Алгоритмическое обеспечение моделирования

DOI 10.24412/2221-2574-2022-4-60-68 УДК 004.582

ПРАВИЛА ВАЛИДАЦИИ ПРОХОЖДЕНИЯ ШАГОВ ВАРИАНТА ИСПОЛЬЗОВАНИЯ НА ОСНОВЕ ТРЁХЗНАЧНОЙ ЛОГИКИ

Дубов Илья Ройдович
доктор технических наук, профессор, профессор кафедры вычислительной техники и систем управления ФГБОУ ВО «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых».
E-mail: dubov@vlsu.ru

Куликов Константин Владимирович
кандидат технических наук, заведующий кафедрой вычислительной техники и систем управления ФГБОУ ВО «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых».
E-mail: kulikov@vlsu.ru

Адрес: 600000, Российская Федерация, г. Владимир, ул. Горького, д. 87.

Аннотация: Предлагается использовать аппарат трёхзначной логики для валидации прохождения шагов варианта использования. Предполагается, что между вариантом использования и реализацией программы имеется однозначное соответствие, и правила валидации являются частью функциональных требований к системе. Значение предиката валидации соответствует одному из трёх случаев: (1) шаг завершён корректно, (2) возникла ошибка при выполнении шага, приводящая к альтернативному пути, (3) неопределённость, которая требует принятия решения актантом для продолжения выполнения текущего пути или перехода к альтернативному пути. В каждом исходном атомарном факте, являющимся параметром предиката валидации, логическое значение сопровождается заключением на естественном языке. При выполнении операций над логическими значениями выполняется также обработка текстовых заключений в соответствии с предлагаемыми в работе правилами. В результате формируется итоговое объяснение логического значения предиката в понятной для актанта форме на естественном языке.

Ключевые слова: трёхзначная логика, логика высказываний, вариант использования, валидация программы, актант.

Вопрос реализации валидации является очень важным в разработке программного обеспечения. Известны различные подходы к решению этой проблемы. В рамках объектно-ориентированного программирования естественным подходом считается связывание правил валидации с реализацией бизнес-объектов [1], в которой правила представляются методами соответствующих программных классов. Эти методы выполняются для определения возможного нарушения заданных ограничений при создании нового объекта, изменении или удалении существующего объекта. Другой принятый подход к валидации программ основан на шаблоне проектирования «Спецификация» [2] и предполагает формирование правил валидации в виде предикатов булевой алгебры. В этом случае правило может применяться не только к отдельным объектам, но и к совокупности объектов. В данной работе, в отличие от упомянутых выше подходов, предлагается рассматривать правила проверки правильности выполнения программы как составную часть функциональных требований, предъявляемых к системе.

В литературе по разработке программного обеспечения рассматриваются различные подходы к формированию функциональных тре-

бований [3]. Применение модели вариантов использования [4] в качестве указанных требований в настоящее время представляется основным подходом. В рамках некоторых технологий разработки вместо понятия «вариант использования» могут применяться другие родственные понятия, например, «пользовательская история» [5], но различия между ними не существенные. Вариант использования описывает последовательность шагов взаимодействия системы и актанта, приводящую к важному для актанта результату [4, 6]. Реализация программной системы должна обеспечивать выполнение совокупности функциональных требований. Для этого спецификации вариантов использования должны быть преобразованы в программный код.

Это преобразование может осуществляться с различной степенью формализма. Например, в [7, 8] имеются не формальные практические рекомендации для разработчиков по осуществлению такого перехода. А в рамках концепции архитектуры, управляемой моделью, разработаны подходы формального перевода текстовой спецификации варианта использования в модель анализа и в последующую реализацию системы [9], но при этом на текст варианта использования накладываются определённые ограничения.

Полагая, что между спецификацией варианта использования и его программной реализацией имеется однозначное соответствие, далее будем говорить о выполнении варианта использования актантом и системой, понимая под этим взаимодействие реального актанта с программной реализацией модели вариантов использования. При этом реальный актант может быть не только пользователем, но и внешней технической системой при условии, что эта внешняя система способна принимать решение на основе результатов валидации шагов вариантов использования.

В процессе выполнения программы в валидации могут участвовать отдельные поля объектов, объекты в целом, граф объектов, база данных и другие элементы, совокупное состояние которых оценивается по определённым правилам. В рамках парадигмы вариантов использования процедура валидации выявляет возможность продолжения выполнения варианта использования на его определённых шагах. При таком понимании валидности, правила должны быть привязаны к шагу варианта использования, или более конкретно — к совокупности объектов, существующих в системе на данном шаге, а не к классам объектов. Поэтому могут быть ситуации, в которых на разных шагах варианта использования правила валидации для одних и тех же объектов оказываются разными, что существенно отличается от подхода, в котором правила валидации связаны с классами объектов [1].

На шаге варианта использования актант может столкнуться со следующими альтернативами продолжения:

1) Для продолжения не требуется вмешательство актанта, система переходит к выполнению следующего шага. В данной ситуации иногда имеет смысл проинформировать актанта о деталях результата выполненной валидации, если результат не очевиден и информация о нем может быть полезна актанту для прохождения последующих шагов. В большинстве же случаев детали успешной валидации актанту не требуются.

2) Вариант использования не может быть продолжен по ожидаемому актантом пути. Система должна сообщить об этом актанту и перейти к альтернативному потоку. В данном случае актант сможет использовать информацию о результате валидации для осмысленного продолжения работы в соответствии с альтернативным потоком. В простейшем случае альтернативный поток может быть возвратом к повторному выполнению предыдущих шагов, на которых актант должен будет принять новые решения. В более сложном случае результат валидации будет использован актантом для достижения цели альтернативным путём.

3) Система не может определить, по какому пути должен выполняться вариант использования, чтобы удовлетворить потребности актан-

та. Актант должен разрешить эту неопределённость, руководствуясь результатом валидации. Наличие данной ситуации является чертой интерактивной программы. В ней возможность актанта принимать решение дополняет ту часть программы, в которой разработчики не могут создать реализацию системы, способную решить задачу без внешнего вмешательства. Неопределённость может быть формализована различными средствами [10–12]. Здесь будем использовать понятие неопределённости в соответствии с логикой Лукасевича [13] — это промежуточное значение между истинной и ложью. Заметим, что в некоторых приложениях трёхзначной логики в области разработки программ используют иное значение неопределённости — в соответствии с логикой Клини, где неопределённость означает, что значение может быть только или истиной, или ложью, но неизвестно каково оно в момент рассмотрения [14, 15].

Результат валидации назовём заключением. Заключение (v,s) состоит из двух частей:

1) Значение v — одна из трёх констант {false, true, undef} трёхзначной логики, обозначающая возможