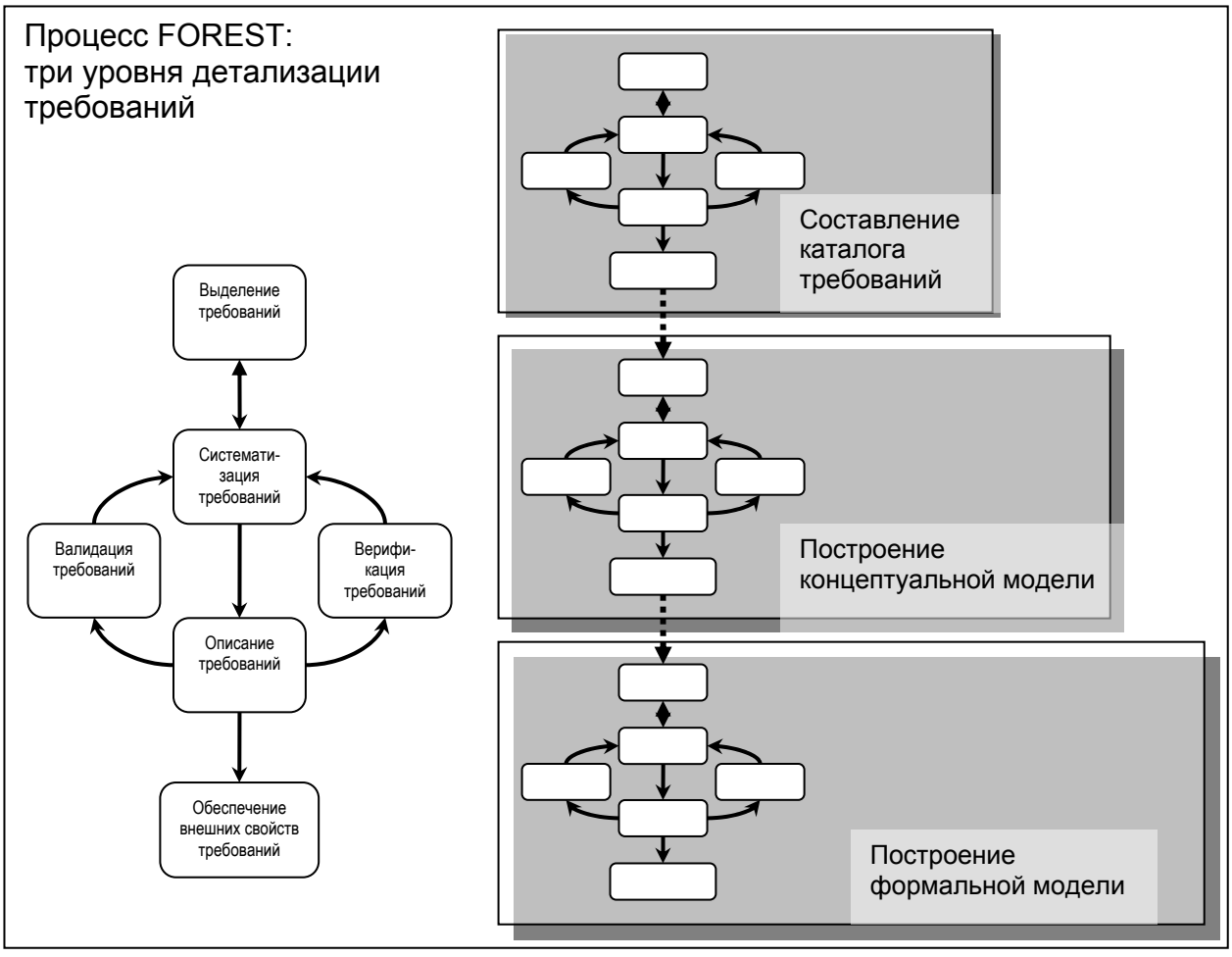


Рисунок 6. Процесс работы с требованиями состоит из нескольких взаимосвязанных этапов.

Обзор процесса FOREST

Процесс построен итеративно и состоит из трех основных этапов, соответствующих трем уровням детализации требований, при этом на каждый этап организован в соответствии с Ф-процессом и содержит некоторый замкнутый набор работ с требованиями. Трудоемкость выполнения этапов возрастает от первого к третьему. Результаты каждого этапа нужны для выполнения последующих, но могут быть использованы и отдельно. Поэтому при необходимости работа по процессу FOREST может быть остановлена после выполнения любого из этапов, если полученные

результаты уже можно применить в разработке ПО, а работы следующих этапов оказываются недостаточно эффективными в рамках конкретного проекта.

На первом этапе требования извлекаются из первичных документов и систематизируются. При этом происходит структуризация доступного знания о предметной области. В результате получается *каталог требований*, в котором требования сформулированы максимально однозначно, с каждым требованием сопоставлен уникальный идентификатор, требования классифицированы, и, возможно, установлены связи между отдельными требованиями. Каталог требований используется на последующих этапах. В соответствии с решаемыми задачами этот этап называется «Составление каталога требований».

Второй этап ориентирован на анализ требований. В качестве основного средства анализа используются концептуальные модели требований. Соответственно, этот уровень называется «Построение концептуальной модели». В ходе разработки и анализа концептуальных моделей происходит реструктуризация знания о предметной области, и производится проверка свойств требований, таких как адекватность, полнота, и т.п. Разработка модели опирается на каталог требований, построенный в процессе первого уровня.

Третий этап нацелен на представление требований в формальном виде. Этот уровень называется «Построение формальной модели». Требования из каталога записываются с использованием математического формализма контрактных спецификаций. Перевод требований из текстового представления в формальное облегчается тем, что знания о предметной области, были реструктурированы и проанализированы на первых двух этапах.

В последующих разделах мы подробнее рассмотрим каждый этап.

Формат описания этапов процесса

В начале описания каждого этапа процесса приводится его краткая характеристика в виде таблицы. Таблица имеет следующий формат.

Задачи этапа	Список задач работы с требованиями, решаемых на данном этапе. Список основан на задачах, представленных в схеме Ф-процесса.
Входные данные этапа	Список видов документов, которые служат в качестве источников информации для работ этапа.
Схема выполнения этапа	Схема выполнения этапа представляет собой диаграмму UML Activity, которая описывает последовательность выполнения его работ.
Результаты этапа	Список документов, в которых представлены результаты выполнения работ данного этапа.

Таблица 2. Формат краткого описания этапа процесса.

Следом за краткой характеристикой процесса идет описание работ процесса.

Составление каталога требований

На первом этапе производится выделение требования из различных источников и общего контекста предметной области. Несмотря на то, что в процессе FOREST в качестве источников требований выступают хорошо структурированные документы, извлечение требований в его рамках не исключает использования дополнительной литературы или обращения к экспертам и авторам технической документации.

Задачи этапа	• Выделение требований
	  o Определение источников
	  o Извлечение требований
	  o Согласование требований
	• Систематизация требований
	• Описание требований
	• Валидация требований
	• Верификация требований
	  o Проверка систематичности
	• Обеспечение внешних свойств требований
	  o Обеспечение прослеживаемости
	  o Обеспечение модифицируемости

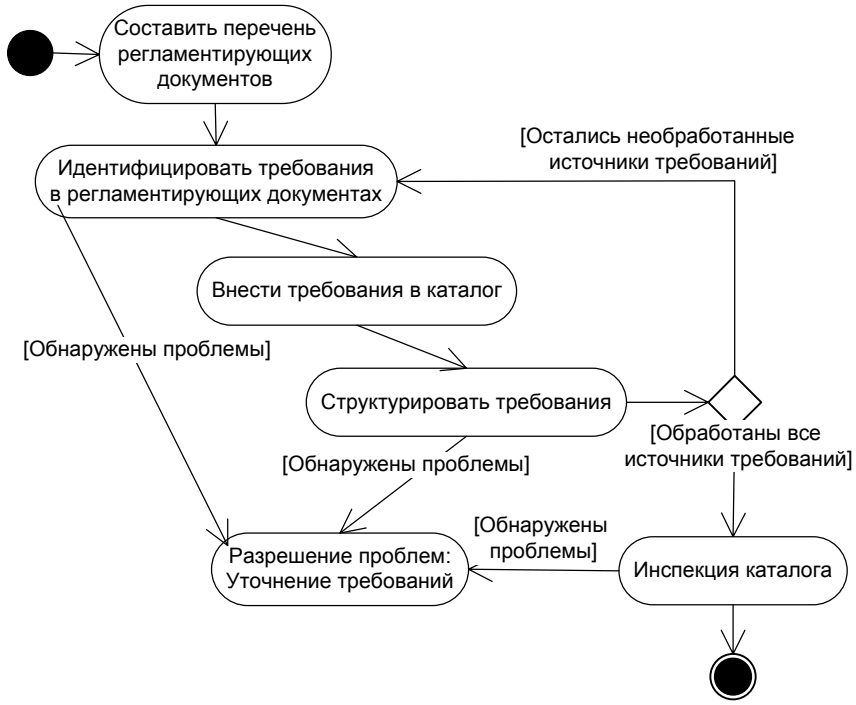
Входные данные этапа	Текстовая спецификация требований. Критерии классификации требований.
Схема выполнения этапа	
Результаты этапа	Каталог требований

Таблица 3. Краткое описание этапа «Составление каталога требований».

Составление перечня регламентирующих документов

Результаты рассматриваемого процесса используются, как правило, для тестирования соответствия спецификациям, поэтому в большинстве случаев в качестве источников требований используются устоявшиеся текстовые спецификации: техническое задание, отраслевой или международный стандарт, стандарты и рекомендации национальных или международных организаций. Определение множества источников начинается со спецификаций *целевой системы*, т.е. анализируемой или разрабатываемой в рамках данного проекта. Если эта система взаимодействует с другими системами или функционирует в рамках более общей системы, то в множество источников требований включаются спецификации смежных систем и объемлющей системы.

Помимо текстовых спецификаций требований в качестве их источников могут выступать разработчики спецификаций и эксперты в предметной области.

Рисунок 7. На первом этапе требования выделяются из различных источников.

Идентификация требований

В процессе FOREST требования извлекаются из технической документации, где они не всегда явно отделены от прочего текста — пояснений и вспомогательных описаний. Поэтому на данном этапе может выполняться работа по *идентификации требований* в источниках, задача которой — выделение тех частей текста, в которых сформулированы требования к целевой системе и ограничения на ее работу.

Для работы с текстовыми спецификациями в процессе используется специальная техника — *разметка текстовой спецификации*. В ходе идентификации требований аналитики выделяют в текстах спецификаций участки текста, которые, по их мнению, являются функциональными требованиями к реализации; выделенные участки текста помечаются, например, цветом. Благодаря разметке упрощается оценка полноты идентификации требований — требования, пропущенные аналитиками, могут находиться только в неразмеченной части текста.

Для прояснения требований используется техника представления поведения системы в виде сценариев поведения. В качестве сценариев могут использоваться диаграммы MSC или сценарные диаграммы UML, а также текстовые описания последовательности действий в неформальном виде. В тех случаях, когда не удалось обнаружить требования к некоторым возможным ситуациям или выявлены противоречия в соответствующих требованиях, можно передать сценарий поведения разработчикам спецификации или экспертам, чтобы они прояснили требования.

Внесение требований в каталог

Выделенные требования заносятся в каталог, который представляет собой базу данных или таблицу описаний требований. В нем с каждым требованием связан уникальный идентификатор, текст требования, ссылка на его источник и, возможно, какие-то дополнительные атрибуты.

При внесении требований в каталог может проводиться инспекция или рецензирование идентифицированных требований экспертами или аналитиками для проверки того, что заносимая туда информация действительно представляет собой требования.

Структурирование требований

Систематизация требований призвана разделять требования на группы «похожих» требований. В рамках данного процесса конечной целью является разработка формальной спецификации требований, поэтому систематизация должна выделять группы требований, которые формализуются сходным образом.

Систематизация основана на классификации требований. Разумеется, классификация зависит от предметной области, но есть ряд базовых критериев, которые, на наш взгляд, должны быть отражены в каждой классификации:

- **Ограничения поведения и ограничения целостности**. Под ограничениями поведения мы понимаем требования к поведению системы в определенных условиях или некоторой ситуации. Под ограничениями целостности мы понимаем требования к связям между элементами системы, к структуре ее данных и т.п., которые должны выполняться на протяжении всего времени функционирования системы или почти во всех ситуациях, возникающих при работе системы.
- **Объект требования**. Требования в спецификации предъявляются к поведению различных компонентов или объектов системы. Требования, описывающие поведение одного объекта удобно выделять в одну группу, это облегчает использование каталога требований в последующих работах.
- **Степень обязательности**. Во многих стандартах описание поведения системы включает обязательные функции, которые должны быть представлены в каждой реализации стандарта, и необязательные для реализации (optional). В зависимости от особенностей стандарта, этот критерий делит требования на два или более классов. Каждый класс соответствует определенному уровню обязательности.
- **Позитивные и негативные требования**. Под позитивными требованиями мы понимаем требования, предписывающие реализации совершить определенные действия, например, отправить сообщение в сеть или возвратить некоторый код ответа. Под негативными требованиями мы понимаем требования, которые запрещают реализации совершать определенные действия. Например, спецификация протокола ICMPv6 запрещает реализации высылать сообщения определенного вида. Этот критерий особенно важен при формализации, так как позитивные и негативные требования формализуются различным образом. Данный критерий делит требования на два класса.
- **Требования к реализации и требования к окружению**. Под требованиями к окружению понимаются ограничения, которые должны выполняться окружением при использовании реализации. Если окружение не удовлетворяет таким ограничениям, то поведение реализации не определено.

В конкретных случаях применения процесса FOREST могут использоваться дополнительные критерии классификации требований, основанные на предметной области.

Проверка полноты извлечения требований

Для обеспечения полноты каталога требований в данном процессе используются разметка текстовых спецификаций и инспекция требований экспертом в предметной области.

Инспекция каталога

Инспекция каталога производится экспертами в предметной области с целью верификации и валидации требований в каталоге. В случае выявления проблем, например, неадекватность сформулированных в каталоге требований, противоречия или неполнота, эксперты и аналитики устраняют выявленные проблемы. При этом возможно выделение дополнительных требований или обращение к авторам спецификации за разъяснениями.

Связь с Ф-процессом

Предложенный процесс можно рассматривать как специализацию общего Ф-процесса.

В отображении работ первого этапа на Ф-процесс, представленном на Рис. 8, заретуширована верификация требований, так как на первом этапе ее трудно проводить из-за их значительного

объема и того, что требования представлены по большей части в виде текста на естественном языке. Специально для решения задач верификации и валидации требований в процесс FOREST введен этап построения концептуальной модели требований (см. далее).

Рисунок 8. Отображение работ первого этапа на Ф-процесс

Построение концептуальной модели требований

При работе с требованиями анализ требований происходит практически все время. Вдумчивый аналитик волей-неволей оценивает требования уже на этапе их извлечения и каталогизации. В процессе формализации требований они подвергаются очень тщательному анализу на предмет полноты и непротиворечивости. Тем не менее, мы выделяем специальный этап анализа и валидации требований, который стоит между каталогизацией требований и их формализацией.

После того, как текст спецификации был размечен, требования идентифицированы и занесены в каталог, возникает вопрос: как проверить адекватность требований и выполнение внутренних свойств в наборе выделенных требований?

Задачи этапа	• Выделение требований
	  o Согласование требований
	• Систематизация требований
	• Описание требований
	• Валидация требований
	• Верификация требований
	  o Проверка однозначности
	  o Проверка полноты
	  o Проверка непротиворечивости
	  o Проверка минимальности
	  o Проверка систематичности
	• Обеспечение внешних свойств требований
	  o Обеспечение прослеживаемости
	  o Обеспечение модифицируемости

Входные данные этапа	Каталог требований и исходная спецификация требований

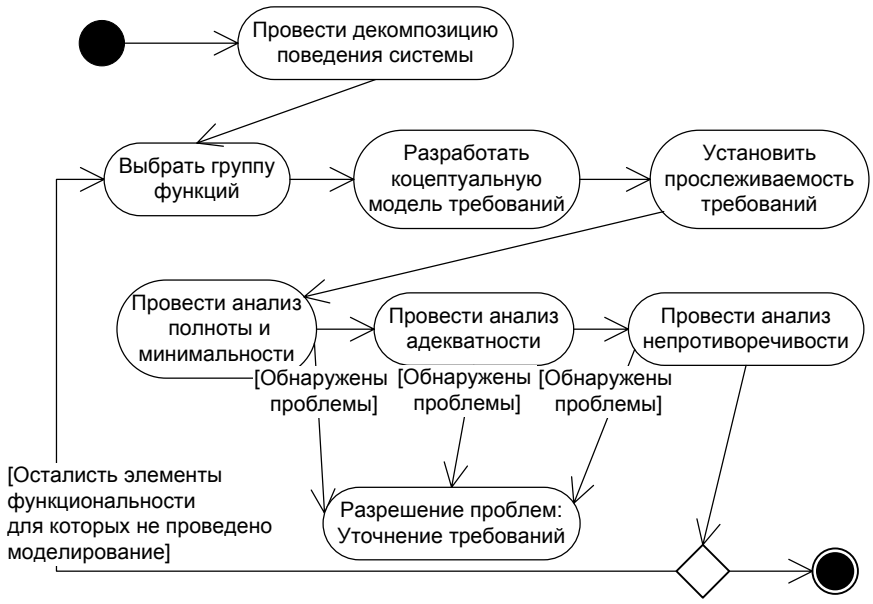
Схема выполнения этапа	
Результаты этапа	Концептуальная модель требований

Таблица 4. Краткое описание этапа «Построение концептуальной модели требований».

В рамках процесса FOREST для решения задач валидации и верификации требований используются их концептуальные модели.

Концептуальная модель описывает внешне наблюдаемое поведение системы как операции над некоторым набором абстрактных компонентов и объектов, составляющих систему. Эти компоненты используются только для моделирования поведения и могут никак не соотноситься с разбиением самой системы на компоненты. Разные ее реализации могут использовать собственные структуры данных, объекты и компоненты, с условием, что внешне наблюдаемое поведение будет соответствовать описанному в модели.

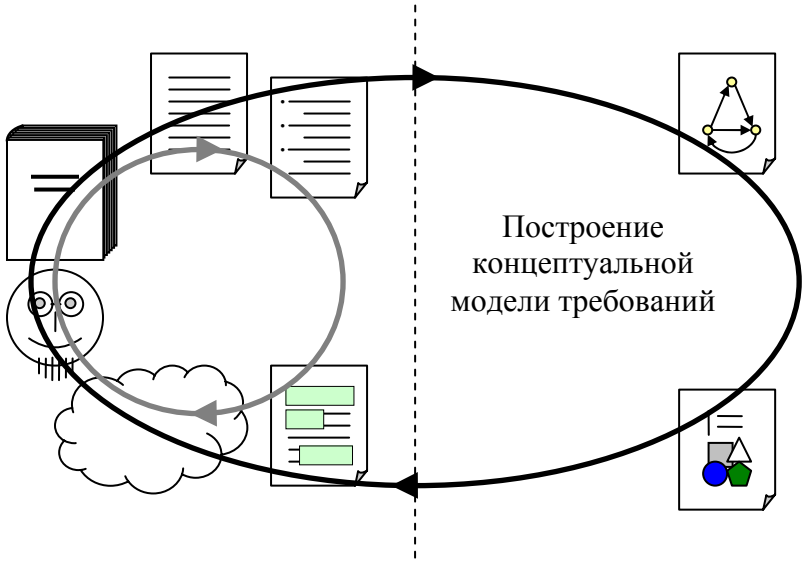


Рисунок 9. На втором этапе проводится анализ свойств требований.

Концептуальная модель поведения представляет собой не полностью формальную явную спецификацию поведения системы. С одной стороны, такое представление требований требует от экспертов значительно меньше усилий, чем построение полностью формальной спецификации, с

другой — позволяет провести достаточно тщательный анализ полноты, минимальности, адекватности и непротиворечивости требований.

Декомпозиция поведения системы

Концептуальная модель поведения разрабатывается для отдельных элементов поведения системы, таких как алгоритм ее поведения в некоторой ситуации или используемый протокол обмена данными с другими системами. Декомпозицию системы проводят эксперты в предметной области. Они выделяют компоненты или группы связанных функций системы, которые оперируют общим набором данных. В случае простых систем, таких как реализация отдельного протокола или специализированного программного интерфейса, такая группа может быть одна. Для более сложных систем групп функций может быть много.

Для представления декомпозиции используются диаграммы UML — диаграммы классов, диаграммы объектов или диаграммы развертывания (deployment).

Разработка концептуальной модели и установление прослеживаемости требований

После того как проведена декомпозиция поведения системы, эксперты разрабатывают модели поведения для выделенных групп функций.

Представление поведения в модели зависит от характера поведения: для алгоритмов обработки данных или программного интерфейса это *псевдокод*, для протокола обмена сообщениями — *расширенный автомат*, заданный таблицей переходов.

В псевдокоде используются элементы какого-либо языка программирования (Си, Java и т.п.), но отдельные операторы и выражения могут заменяться на фразы на естественном языке. Такая полуформальная модель поведения может рассматриваться как не полностью формальная явная спецификация, или как прототип или схема реализации. С каждым ветвлением и с каждым действием в модели должен быть связан набор регламентирующих требований из каталога.

В таблице, описывающей автомат, задаются переходы между его состояниями. Для каждого перехода определяются его начальное состояние, входной символ, производимое действие, выходной символ и конечное состояние. В описаниях действия и выходного символа допускается свободный неформальный текст о том, что происходит при выполнении действия и какими свойствами обладает результат. С каждым переходом сопоставляется набор требований, регламентирующих его действие, изменение состояния автомата и выходные сообщения. Табличное задание автомата может быть не столь наглядно, как его графическое описание или UML Statechart, но позволяет проводить более аккуратный анализ полноты и адекватности требований к автомату.

Концептуальная модель разрабатывается на основе спецификации требований и общих знаний о том, как должна работать система. Главное ограничение, которому должна удовлетворять концептуальная модель — это ее *функциональная полнота*. То есть, в ней должны быть описаны все варианты поведения, допустимые в рамках ограничений на входные данные.

Установление прослеживаемости требований заключается в том, что каждому действию в модели сопоставляется набор требований, регламентирующих его результаты.

Анализ полноты и минимальности набора требований

Функциональные требования могут покрывать только часть функциональности системы. Набор требований будем считать полным, если для тех вариантов поведения системы, которые не покрыты функциональным требованиями, в спецификации явным образом сказано, что их реализация оставлена на усмотрение разработчиков (implementation dependent).

Набор требований будем считать минимальным, если к каждому действию предъявляется не более одного требования.

Если при разработке концептуальной модели или установлении прослеживаемости требований выяснится, что при некоторых условиях на состояние и входные данные модели на поведение системы нет требований, это может означать неполноту выделенного набора требований. Можно указать несколько возможных причин такой неполноты.

- При разметке и идентификации требований некоторые из них могли быть не выявлены. Использование разметки документов способствует выделению всех возможных функциональных требований, но не может служить абсолютной гарантией. Поэтому в случае обнаружения неполноты набора требований необходимо прежде всего проверить, есть ли в неразмеченной части спецификации соответствующие требования. Если они будут обнаружены, то необходимо вернуться на первый этап процесса и внести требования в каталог, а затем добавить их к модели.
- Встречаются ситуации, когда требования оказываются «размазаны» по нескольким документам. Так, например, основополагающие требования к реализации IPMP-2 сформулированы в стандарте ISO/IEC 13818-11 [50], но часть ограничений накладывается общими требованиями к MPEG-2 (ISO/IEC 13818-1,2,3 [53-55]). По этой причине если в исходной спецификации недостающие требования не найдены, то надо изучать документы, описывающие объемлющую систему или смежные системы. Если в ходе анализа неполноты будут обнаружены не рассмотренные ранее документы с требованиями, то необходимо ввести эти документы в состав спецификации и вернуться на первый этап.
- Еще одна причина возможной неполноты каталога требований заключается в том, что некоторые требования кажутся настолько очевидными разработчикам спецификации, что не вносятся в ее текст. Например, для разработчиков исходной спецификации IPMP-2 было *очевидно*, что реализация должна игнорировать неверно сформированные входные данные, но в тексте спецификации об этом не было ни слова. Если обнаружен такого рода «пробел», то для достижения полноты требований такие «очевидные» ограничения фиксируются и вносятся в каталог как извлеченные из предметной области, не имеющие представления в виде фрагментов текста спецификации. При этом очень полезно иметь контакт с разработчиками спецификации, чтобы узнать, как они трактуют выявленное «очевидное» поведение.
  Если удалось выявить такого рода подразумеваемые, но неоговоренные требования и ограничения, стоит включить их в отчет о дефектах спецификации и представить рекомендацию внести требование в спецификацию, так как то, что ясно одним, может оказаться совсем не очевидным для других.

Анализ адекватности концептуальных моделей

Концептуальная модель поведения представляет собой сжатое и четкое описание желаемого поведения системы, снабженное ссылками на регламентирующие требования. Это дает возможность экспертам и аналитикам проводить анализ адекватности требований. В рамках данного метода для валидации предлагается использовать возможные сценарии работы системы: эксперт предметной области может мысленно или на бумаге выполнить некоторый сценарий и проверить, что реализация сценария в рамках построенной модели соответствует ожиданиям.

Например, для телекоммуникационных протоколов очень важно, чтобы корректные реализации могли успешно взаимодействовать (interoperate). Для проверки этого аспекта адекватности эксперт в мысленном эксперименте строит различные корректные сценарии обмена сообщениями и проверяет, что реализации сохранят способность взаимодействовать в каждом из них. При проведении анализа особое внимание следует обращать на «исключительные» ситуации — ошибки передачи, нарушение порядка сообщений и т.п.

В ряде случаев второй этап процесса FOREST кажется очень простым и есть соблазн сразу перейти от каталогизации к формализации требований. В частности, такая ситуация может возникнуть, когда опытные разработчики работают над простыми программными интерфейсами

со строгим разбиением на подсистемы и четко прописанными требованиями (например, хорошо документированы некоторые подсистемы POSIX API). Тем не менее, мы рекомендуем всегда выполнять представленные выше действия. Как показывает практика, разработка концептуальной модели значительно облегчает анализ требований и последующую формализацию.

Разрешение проблем

В процессе FOREST для разрешения неполноты, противоречивости или неадекватности требований используется техника построения сценариев. Для проблемных ситуаций строятся сценарии на естественном языке или с использованием графических нотаций, которые описывают эволюцию системы, приводящую в такую ситуацию. Эти сценарии сопровождаются описанием выявленных проблем и передаются экспертам в предметной области или авторам спецификаций. Результатом обсуждения может быть исправление требований или добавление новых требований. В обоих случаях необходимо внести изменения в каталог требований и концептуальные модели.

Связь с Ф-процессом

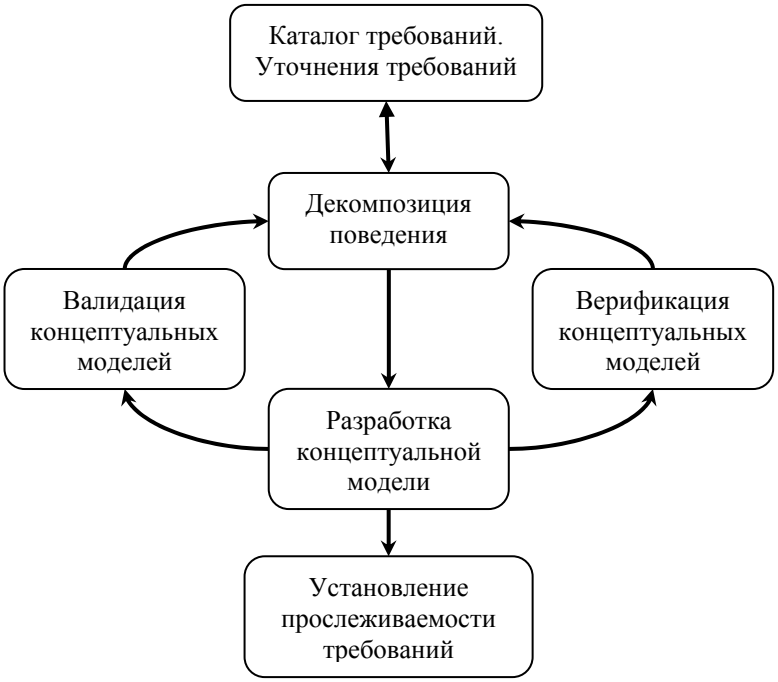


Рисунок 10. Отображение процесса второго уровня на Ф-процесс

В отображении работ второго этапа на Ф-процесс, представленном на Рис. 10, выделение требований опирается на каталог требований, полученный в результате первого этапа. Каталог требований может изменяться в процессе работы с концептуальными моделями — могут добавляться новые требования и уточняться существующие.

Валидация и верификация требований Ф-процесса соответствуют валидации и верификации концептуальных моделей. Внешние свойства требований обеспечиваются благодаря тому, что в концептуальных моделях обязательно сохраняются ссылки на исходные требования из каталога.

Построение формальной модели требований

Задачи этапа	• Выделение требований
	  o Согласование требований
	• Систематизация требований
	• Описание требований
	• Валидация требований
	• Верификация требований
	  o Проверка однозначности

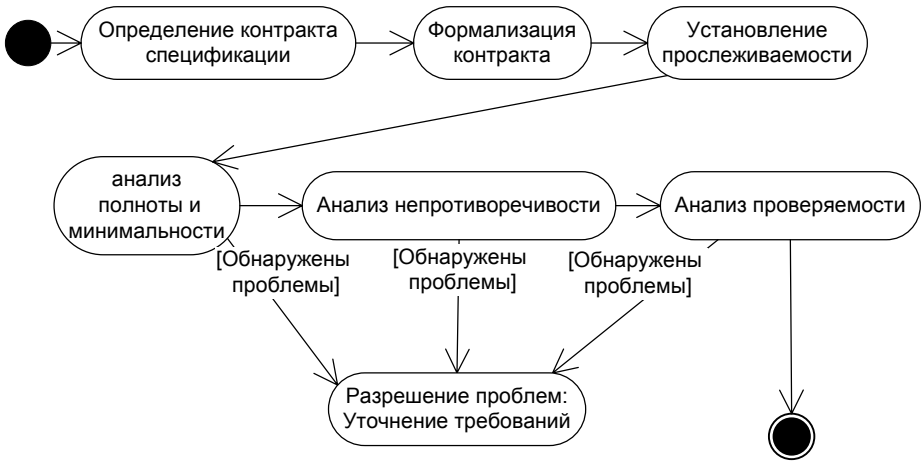
	○ Проверка полноты ○ Проверка непротиворечивости ○ Проверка минимальности ○ Проверка систематичности • Обеспечение внешних свойств требований ○ Обеспечение прослеживаемости ○ Обеспечение проверяемости ○ Обеспечение модифицируемости
Входные данные этапа	Каталог требований, текстовая спецификация, концептуальная модель поведения
Схема выполнения этапа	
Результаты этапа	Формальная спецификация

Таблица 5. Краткое описание этапа «Построение формальной модели требований».

В процессе FOREST требования формализуются в виде *контрактных спецификаций*. Рассмотрим достоинства таких спецификаций в контексте анализа требований.

- Контрактные спецификации позволяют сравнительно легко описывать различные виды недетерминированного поведения, в том числе недетерминизм, вызванный неполнотой исходной спецификации или особенностями реализации.
- Существуют эффективные способы автоматического построения тестовых оракулов из контрактных спецификаций [56,57]. Это позволяет автоматизировать процесс проверки соответствия реализации требованиям.

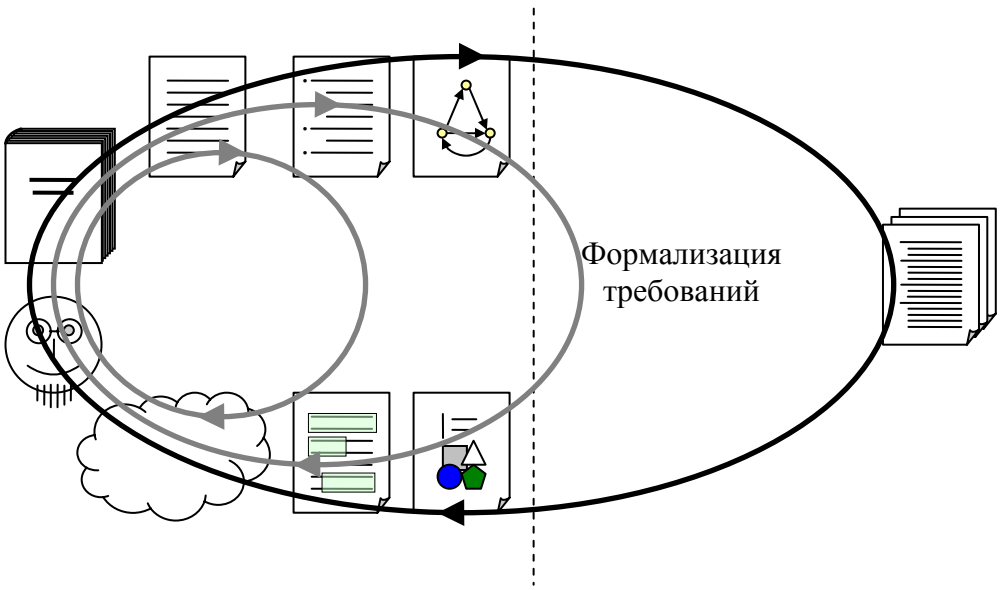


Рисунок 11. Разработка формальных спецификаций опирается на все результаты, полученные ранее.

- Разработка контрактных спецификаций требует мышления в стиле требований к результату, который отличается от характерного стиля мышления разработчиков программных систем. Это позволяет выявить особенности требований, которые остались бы незамеченными при анализе требований при помощи прототипирования или разработки экспериментальных реализаций. В частности, мы считаем, что контрактная спецификация позволяет лучше анализировать требования к недетерминированному поведению, чем явные исполнимые спецификации или прототипы.

Третий этап состоит из нескольких шагов.

1. **Определение состава формального интерфейса**. В результате этого шага определяется множество операций и событий, посредством которых программная система взаимодействует с внешним миром.
2. **Формализация контракта**. На этом шаге функциональные требования, извлеченные на первых двух этапах, записываются в виде формальных спецификации контракта.
3. **Валидация и верификация требований**.

В описываемом подходе используются так называемые событийные контрактные спецификации [58,59], в которых интерфейс системы состоит из *стимулов* и *реакций*. Стимулы соответствуют воздействиям, которые оказывает окружение на систему, реакции — воздействиям системы на ее окружение.

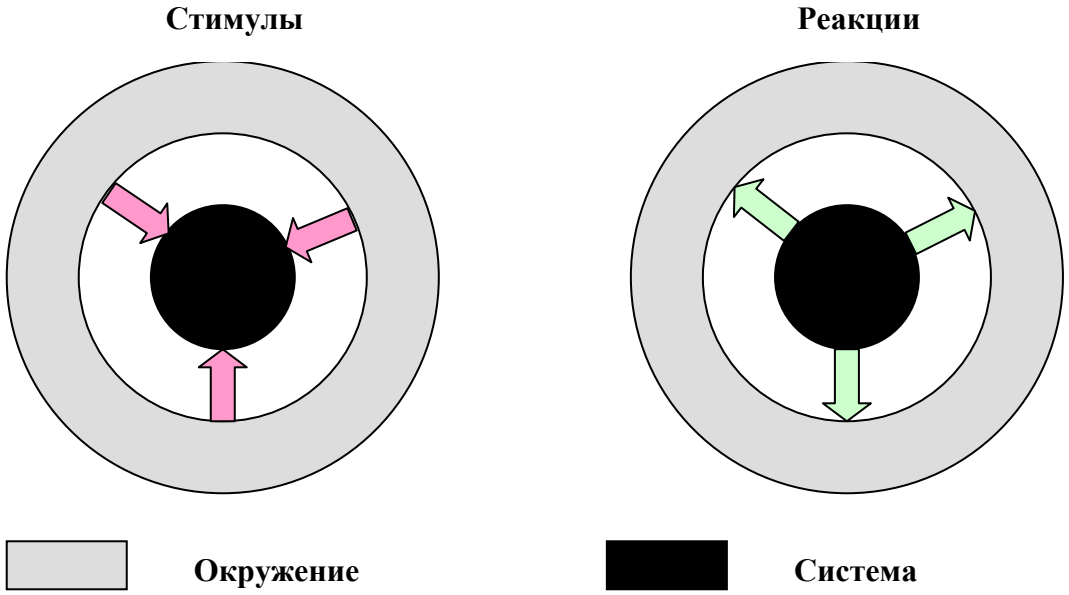


Рисунок 12. Стимулы и реакции.

Определение состава формального интерфейса

Задача первого шага формализации — представить возможные взаимодействия системы с окружением в виде набора стимулов и реакций.

Наиболее просто эта задача решается для систем, имеющих синхронный программный интерфейс, не предусматривающий какого-либо асинхронного взаимодействия с окружением или возможности одновременного поступления нескольких воздействий на систему. Формальный интерфейс таких систем состоит просто из их интерфейсных операций.

Более сложно устроен интерфейс для асинхронных систем, предусматривающих возможность параллельного вызова нескольких операций или передачи вовне данных без непосредственно предшествующего запроса на них, систем с обратными процедурными вызовами (callbacks), событиями (events) в интерфейсе. В этих случаях каждая интерфейсная операция разделяется на

два элемента формального интерфейса: стимул-вызов функции и реакцию-возврат управления. Различные виды асинхронные события, генерируемых системой, становятся реакциями.

Наименее очевидным образом выделяются элементы формального интерфейса для протоколов и систем, которые взаимодействуют черед сложно устроенные структуры данных, например, при помощи передачи сообщений в формате XML.

В этих случаях необходимо различать определение типов данных и структуры сообщений в спецификации системы (физические данные) и абстрактный стимул или реакцию. На практике встречаются самые разные соотношения между физическими данными и элементами формального интерфейса. Рассмотрим некоторые примеры.

- В многоуровневых стеках протоколов один пакет переносит данные нескольких протоколов, но элементами формального интерфейса удобно считать не пакеты, а заголовки отдельных протоколов. Тем самым один пакет может соответствовать нескольким стимулам или реакциям.
- В потоковом видео данные разделяются на небольшие блоки, причем текст стандарта специфицирует представление данных (таблиц) именно в таких небольших блоках. Несмотря на такую спецификацию, стимулом лучше считать получения всей таблицы данных, абстрагируясь от ее представления в потоке. В этом случае несколько физических разделенных блоков данных соответствуют одному элементу формального интерфейса.

Рассмотрим состав формального интерфейса для различных видов программных систем.

- **Системы с синхронным программным интерфейсом.**
  В этом случае каждой интерфейсной операции соответствует один элемент формального интерфейса.
- **Системы с асинхронным программным интерфейсом.**
  - *Синхронные функции*, т.е. функции, которые для внешнего наблюдателя исполняются атомарно в любом состоянии и в любой момент времени.
    Каждой синхронной функции соответствует один элемент формального интерфейса.
  - *Асинхронные функции* — интерфейсные функции, которые не являются синхронными.
    Если формализм поддерживает только стимулы и реакции, то вызов асинхронной интерфейсной функции входит в формальный интерфейс как стимул, а возврат ею управления — как реакция.
    Возможно, в используемом формализме есть поддержка *расщепленных операций*, то есть можно разделить операцию на несколько атомарных действий и задать постусловие для каждого из них. Постусловия определяют изменения состояния непосредственно после действия, состояние после возврата управления из функции определяется постусловием последнего действия. В таком случае функция входит как один элемент формального интерфейса с как минимум двумя атомарными действиями (одно — вызов функции, другое — возврат управления).
  - *Обратные функции (callbacks)*.
    Обратные функции, как правило, используются в контексте многопотоковых приложений, поэтому мы рекомендуем всегда рассматривать их как асинхронные операции.
    Если формализм поддерживает только стимулы и реакции, то вызов обратной функции (система передает управление окружению) входит в формальный интерфейс как реакция, передача управления обратно системе входит как стимул. Формализм может поддерживать спецификацию обратных вызовов как функций специального вида, но и в этом случае для асинхронных интерфейсов необходимо разделять действия по вызову обратной функции и возврату ею управления.

Возможно, формализм поддерживает описание расщепленных операций. Тогда обратная функция соответствует одному элементу формального интерфейса с несколькими атомарными действиями.

- *События (events)*.
  Рассматриваются как частный случай обратных функций.

- **Системы с обменом сообщениями**, с одним уровнем транспорта сообщений и простыми сообщениями.
  Если система предоставляет программный интерфейс, то стоит следовать рекомендациям для асинхронных программных интерфейсов.
  Для каждого сообщения из алфавита сообщений в формальный интерфейс вводятся два элемента.
  - *Получение сообщения*. В этом элементе формального интерфейса специфицируется обработка входящего сообщения.
  - *Отправка сообщения*. Здесь специфицируется проверка корректности исходящего сообщения. Отправку сообщения можно рассматривать как частный случай обратного вызова.

- **Реализации протоколов верхнего уровня**.
  Реализации протокола обмениваются сообщениями через транспортный уровень и предоставляют программный интерфейс приложениям. Их формальный интерфейс устроен так же, как в предыдущем случае.

- **Системы, взаимодействующие с аппаратурой** (драйверы, HAL).
  Контракт устроен так же, как для систем с обменом сообщениями. Обращения к аппаратуре рассматриваются как отправка и получение сообщений. Прерывания рассматриваются как получение сообщений.

- **Стеки протоколов**.
  Если стек предоставляет программный интерфейс, то стоит следовать рекомендациям для асинхронных программных интерфейсов.
  Если стек взаимодействует с протоколами верхнего уровня, то это рассматривается как асинхронный программный интерфейс с обратными вызовами: отправка пакета верхнего уровня соответствует вызову интерфейсной функции, доставка пакета, полученного из сети — вызову обратной функции.
  Каждый уровень стека представляется в контракте следующими элементами.
  - Обработка заголовка заданного протокола во входящем сообщении,
  - Обработка окончания (trailer), если оно есть, заданного протокола во входящем сообщении.
  - Отправка сообщения заданного протокола.

  Если протоколы, составляющие стек, поддерживают самовложенность, то есть в качестве вложенного сообщения может выступать сообщение того же или более низкого уровня, то можно выделить получение/отправку такого вложенного сообщения в отдельные элементы формального интерфейса.
  Если в некотором уровне стека можно выделить подуровни (например, вспомогательные заголовки в IPv6), то
  - для каждого подуровня в формальный интерфейс вводится элемент, соответствующий обработке заголовка данного подуровня;
  - для исходящих сообщений в контракт вводится только один элемент формального интерфейса, соответствующий отправке сообщения данного уровня.

- **Состав контракта для систем со сложными сообщениями**.
  Под сложными сообщениями мы понимаем сообщения, которые переносят информацию одного протокола, но при этом в одном физическом пакете могут содержать данные нескольких логических сообщений. Примером такой системы может служить анализатор трассы UniTESK [57]: одна трасса состоит из множества сообщений о различных событиях, происходивших в ходе тестирования. Другой пример — IPMP-2: поток MPEG-2 содержит последовательность структур данных, относящихся к одному логическому сообщению.
- Как правило, сложные сообщения являются вложенными, то есть сообщения одного вида содержат одно или несколько сообщений другого вида. Поэтому их можно рассматривать как многоуровневый протокол, в котором каждому виду сообщений соответствует отдельный уровень.

В контрактных спецификациях стимулы и реакции представляются как функции с пред- и постусловиями. Такие функции мы будем называть *спецификационными*. В результате выполнения первого шага должны быть определены сигнатуры спецификационных функций.

Формализация контракта

На втором шаге этапа формализации требования записываются как логические условия, входящие в состав пред- и постусловий спецификационных функций.

Построение формальной спецификации требований сродни программированию. В обоих случаях неформальное понимание того, что нужно сделать, или описывающий его текст на естественном языке преобразуется в текст на строго определенном формальном языке.

В последнее время методология программирования сделала значительный шаг вперед благодаря выделению образцов, или паттернов, проектирования программ. Методология формализации пока разработана не столь глубоко, поэтому далее в тексте мы представим только примеры отдельных приемов формализации требований без описания деталей.

- **Формализация требований к целостности структур данных**. В контрактных спецификациях ограничения целостности данных формализуются, как правило, в инвариантах типов данных, моделирующих соответствующие структуры.
- **Формализация негативных требований**. Под негативными требованиями понимаются требования, которые запрещают системе вести себя определенным образом. Негативные требования для возвращаемого значения функции специфицируются в постусловии соответствующей спецификационной функции, запрет на демонстрацию реакции задается в постусловии реакции. Такие требования специфицируются в постусловии: если система возвращает запрещенное значение или демонстрирует запрещенное поведение, то постусловие выносит отрицательный вердикт о корректности ее поведения.
- **Формализация требований к ожидаемым реакциям**.
  - В большинстве формальных языков, поддерживающих контрактные спецификации, нет встроенной поддержки темпоральных спецификаций, то есть спецификаций свойств цепочки событий, упорядоченных во времени. Поэтому в этих формализмах требования к ожидаемой реакции формулируются в постусловии спецификационной функции, соответствующей реакции. Постусловие выносит вердикт на основании истории взаимодействий, которая хранится в состоянии модели.
  - В некоторых формализмах, разработанных для технологии UniTESK, есть частичная поддержка темпоральных спецификаций: разработчик спецификации может указать в описании стимула, что в будущем ожидается реакция и задать требования к реакции в виде логического выражения.
- **Формализация требований к окружению**. Ограничения, накладываемые на окружение, представляются в предусловиях и ограничениях целостности для входных параметров.

- **Требования к необязательным (optional) функциям** или необязательному поведению. Для необязательных функций в формальную спецификацию включаются параметры, значение которых определяет включение соответствующих требований в состав спецификации системы.
  Набор фактических значений параметров спецификации служит формальной записью реализационно-зависимой части требований. Значения параметрам присваиваются экспертами на основе текстового описания соответствия реализации (Implementation Conformance Statement), исследования исходных текстов реализации или изучении ее поведения.

Обеспечение прослеживаемости требований в формальной спецификации

В предлагаемом подходе требования представляются в виде логических выражений, входящих в состав пред- и постусловий контрактных спецификаций и инвариантов данных.
Прослеживаемость требований обеспечивается установлением явных связей между формальным представлением требования и его описанием в тексте исходных документов.

В предлагаемом процессе можно использовать два подхода к обеспечению прослеживаемости.

- Можно поместить в спецификацию специальные конструкции, указывающие на идентификаторы требований, соответствующих тем или иным ее элементам.
- Можно установить связь между требованиями и элементами контрактных спецификаций средствами систем управления требованиями. Например, некоторые инструменты управления требованиями позволяют строить отображения из требований в произвольные объекты. В таких инструментах можно представить элементы спецификаций как внешние объекты и задать отображения требований в них.

Связь с Ф-процессом

Рисунок 13. Отображение процесса построения формальной модели на Ф-процесс

В отображении процесса формализации требований на Ф-процесс, представленном на Рис. 13, выделение требований опирается на каталог требований и концептуальные модели, полученные в на предшествующих этапах. Каталог требований и концептуальные модели могут изменяться в

процессе формализации требований, в них могут добавляться новые требования и уточняться существующие.

Валидации и верификации требований в Ф-процессе соответствуют валидации и верификации формальных моделей. Внешние свойства требований обеспечиваются благодаря тому, что в концептуальных моделях обязательно сохраняются ссылки на исходные требования из каталога. Тестируемость требований может быть обеспечена инструментальной поддержкой технологии тестирования UniTESK [56,57].

В следующем разделе кратко описан процесс тестирования с использованием контрактных спецификаций по технологии UniTESK.

Тестирование соответствия требованиям

Процесс формализации требований FOREST использовался в проектах по тестированию различных программных системы. По этой причине в качестве примера применения формальных спецификаций мы кратко представим, как формальные спецификации использовались для тестирования соответствия.

Задачи этапа	• Разработать тестовый набор для проверки соответствия системы исходным требованиям • Проверить, насколько данная система удовлетворяет функциональным требованиям к ней
Входные данные	Каталог требований, текстовая спецификация, концептуальная модель, формальная спецификация
Результаты этапа	Тестовый набор для проверки соответствия требованиям, отчеты о проведенном тестировании.

Таблица 6. Краткое описание этапа «Тестирование соответствия требованиям».

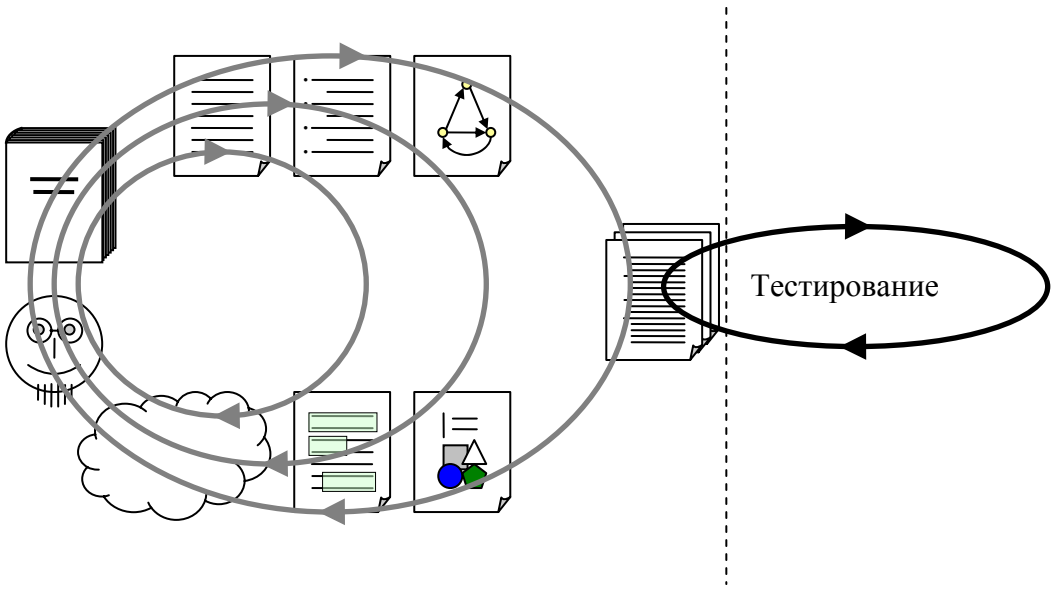


Рисунок 14. Тестирование может опираться на формальные спецификации требований.

Проекты тестирования соответствия, основанные на процессе FOREST и технологии UniTESK, включают следующие группы работ.

1. **Определение критериев полноты тестирования**. Критерии полноты тестирования в UniTESK формулируются в виде метрик покрытия различных элементов формальной модели. При этом задается разбиение пространстве входных и выходных параметров спецификационных функций на классы эквивалентности, а в контрактные спецификации

вводятся специальные инструкции, которые формализуют определения классов в рамках этого разбиения.

2. **Выбор целей тестирования**. Цели тестирования в предлагаемом процессе формулируются в терминах метрик покрытия, определенных на предыдущем шаге. Например, цель тестирования может состоять в покрытии 80% выделенных классов эквивалентности ситуаций, или в обязательном покрытии некоторого подмножества этих классов. Оценка покрытия требований производится после прогона тестов.
3. **Разработка тестов для достижения выбранных целей тестирования**. На данном шаге разрабатываются генераторы тестовых данных и тестовые адаптеры. Оракулы для вынесения вердикта о соответствии поведения реализации требованиям автоматически строятся из контрактных спецификаций.
4. **Отладка тестов**. Для отладки тестов используется прогон тестового набора на эталонной (reference) реализации. Эталонная реализация признана экспертами в предметной области как полностью соответствующей спецификации, поэтому сообщения об ошибках, выявленных при прогоне, свидетельствуют о наличии ошибки в тестах. Случаи, когда тесты выявляют ошибку в эталонной реализации, мы здесь не рассматриваем, это отдельный деликатный вопрос.
   В тех случаях, когда эталонная реализация недоступна, мы рекомендуем разработать функциональный прототип системы и провести испытания тестов на нем.
   Помимо проверки того, что тесты работают правильно на корректных реализациях, необходимо убедиться, что они действительно способны выявлять ошибки. Для этого используется прогон тестового набора на *мутанте* эталонной реализации (так называется это программа, полученная из исходной программы внесением ровно одного дефекта, способного привести к наблюдаемому некорректному поведению.) При прогонах тестов необходимо убедиться, что хотя бы один тест из набора выявляет привнесенный дефект. Задача построения мутантов выходит за рамки данной статьи.
5. **Адаптация тестового набора к реализации**. К этой группе относятся работы по заданию параметров спецификации, настройке связи с тестовыми агентами, и т.п.
6. **Прогон тестов на целевой реализации и анализ результатов**.

Подробное описание разработки тестов с использованием контрактных спецификаций выходит за рамки данной статьи, см. [56-59].

Результаты отдельных этапов и их связи с задачами процесса в целом

- **Каталог требований.**
  Каталог состоит из списка требований, налагаемых текстовой спецификацией и общим контекстом предметной области. Каждое требование, извлеченное из спецификации, каталог привязывает к соответствующему месту в ее тексте. Одно такое требование обычно соответствует логически замкнутому фрагменту текста, выражающему одно ограничение на функциональность или структуру интерфейсных данных системы. Каждое требование имеет уникальный идентификатор. Каталог требований позволяет в дальнейшем оценивать адекватность тестирования или полноту реализации в терминах исходного текста стандарта.
- **Размеченный текст спецификации**.
  Это исходный текст спецификации, в котором наглядным образом, например цветом, выделены места, соответствующие требованиям, внесенным в каталог. Размеченный текст позволяет проверить полноту выполненного анализа текста стандарта и убедиться в том, что ни одно требование не пропущено.
- **Концептуальная модель поведения**.
  Это документ или набор документов, в которых поведение целевой системы — объекта описания текстовой спецификации — задано при помощи исполнимой модели. В ней

программные инструкции могут быть заменены текстом на естественном языке (псевдокод). С каждым условным оператором и действием в псевдокоде связан набор регламентирующих требований из каталога (аннотирование). Концептуальная функциональная модель позволяет проверить полноту выделенного набора требований, адекватность требований, их минимальность и непротиворечивость.

- **Дефекты спецификации требований и замечания к ее тексту**.
  Это список обнаруженных двусмысленностей, несоответствий между утверждениями в различных частях спецификации, ненамеренно неясных и неточных ограничений, неполных описаний функциональности и т.д. Он передается группе разработчиков спецификации и позволяет сделать следующие версии спецификации более точными и непротиворечивыми.
- **Формальная спецификация требований**.
- **Тестовый набор для проверки соответствия требованиям**.

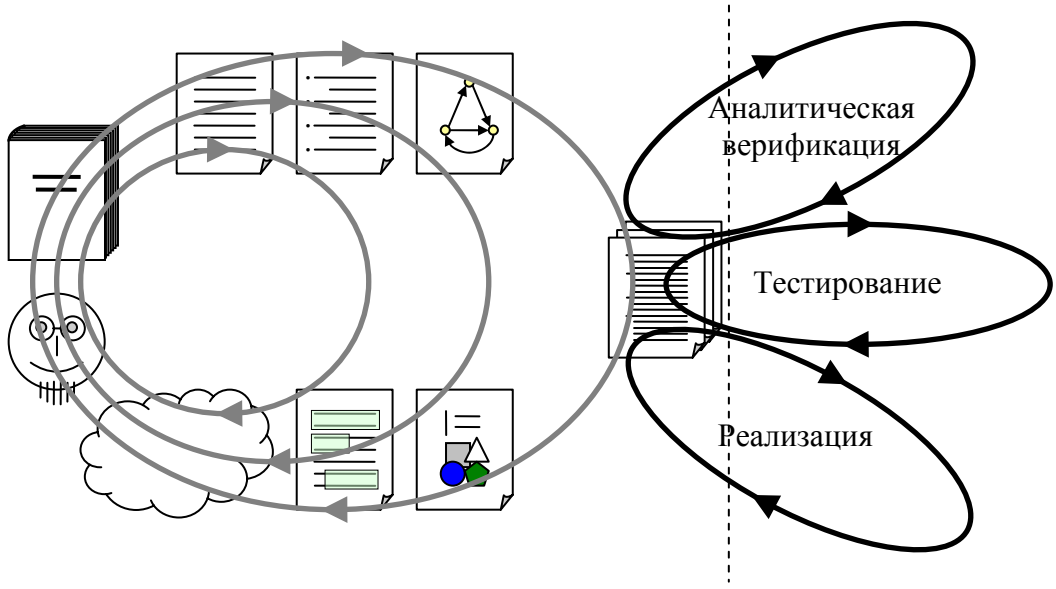


Рисунок 15. Процесс FOREST может использоваться в широком спектре задач разработки ПО.

Практические применения предложенного процесса

Данный раздел описывает три примера использования представленного выше процесса формализации требований на практике. Во всех рассматриваемых проектах формализация требований проводилась с целью дальнейшего построения тестовых наборов на соответствие требованиям, хотя во втором примере до этого дело не дошло — проект, несмотря на полученные полезные результаты, был свернут из-за отсутствия адекватного финансирования.

Разработка тестового набора для стека протоколов IPv6

Интернет-протокол нового поколения IPv6 [49] — это совокупность протоколов, которые относятся к сетевому уровню в эталонной модели OSI [60]. Протоколы IPv6 предоставляют сервисы протоколам транспортного уровня, прежде всего TCP и UDP.

Этот Интернет-протокол предусматривает значительно большее адресное пространство, нежели используемый ныне Интернет-протокол IPv4. Большое адресное пространство позволяет организовать полноценную двустороннюю связь между произвольными узлами в глобальной сети. Помимо расширенного адресного пространства IPv6 содержит улучшенную архитектуру маршрутизации, интегрированный набор протоколов для автоматической конфигурации и автоматического определения сетевого окружения.

Протоколы IPv6 стандартизованы Комитетом по стандартизации Интернета (Internet Engineering Task Force, IETF) в наборе IETF RFC (Request for Comments) [61-71].

Работы в проектах по тестированию реализаций IPv6 проводились в соответствии с процессом FOREST.

Составление каталога требований

Составление перечня регламентирующих документов. Насчитывается несколько десятков спецификаций протоколов и технологий, относящихся к IPv6, поэтому перед началом работ были проведены исследования и очерчена предметная область, для которой будут разрабатываться формальные спецификации и тесты. Кратко перечислим функции протокола IPv6, выбранные для тестирования.

- Передача датаграмм транспортного уровня в сеть и обработка входящих пакетов IPv6.
- Группа протоколов автоматического определения окружения в локальной сети Neighbor Discovery на оконечных узлах (host). В состав Neighbor Discovery входят протоколы для обнаружения маршрутизаторов и оконечных узлов, присоединенных к сегменту локальной сети, определение достижимости узлов в сегменте локальной сети, определения параметров сегмента локальной сети.
- Автоматическая конфигурация адресов IPv6 на сетевом интерфейсе. Эта функция позволяет устройствам самостоятельно сформировать сетевые адреса и уменьшить нагрузку на администраторов по конфигурированию локальной сети.
- Протокол Multicast Listener Discovery (MLD) для конечного узла. Протокол MLD позволяет маршрутизатору собрать информацию о том, какие узлы в сегменте локальной сети присоединились к определенному групповому (multicast) адресу.
- UDP в сети IPv6.
- Мобильность узлов в сети IPv6.

Требования к указанным функциям извлекались преимущественно из IETF RFC 2292, 2460, 2461, 2462, 2463, 2553, 2675, 2710, 3513 [61,63-66,68-71], проекта стандарта протокола Mobile IPv6 [72,73]. Для моделирования состояния протокола изучались описания Management Information Base (MIB), относящихся к IPv6 [74-82]. Суммарный объем проанализированных документов составил 500 страниц.

Идентификация требований. Для идентификации требований использовалась разметка спецификаций и выделение требований цветом. Многие требования в RFC помечаются ключевыми словами MUST, SHOULD, MAY и т.п., поэтому в первую очередь размечаются такие помеченные требования. Требования, помеченные ключевыми словами, составляют около 70% от всего набора выделенных требований.

Еще одна особенность представления требований в RFC заключается в том, что встречаются отдельные участки текста, которые определяют сразу два требования — одно к обработке входящих сообщений и одно — к формированию исходящего сообщения. Мы разделяли такие требования, так как они формализуются по-разному.

Идентификаторы для требований выбирались на основе их смысла. Мы отказались от нумерации требований в пользу большей понятности спецификации — ссылка на именованное требование несет больше информации для читателя спецификации, чем ссылка по номеру.

Внесение требований в каталог. Идентифицированные требования вносились в каталог. Помимо них в каталог вносились требования, извлеченные из контекста предметной области. Доля таких требований в каталоге составляет около 20%. Всего в каталог внесено порядка 3000 требований.

Структурирование требований. Для требований к реализациям протоколов IPv6 была разработана специализированная классификация. В частности, общие категории, представленные в разделе «Структурирование требований» данной работы, были уточнены: были выделены типы объектов, к которым могут предъявляться требования, и введены четыре класса требований по степени их обязательности.

Построение концептуальной модели требований

Разработка концептуальной модели и установление прослеживаемости требований. В спецификациях протоколов IPv6 операции протоколов описываются, как правило, текстом на английском языке без использования каких-либо специальных нотаций. Это относится как к описаниям алгоритмов обработки данных, так и описаниям последовательностей сообщений. В некоторых спецификациях приводятся автоматные модели протоколов, но анализ показал, что эти модели существенно не полны и плохо подходят для анализа требований.

Для анализа описаний алгоритмов были разработаны модели алгоритмов на псевдокоде.

По описаниям сценариев обмена сообщениями были разработаны таблицы переходов автоматов протоколов. Разработка автоматных моделей осложнялась тем, что в стеке IPv6 одновременно работают несколько служебных протоколов, которые влияют на работу всех остальных протоколов. По этой причине разработанные модели охватывают сразу несколько протоколов.

Модели алгоритмов и автоматные модели основывались на концептуальных структурах данных, представляющих внутреннее состояние протоколов IPv6. Для ряда протоколов такие структуры данных приведены в спецификации протокола (RFC), для других они были разработаны на основании анализа операций протокола и информационной модели протокола, изложенной в Management Information Base [74-82].

Анализ полноты и минимальности набора требований. В ходе анализа RFC был выявлен ряд несоответствий и пропусков в спецификациях IPv6. Например, спецификация IPv6 RFC 2460 [63] перечисляет несколько ситуаций, которые должны рассматриваться как ошибки, и несколько ситуаций, которые ошибками не являются. К сожалению, этот перечень не содержит ряд важных случаев, в особенности проблемных ситуаций, которые могут возникнуть при сборке пакета из нескольких фрагментов.

Тем не менее, несмотря на обнаруженные недочеты, мы считаем, что в целом спецификации протоколов IPv6 хорошо определены. Они содержат достаточно деталей, чтобы обеспечить совместимость между реализациями, и одновременно не накладывают слишком жестких ограничений на реализацию.

Анализ адекватности концептуальных моделей. Адекватность построенных концептуальных моделей исходным требованиям проверялась при помощи инспекции моделей.

Протоколы IPv6 представляют собой сложные протоколы, которые длительное время разрабатывались большим сообществом специалистов, поэтому в ходе данной работы детальная проверка адекватности собственно протоколов не производилась.

Построение формальной модели требований

Определение состава формального интерфейса. Реализации IPv6 предоставляют три вида интерфейсов: процедурный (программный интерфейс, API), двоичный (Application Binary Interface, ABI) и интерфейс сообщений.

Процедурный интерфейс включает общий интерфейс сокетов, а также специализированные расширения для IPv6. Двоичный интерфейс состоит из нестандартизованных, реализационно-зависимых средств доступа к реализации протокола, обычно расположенной в ядре ОС. Примерами такого интерфейса могут служить коды команд ioctl в операционных системах семейства Unix или управляющие коды функции DeviceIoControl в Windows вкупе с правилами

представления в памяти структур данных для запросов и возвращаемых значений. Интерфейс сообщений служит абстракцией для операций получения пакетов протокола IPv6 с канального уровня и их отправки на этот уровень.

Двоичный интерфейс и часть программного интерфейса протокола IPv6 не стандартизованы и зависят от реализации. По этой причине в они не вошли в его формальную спецификацию.

Помимо внешне наблюдаемых событий, входящих в интерфейс, в состав формального интерфейса были введены внутренние события, соответствующие обработке заголовков расширения протокола IPv6 и протоколов верхнего уровня. Это сделано для упрощения достижения заданного покрытия требований во время тестирования и улучшения модифицируемости формальной спецификации протокола за счет ее модульности.

Формализация контракта. Модельное состояние спецификации представляет собой модель данных сетевого узла IPv6. Она основывается на концептуальных моделях протоколов, входящих в состав стека IPv6, и информационных моделей, представленных в MIB [74-82].

Для каждого элемента формального интерфейса были разработаны спецификационные функции. Предусловия использовались для ограничения формальной спецификации только хорошо определенными требованиями — они отсекали такие сообщения, для которых стандарты протоколов IPv6 не содержат требований по обработке.

Инварианты структур данных для сообщений формализуют общие требования к корректным сообщениям, как входящим, так и исходящим.

Постусловия спецификационных функций формализуют требования к соответствующему объекту или требования к изменению состояния соответствующего протокола.

Обеспечение прослеживаемости требований в формальной спецификации. Для указания требований, на основании которых разрабатываются спецификации, в постусловия и инварианты вводятся специальные конструкции. С каждой логической формулой, представляющей некоторое требование, связывается идентификатор этого требования. Например, если требование с идентификатором ID записывается логическим выражением F(arg), то в формальной спецификации вместо F(arg) записывается выражение REQ(F(arg), “ID”). Такая форма обеспечения прослеживаемости требований дает два следствия. Во-первых, при валидации постусловия легко проверить адекватность выражения F(arg), так как указано, какое именно требование оно представляет. Во-вторых, при проверке постусловия в ходе тестирования в трассу теста будет помещено сообщение о том, что требование ID проверено, что позволит автоматически определить покрытие требований тестами.

Тестирование соответствия требованиям

На основе формальных спецификаций были разработаны тестовые наборы для тестирования соответствия реализаций IPv6 соответствующим стандартам. Тестовые наборы использовались для тестирования реализации IPv6 от Microsoft Research и реализации IPv6 для Windows XP/Windows CE 4.1.

Прояснение стандарта и разработка тестового набора для IPMP-2

Стандарт управления и защиты прав интеллектуальной собственности на цифровые данные формата MPEG-2 [53-55] (MPEG-2 Intellectual Property Management and Protection, IPMP-2 [50]) представляет собой попытку создать гибкую систему управления правами на произведения в цифровой форме (Digital Rights Management, DRM), которая призвана обеспечить совместимость между системами создания, распространения и воспроизведения контента от различных производителей.

Введение в IPMP-2

Первая стандартизированная технология защиты MPEG-2, получившая название «ограниченный доступ» (conditional access, CA), не обеспечивает достаточный уровень совместимости между различными устройствами, работающими с данными в формате MPEG-2. Для воспроизведения контента, защищенного средствами некоторого производителя, требуется проигрыватель от того же производителя. Средства защиты, разработанные разными производителями, несовместимы между собой.

Одна из целей, поставленных перед разработчиками IPMP-2, было обеспечение совместимости между средствами создания произведений и проигрывателей: произведения, защищенные инструментальными средствами одного производителя, должны корректно воспроизводиться на устройстве другого производителя, если они соответствуют спецификации IPMP-2.

Стандарт IPMP-2 определяет порядок операций по поддержки защиты прав на стороне пользователя. Устройство с поддержкой IPMP-2 состоит из терминала и нескольких инструментов IPMP. Инструменты IPMP реализуют операции, необходимые для подготовки контента к воспроизведению, такие как авторизация пользователя, расшифрование контента, обработка водяных знаков, и т.п. Терминал перехватывает мультимедийные данные и направляет их для обработки в соответствующие инструменты. Результаты обработки (например, расшифрованные данные) возвращаются в терминал для дальнейшей обработки и воспроизведения. Инструменты IPMP взаимодействуют друг с другом и с терминалом посредством обмена сообщениями. Спецификация IPMP-2 описывает несколько наборов сообщений для различных целей, таких как аутентификация инструментов или уведомления о использовании определенного контента.

При создании контента в него включаются управляющая информация и индикаторы защиты. Эта информация предоставляет сведения о том, какие инструменты необходимо использовать, как их инициализировать и т.п. Устройство, поддерживающее IPMP-2, производит разбор данных, и, при необходимости, загружает инструменты IPMP из сети. Затем устройство создает экземпляры инструментов с заданными параметрами и начинает воспроизведение.

Формализация требований IPMP-2 проходила по процессу FOREST.

Составление каталога требований

Составление перечня регламентирующих документов. Большая часть требований к реализации IPMP-2 изложена в 11-й части стандарта MPEG-2 ISO/IEC 13181-11 [50]. На момент выполнения проекта эта часть стандарта проходила последние согласования (Final Draft) и обозначалась как ISO/IEC 13818-11 FDIS. Помимо этого, часть требований была извлечена из первой части ISO/IEC 138181-1 [51], описывающей общую архитектуру систем на MPEG-2, и поправок к этому документу (Amendments). Кроме ISO/IEC 13818, в ходе выполнения работ приходилось обращаться к стандарту ISO/IEC 14496 (MPEG-4) [84], так как исходно архитектура IPMP была разработана именно для MPEG-4 и только затем перенесена в MPEG-2. В общей сложности было изучено 8 документов суммарным объемом 300 страниц.

Были установлены контакты с разработчиками IPMP-2. Разработчики охотно отвечали на наши письма и активно участвовали в прояснении требований и подготовке исправлений и дополнений стандарта IPMP-2.

Идентификация требований. Разметке стандарта и идентификации функциональных требований предшествовал подготовительный этап, в ходе которого изучались MPEG-2 и MPEG-4, различные поясняющие и проектные документы по IPMP из архивов рабочей группы, занимавшейся разработкой IPMP-2. Так как в ИСП РАН до начала выполнения работ по формализации IPMP-2 не было экспертов в области современных форматов для передачи аудио- и видеоинформации, то исполнителям проекта пришлось самим стать такими экспертами. Затем была проведена разметка регламентирующих документов.

Внесение требований в каталог. Данный проект был нацелен на разработку тестового набора для проверки соответствия реализации IPMP-2 спецификации. К сожалению, полностью этой цели достичь не удалось по причинам нетехнического характера. Был разработан тестовый набор для подсистемы обработки управляющей информации IPMP-2 в битовых потоках. Соответственно, в каталог вносились только требования к этой подсистеме. Общий объем текстовой спецификации указанной подсистемы составил 4 страницы. Всего в каталог было внесено 11 требований.

Инспекция каталога. Один участник проекта исполнял роль аналитика и проводил идентификацию требований и внесение требований в каталог. Другой участник исполнял роль эксперта и проверял правильность выделения требований и каталога требований.

В ходе выделения требований было выявлено множество опечаток, нарушений стиля в оформлении ISO/IEC 13818-11, несколько ошибок в стандарте, а так же ряд проблем, которые потенциально могли привести к несовместимости контента и проигрывателя.

Построение концептуальной модели требований

Подсистема, выбранная для разработки тестового набора, осуществляет сложную обработку данных, полученных из битовых потоков. Для анализа требований к этим операциям были разработаны концептуальные модели в виде описаний алгоритмов на псевдокоде.

Анализ полноты и минимальности набора требований. Полученный псевдокод был проанализирован на предмет полноты. Анализ показал, что проект стандарта ISO/IEC 13818-11 FDIS не полон. В проекте стандарта были заданы синтаксис сообщений и структур данных, но почти полностью отсутствовали ограничения целостности. В результате можно было построить соответствующий стандарту битовый поток, для которого нет правил обработки. Например, спецификация IPMP-2 определяет структуру IPMP Tool List как контейнер структур типа IPMP Control Info, но в действительности IPMP Tool List предназначен для переноса данных более узкого класса — IPMP Tool Info, и стандарт не определяет обработку IPMP Tool List, содержащую управляющую информацию, отличную от IPMP Tool Info.

Из обсуждений с разработчиками IPMP-2 мы выяснили, что они считали ограничения целостности «очевидными» и по этой причине не стали вносить их в стандарт, равно как и правила обработки данных с нарушениями этих ограничений. Оказалось, что даже сами разработчики IPMP-2 не имели общего мнения о том, какие ограничения целостности есть, и как обрабатывать их нарушения.

Анализ адекватности концептуальных моделей. Псевдокод был также проанализирован на предмет адекватности. Как уже упоминалось выше, одна из главных задач IPMP-2 — обеспечить совместимость систем защиты прав собственности с системами воспроизведения. Поэтому при проверке адекватности особое внимание уделялось возможным проблемам, которые потенциально могли привести к тому, что произведение в формате MPEG-2, соответствующее спецификации IPMP-2, не будет воспроизводиться на некотором устройстве, также поддерживающем IPMP-2.

Например, была выявлена неполнота спецификации протокола взаимной аутентификации инструментов — ключевого механизма установления связи между инструментами и терминалом IPMP. Было показано, что описанная в стандарте спецификация протокола взаимной аутентификации не обеспечивает совместимость между реализациями различных производителей — возможны такие соответствующие стандарту реализации, которые не смогут успешно завершить процедуру взаимной аутентификации.

Разрешение проблем. За пояснениями по поводу выявленных проблем в стандарте мы обращались к разработчикам спецификации IPMP-2. Суммарный объем переписки составляет более 40 страниц.

Совместно с разработчиками IPMP-2 мы подготовили и подали в AVS DRM документы, в которых обозначили выявленные проблемы в спецификации IPMP-2 и предложения по их разрешению [84,85].

Построение формальной модели требований

Определение состава формального интерфейса. В состав формального интерфейса были включены функции обработки отдельных блоков управляющей информации IPMP.

Формализация контракта. Предусловия спецификационных функций накладывают ограничения на допустимость входных данных. Так как стандарт IPMP-2 определяет обработку не всех возможных данных, то предусловия ограничивают формальную модель только хорошо определенными требованиями. Данные, для обработки которых нет требований, отвергаются предусловием.

Постусловия определяют формальную модель требований, выделенных из стандарта IPMP-2.

Тестирование соответствия требованиям

Был разработан тестовый сценарий для тестирования соответствия обработки управляющей информации IPMP требованиям стандарта IPMP-2. Тестовый сценарий строит битовые потоки с управляющей информацией IPMP-2 и подает их на вход реализации. Так как во время выполнения работ не было ни одной работающей реализации IPMP-2, то для пробных запусков тестов был разработан прототип устройства MPEG-2 с поддержкой IPMP-2.

Результаты

Работа по формализации IPMP-2 проводилась для Рабочей группы по аудио и видео кодированию (AVS DRM) Министерства информационной индустрии КНР. Общий объем спецификации IPMP-2 составляет 30 страниц. По результатам проекта были написаны два технических отчета [84,85], поданных в AVS DRM, и прототип тестового набора для тестирования соответствия обработки управляющей информации IPMP в битовых потоках.

Помимо этого, в ходе проекта были получены следующие результаты.

- Выявлены несоответствия в описаниях синтаксиса данных битовых потоков IPMP. Например, спецификация IPMP допускает включение поля размером 65536 байт (длина поля задается 16-битным значением) в структуру, чей максимальный размер не может превосходить 256 байт.
- Выявлена неполнота спецификации протокола взаимной аутентификации инструментов — ключевого механизма установления связи между инструментами и терминалом IPMP.
- В спецификации IPMP-2 нечетко определены критерии корректности данных. Обсуждения с разработчиками IPMP-2 показали, что они предполагали множество неявных правил, определяющих корректные данные. Список обнаруженных неявных предположений о семантике структур данных и предложений по внесению соответствующих утверждений в спецификацию IPMP-2 приведен в отчете [85].

В процессе анализа спецификации были разработаны концептуальные модели поведения для отдельных функций IPMP-2 и формальные спецификации для наиболее полно заданной функции IPMP-2 — обработки управляющей информации в битовом потоке.

В ходе анализа спецификации IPMP-2 было выявлено множество опечаток в формальных определениях синтаксиса сообщения протокола. Ее изучение показало, что спецификация состоит из нескольких слабо связанных частей, причем некоторые из них противоречат друг другу, некоторые требования недоопределены или содержат ошибки.

Разработка открытого тестового набора для ОС Linux

Этот раздел описывает опыт применения процесса FOREST в проекте по разработке открытого тестового набора ОС Linux, выполняющегося при поддержке Федерального агентства по науке и инновациям Российской Федерации [86,87].

Сведения о проекте

Основными целями проекта по разработке открытого тестового набора ОС Linux являются:

- формализация, уточнение и пополнение набора стандартов, описывающих требования к поведению компонентов операционной системы Linux;
- разработка открытого систематического тестового набора, проверяющего соответствие различных реализаций операционной системы требованиям стандартов.

В течение первого этапа проекта, рассчитанного на 2005-2006 годы, планируется формализовать требования стандарта Linux Standard Base Core Specification 3.1 (LSB) [51,52] относительно функций прикладного программного интерфейса, содержащиеся в части III Base Libraries и части IV Utility Libraries. Эти части стандарта определяют требования к наличию и функциональности 1532 функций прикладного бинарного интерфейса операционной системы.

Проект проводится в координации с организациями-разработчиками стандарта LSB — Free Standards Group [88] и подразделением Open Group [89], отвечающим за работу с LSB.

В качестве основных результатов проекта ожидаются формальные спецификации требований стандарта к упомянутым функциям и систематизированный тестовый набор, предназначенный для проверки соответствия реализаций ОС Linux требованиям стандарта. Еще одним важным результатом будет список замечаний к тексту стандарта и рекомендации по его улучшению.

Процесс создания тестового набора.

Процесс создания тестового набора в проекте соответствует процессу FOREST и имеет дополнительный этап, связанный с разработкой тестов. Таким образом, он состоит из 4-х этапов.

1. Составление каталога требований.
2. Построение концептуальной модели.
3. Построение формальной модели.
4. Разработка тестовых сценариев.

В проекте выделены следующие роли — аналитики, создающие основные документы (разметку стандарта, каталог требований, спецификации, сценарии и другие компоненты тестовой системы), и эксперты, которые контролируют процесс разработки и рецензируют то, что делают аналитики. Все функции прикладного бинарного интерфейса операционной системы разделены на группы связанных функций. Для каждой группы функций этапы процесса выполняются последовательно, но одновременно могут разрабатываться спецификации и тестовые сценарии для разных групп. За каждую группу отвечает несколько человек — аналитики и два эксперта, при этом эксперты одновременно участвуют в работе над несколькими группами функций.

Далее мы рассмотрим деятельности, выполняющиеся на каждом из этапов процесса.

Составление каталога требований

Для составления каталога используется текст стандарта (а точнее, его HTML-представление), в котором элементарные требования помечаются цветом и снабжаются специальным идентификатором. Для этого используется HTML-редактор с разработанными специально для этих целей средствами разметки.

Отрывки текста, встречающиеся в разных разделах описания функции, могут совпадать по смыслу и определять одно и то же требование. В этом случае они обозначаются одним идентификатором.

Одному элементарному требованию может быть присвоено несколько идентификаторов, если оно относится к описанию поведения нескольких функций сразу.

Требования разделяются на требования к реализации функции, описываемой стандартом, и требования к приложению, описывающие правильное использование данной функции. Эта

информация отражается в структуре идентификатора — для требований к приложению идентификатор содержит специальный префикс.

Кроме того, идентификатор позволяет отразить связь между требованиями. Требование, сформулированное в одном разделе стандарта, может затем уточняться требованиями в других местах стандарта. Идентификатор требования, уточняющего какое-то другое, в качестве префикса имеет идентификатор основного требования. В приведенном ниже примере первое общее требование (помеченное идентификаторами fork.01;vfork.01) детализируется последующими фразами, поэтому два последних требования помечены как уточняющие. Кроме того, каждое требование одновременно описывает и поведение функции fork, и поведение функции vfork , поэтому используются два идентификатора для одного элементарного требования.

```
{fork.01;vfork.01} The fork() function shall create a new process. The new process
(child process) shall be an exact copy of the calling process (parent process) except
as detailed below:
      {fork.01.01;vfork.01.01} The child process shall have a unique process ID.
      {fork.01.02;vfork.01.02} The child process ID also shall not match any active
      process group ID.
```

Наконец, есть возможность расширять список требований к функции своими собственными формулировками, если, например, есть информация, что некоторое требование неявно подразумевается в стандарте.

На основе размеченного текста стандарта затем генерируются каталог требований, который используется в последующих этапах проекта.

В процессе разметки текста стандарта и выделения требований однозначность их формулировок может нарушиться, если уточнение некоторого требования не попало в каталог. Наличие разметки стандарта позволяет проверить, что уточнение не пропущено, а идентификатор правильно отражает связь между уточняемым и уточняющим требованиями.

При рецензировании проверяется непротиворечивость создаваемого каталога, в частности, могут быть определены неправильно установленные связи между требованиями. Для обеспечения минимальности эксперт проверяет, что одинаковые требования помечены одним идентификатором.

Построение концептуальной модели

На этом этапе для каждой группы функций выбираются подходы к специфицированию и тестированию. В отличие от других этапов процесса, где основные результаты создаются аналитиком, а затем контролируется экспертом, концептуальная модель создается в результате совместного обсуждения аналитиков и экспертов.

Из каталога требований извлекаются все требования, относящиеся к функциям группы и группе в целом; затем эти требования обсуждаются, анализируются и классифицируются по сложности, проверяемости и т.д., что впоследствии также используется для планирования работ по проекту. Аналитики и эксперты обсуждают и разрабатывают модель общих данных, которая необходима для адекватного отражения поведения этой группы функций, а также строят модель поведения для функций группы, основанную на этой модели данных. В результате может произойти перегруппировка функций — в группу могут быть добавлены функции, работающие с теми же данными. Полученные модели используются на следующем этапе для построения формальной модели. В ходе обсуждения моделей могут быть выявлены неясности и неоднозначности в стандарте, которые были пропущены при создании каталога требований. К таким неясностям в первую очередь относится отсутствие требований для некоторого аспекта поведения системы, выявляемое при построении модели поведения.

Хотя результатом данного этапа являются построенные модели данных и модели поведения, конкретная форма представления этих моделей в проекте не фиксируется. Это могут быть как комментарии к требованиям, схемы и рисунки, так и описания в псевдокоде.

Построение формальной модели

Для построения формальной модели в проекте используется инструментальная поддержка. Размеченный текст стандарта автоматически обрабатывается, и на основе разметки генерируется заготовка формальной спецификации, которая затем дорабатывается аналитиком.

Аналитик добавляет формальное описание модели данных, построенной на предыдущем этапе. Для каждого требования из каталога в сгенерированной заготовке спецификации появляется комментарий, содержащий текст требования, и инструкции, предназначенные для проверки этого требования и содержащие его идентификатор. Аналитик дописывает проверку этого требования в виде некоторого предиката в рамках принятой модели данных.

При тестировании ставится задача покрыть все требования стандарта, однако аналитик может определить дополнительные требования к полноте тестирования и описать их в специальном разделе формальной спецификации.

При формализации проверок могут быть обнаружены неясности, неточности и неоднозначности стандарта, пропущенные на этапе построения концептуальной модели.

Качество созданных формальных спецификаций впоследствии контролируются экспертом. Он проверяет правильность формальной записи по отношению к требованиям, описанным в каталоге, к тексту стандарта, и по отношению к собственному пониманию требований к системе. Проверка полноты отражения требований поддерживается инструментально — автоматически предоставляется информация о требованиях каталога, не отраженных в спецификациях. Кроме того, так как спецификации строятся на основе сгенерированного шаблона, это сокращает вероятность пропуска требований из каталога.

При рецензировании эксперт дополнительно проверяет соблюдение правил и корректность именования, проверяет, все ли аспекты поведения специфицированы, а также проверяет спецификации на непротиворечивость и минимальность.

Результаты проекта

На текущий момент проект еще не завершен, но уже полученные результаты позволяют достаточно уверенно предсказать общие итоги проекта.

В его ходе проанализировано около 6000 страниц стандарта LSB [51] и связанных с ним стандартов POSIX [90,91], ISO C [92] и пр. В результате анализа выделено около 17000 элементарных требований к поведению функций базовых библиотек Linux и к структурам данных, с которыми они работают. Обнаружено около 100 различного рода недочетов в стандарте: неоднозначностей, противоречий, неполных описаний требований и т.п.

Разработанный в ходе проекта тестовый набор нацелен на покрытие всех выделенных требований и обеспечивает его с точностью до реализуемости требования в заданной конфигурации ОС Linux. Сравнение с известными тестовыми наборами для библиотек Linux — LTP [93], тестовый набор для тестирования соответствия LSB [94] и тестовый набор разработчиков библиотеки GLIBC [95] показывает, что достигаемое покрытие кода существенно превосходит покрытие, получаемое при выполнении первых двух наборов и сравнимо с достигаемым последним тестовым набором. Последний факт показывает очень высокое качество полученных тестов, поскольку они строились только на основе требований стандарта LSB, в то время как тесты из набора GLIBC написаны для отладки этой библиотеки ее разработчиками, которым известны все детали ее реализации.

Заключение

В данной работе мы попытались рассмотреть проблемы, встающие при формализации требований в практических, т.е. не исследовательских, проектах по разработке программного обеспечения, а также способы решения этих проблем. Построение формальных моделей используется в таком контексте достаточно редко именно потому, что большинство практиков хорошо представляет

себе трудности работы с такими моделями, но не вполне осведомлено о том, что достаточно эффективные методы их преодоления уже разработаны.

Помимо возможности автоматизации решения ряда трудоемких задач разработки, формализация — за счет радикального изменения структуры знаний — позволяет также более отчетливо и глубоко понять, что собственно требуется от создаваемой системы, понять проблемы и нужды пользователей. Поэтому ее адекватное использование в проекте позволяет повысить вероятность его успеха.

Основное содержание данной статьи связано с процессом FOREST, являющимся примером процесса формализации требований и нацеленным на обеспечения максимальной адекватности построенной модели. Хотя этот конкретный процесс используется, прежде всего, для автоматизации разработки тестов, лежащие в его основе идеи и многие его техники можно использовать и для построения формальных моделей требований в более широком контексте. Например, заменяя используемый в нем формализм программных контрактов на комбинацию из временной логики и конечных автоматов, можно получать модели, удобные для верификации (проверки на моделях, model checking) свойств предлагаемых алгоритмов.

Рассматриваемые примеры применения описанного процесса подтверждают как его практичность, так и жизнеспособность более общей идеи использования формальных моделей при индустриальной разработке ПО.

Благодарности. Авторы выражают признательность А. К. Петренко и А. И. Гриневичу за высказанные ими замечания к содержанию предварительных вариантов этой работы.

Литература

[1] IEEE 830-1998. Recommended Practice for Software Requirements Specifications. New York: IEEE, 1998.
[2] IEEE 1233-1998. Guide for Developing System Requirements Specifications. New York: IEEE, 1998.
[3] IEEE 1003.1-2004. Information Technology — Portable Operating System Interface (POSIX). New York: IEEE, 2004.
[4] ISO/IEC 9899-1999. Programming Languages — C. Geneve: ISO, 1999.
[5] I. Graham. Requirements Engineering and Rapid Development: An Object-Oriented Approach. Addison-Wesley, 1998.
[6] И. Грэхем. Объектно-ориентированные методы. Принципы и практика. 3-е издание. Москва: Вильямс, 2004.
[7] Guide to the Software Engineering Body of Knowledge: 2004 Edition — SWEBOK. IEEE, 2005.
[8] А. Якобсон, Г. Буч, Дж. Рамбо. Унифицированный процесс разработки программного обеспечения. СПб.: Питер, 2002.
[9] P. Kruchten. The Rational Unified Process: An Introduction, Third Edition. Addison-Wesley Professional, 2003.
[10] Э. Халл, К. Джексон, Дж. Дик. Разработка и управление требованиями. Практическое руководство пользователя. Telelogic, 2005.
[11] DOORS Reference Manual. QSS, Oxford Science Park, Oxford OX4 4GA, 1993-1998.
[12] G. P. Mullery. CORE — A Method for Controlled Requirements Expression. In Proc/ of 4-th International Conference on Software Engineering, pp. 126–135, IEEE, CS Press, 1979.
[13] P. Curwen. System Development Using the CORE Method. BAE System technical report, 1993.
[14] P. B. Checkland. Systems Thinking, Systems Practice. Chichester, Wiley, 1981.
[15] B. Wilson. Systemы: Concepts, Methodologies and Applications. Wiley, NY, 1984.
[16] P. B. Checkland, J. Scholes. Soft System Methodology in Action. Chichester, Wiley, 1990.
[17] C. W. George, R. E. Milne. Specifying and Refining Concurrent Systems — an Example from the RAISE Project. In Proceedings of 3rd Refinement Workshop, Springer-Verlag, 1990.
[18] The RAISE Method Group. The RAISE Development Method. Prentice Hall, 1995.

[19] J. Woodcock, J. Davies. Using Z: Specification, Refinement, and Proof. Prentice Hall, 1996.
[20] C. Jones. Systematic Software Development Using VDM. Prentice Hall, 1990.
[21] J.-R. Abrial. The B-Book: Assigning Programs to Meanings. Cambridge University Press, 1996.
[22] J. A. Goguen, C. Linde. Techniques for Requirements Elicitation. In Proc. of Requirements Engineering'1993. S.Fickas, A. Finkelstein, eds. IEEE Computer Society, pp. 152-164, 1993.
[23] M. J. Cetron, G. B. Bernstein. SEER: A Delphic Approach Applied to Information Processing. TFSC 1, No. 1, Spring 1969.
[24] O. Helmer. The Use of Expert Opinion in International Relations Forecasting. Center for Futures Research, Graduate School of Business Administration, University of Southern California, Los Angeles, July 1973.
[25] A. Dardenne, A. van Lamsweerde, S. Fickas. Goal-Directed Requirements Acquisition. Science of Computer Programming Vol. 20, North Holland, 1993, pp. 3-50.
[26] P. Bertrand, R. Darimont, E. Delor, P. Massonet, A. van Lamsweerde. GRAIL/KAOS: an environment for goal drivent requirements engineering. Proceedings ICSE'98, 20-th International Conference on Software Engineering, IEEE-ACM, Kyoto, April 1998.
[27] W. Kunz, H. W. J. Rittel. Issues as Elements of Information Systems. Working Paper 131, Universitat Stuttgart, Institut fur Grundlagen der Planung. 1970.
[28] A. Finkelstein et al. Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. Int. J. of Software Engineering and Knowledge Engineering, 2(1):31-58, 1992.
[29] G. Kotonya, I. Sommerville. Requirements Engineering with Viewpoints. BCS/IEEE Software Eng. J., 11(1):5-18, 1996.
[30] J. Annett, K. D. Duncan. Task Analysis and Training Design. Occupational Psychology, 41, pp. 211-221, 1967.
[31] J. Annett, K. D. Duncan, M. J. Gray. Task Analysis. London: HMSO, 1971.
[32] K. D. Duncan. Strategies for Analysis of the Task. In J. Hartley, ed. Strategies for Programmed Instruction: An Educational Technology. London: Butterworth, 1972.
[33] D. Benyon. The Role of Task Analysis in Systems Design. Interacting with Computers, 4(1):102-123, 1992.
[34] J. Dehoney. Cognitive Task Analysis: Implications for the Theory and Practice of Instructional Design. In Proc. of Selected Research and Development Presentations at the 1995 National Convention of the Association for Educational Communications and Technology, M. R. Simonson, M. L. Anderson, eds. Washington, DC: Association for Educational Communications and Technology, 1995.
[35] T. C. Ormerod. Using Task Analysis as a Primary Design Method: The STG Approach. In J. M. C. Schaagen, S. F. Chipman, V. L. Shalin, eds. Cognitive Task Analysis. Mahwah, NJ: Lawrence Erlbaum Associates, Inc. pp. 181-200, 2000.
[36] W. E. Montague. Elaborative Strategies in Verbal Learning and Memory. In G. Bower, ed. The Psychology of Learning and Motivation (Vol. 6). Academic Press, New York, 1972.
[37] K. A. Ericsson, R. J. Crutcher. Introspection and Verbal Reports on Cognitive Processes — Two Approaches to the Study of Thought Processes: A Response to Howe. New Ideas in Psych. 9:57-71, 1991.
[38] K. A. Ericsson, H. A. Simon. Protocol Analysis: Verbal Reports as Data. MIT Press, Cambridge, MA, 1993.
[39] R. J. Crutcher. Telling What We Know: The Use of Verbal Report Methodologies in Psychological Research. Psych. Sci. 5: 241-244, 1994.
[40] G. A. Kelly. The Psychology of Personal Constructs. NY: W. W. Norton, 1955.
[41] R. P. Wolf, H. S. Delugach. Knowledge Acquisition via Tracked Repertory Grids. Computer Science Dept., Univ. Alabama in Huntsville, 1996.
[42] G. Rugg, C. Corbridge, N. P. Major, A. M. Burton, N. R. Shadbolt. A comparison of sorting techniques in knowledge elicitation. Knowledge Acquisition, 4(3), pp. 279-291, 1992.
[43] T. J. Reynolds, J. Gutman. Laddering Theory, Method, Analysis, and Interpretation. Journal of Advertising Research, vol. 28, pp. 11-31, 1988.

[44] G. Rugg, M. Eva, A. Mahmood, N. Rehman, S. Andrews, S. Davies. Eliciting information about organisational culture via laddering. Proceedings of the EMRPS'99 workshop, Venice, 25-26 November, 1999.
[45] B. R. Gaines, M. L. G. Shaw. Comparing the Conceptual Systems of Experts. In IJCAI'89, pp. 633-638, 1989.
[46] E. Hudlicka. Summary of Knowledge Elicitation Techniques for Requirements Analysis, Course Material for Human Computer Interaction, Worcester Polytechnic Institute, 1997.
[47] C. Goodwin, J. Heritage. Conversation Analysis. Annual Review of Anthropology, 19:283-307, 1990.
[48] M. Jirotka. Ethnometodology and Requirements Engineering. Technical Report, Centre for Requirements and Foundations, Oxford University, Computing Lab, 1991.
[49] Microsoft Corporation. Introduction to IP version 6. Доступна по http://www.microsoft.com/technet/itsolutions/network/ipv6/introipv6.mspx.
[50] ISO/IEC 13818-11:2004. Information technology — Generic coding of moving pictures and associated audio information — Part 11: IPMP on MPEG-2 systems. Geneve: ISO, 2003.
[51] ISO/IEC 23360-1-8:2005, Linux Standard Base (LSB) Core Specification 3.1. Geneve: ISO, 2005.
[52] http://www.linuxbase.org/spec
[53] ISO/IEC 13818-1:2000. Information technology — Generic coding of moving pictures and associated audio information — Part1: Systems. Geneve: ISO, 2000.
[54] ISO/IEC 13818-2:2000. Information technology — Generic coding of moving pictures and associated audio information — Part 2: Video. Geneve: ISO, 2000.
[55] ISO/IEC 13818-3:1998. Information technology — Generic coding of moving pictures and associated audio information — Part 3: Audio. Geneve: ISO, 1998.
[56] I. Bourdonov, A. Kossatchev, V. Kuliamin, and A. Petrenko. UniTesK Test Suite Architecture. Proc. of FME 2002. LNCS 2391, pp. 77-88, Springer-Verlag, 2002.
[57] В. В. Кулямин, А. К. Петренко, А. С. Косачев, И. Б. Бурдонов. Подход UniTesK к разработке тестов. Программирование, 29(6):25–43, 2003.
[58] V. Kuliamin, A. Petrenko, N. Pakoulin. Practical Approach to Specification and Conformance Testing of Distributed Network Applications. In M. Malek, E. Nett, N. Suri, eds. Service Availability. LNCS 3694, pp. 68–83, Springer-Verlag, 2005.
[59] V. Kuliamin, A. Petrenko, N. Pakoulin. Extended Design-by-Contract Approach to Specification and Conformance Testing of Distributed Software. Proc. of 9-th WMSCI, Orlando, USA, July 2005, v. VII. Model Based Development and Testing, pp. 65-70.
[60] ISO/IEC 10731:1994. Information technology — Open Systems Interconnection — Basic Reference Model — Conventions for the definition of OSI services. 1994.
[61] RFC 2292. W. Stevens, M. Thomas. Advanced Sockets API for IPv6. February 1998.
[62] RFC 2373. R. Hinden, S. Deering. IP Version 6 Addressing Architecture. July 1998.
[63] RFC 2460. S. Deering, R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. December 1998.
[64] RFC 2461. T. Narten, E. Nordmark, W. Simpson. Neighbor Discovery for IP Version 6 (IPv6). December 1998.
[65] RFC 2462. S. Thomson, T. Narten. IPv6 Stateless Address Autoconfiguration. December 1998.
[66] RFC 2463. A. Conta, S. Deering. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. December 1998.
[67] RFC 2464. M. Crawford. Transmission of IPv6 Packets over Ethernet Networks. December 1998.
[68] RFC 2553. R. Gilligan, S. Thomson, J. Bound, W. Stevens. Basic Socket Interface Extensions for IPv6. March 1999.
[69] RFC 2675. D. Borman, S. Deering, R. Hinden. IPv6 Jumbograms. August 1999.
[70] RFC 2710. S. Deering, W. Fenner, B. Haberman. Multicast Listener Discovery (MLD) for IPv6. October 1999.
[71] RFC 3513. R. Hinden, S. Deering. Internet Protocol Version 6 (IPv6) Addressing Architecture. April 2003.

[72] D. B. Johnson, C. Perkins. Mobility Support in IPv6. IETF Mobile IP Working Group, Internet Draft.
Доступен как http://www3.ietf.org/proceedings/01mar/I-D/mobileip-ipv6-13.txt.
[73] RFC 3775. D. Johnson, C. Perkins, J. Arkko. Mobility Support in IPv6. June 2004.
[74] RFC 1213. K. McCloghrie, M. T. Rose. Management Information Base for Network Management of TCP/IP-based internets: MIB-II. March 1991.
[75] RFC 2011 K. McCloghrie, Ed. SNMPv2 Management Information Base for the Internet Protocol using SMIv2. November 1996.
[76] RFC 2465. D. Haskin, S. Onishi. Management Information Base for IP Version 6: Textual Conventions and General Group. December 1998.
[77] RFC 2579. K. McCloghrie, D. Perkins, J. Schoenwaelder. Textual Conventions for SMIv2. April 1999.
[78] RFC 2665. J. Flick, J. Johnson. Definitions of Managed Objects for the Ethernet-like Interface Types. August 1999.
[79] RFC 2863. K. McCloghrie, F. Kastenholz. The Interfaces Group MIB. June 2000.
[80] RFC 3635. J. Flick. Definitions of Managed Objects for the Ethernet-like Interface Types. September 2003.
[81] RFC 4293. S. Routhier, Ed. Management Information Base for the Internet Protocol (IP). April 2006.
[82] RFC 4295. G. Keeni, K. Koide, K. Nagami, S. Gundavelli. Mobile IPv6 Management Information Base. April 2006.
[83] ISO/IEC 14496-1:2001, Information technology — Coding of audio-visual objects — Part 1: Systems. Geneve: ISO, 2001.
[84] N. Pakoulin, V. Omelchenko, A. Koptelov, A. Petrenko, A. Kossatchev, C. Cheng. MPEG-2 IPMP Conformance Test Suite Development. AVS M1263: 2004-06.
[85] N. Pakoulin, A. Monakhov, C. Cheng, J. Wen. Enhancing IPMP-2 for Conformance Testing. AVS M1487: 2004-12.
[86] http://www.linuxtesting.ru.
[87] А. Гриневич, В. Кулямин, Д. Марковцев, А. Петренко, В. Рубанов, А. Хорошилов. Использование формальных методов для обеспечения соблюдения программных стандартов. Труды ИСП РАН, 2006.
[88] http://www.freestandards.org/
[89] http://www.opengroup.org/
[90] ISO/IEC 9945-1:2003. Information technology — Portable Operating System Interface (POSIX) — Part 1: Base Definitions. Geneve: ISO, 2003.
[91] ISO/IEC 9945-2:2003. Information technology — Portable Operating System Interface (POSIX) — Part 2: System Interfaces. Geneve: ISO, 2003.
[92] ISO/IEC 9899:1999. Programming Languages — C. Geneve: ISO, 1999.
[93] http://ltp.sourceforge.net/
[94] http://www.linuxbase.org/download/#test suites
[95] ftp://ftp.gnu.org/gnu/glibc/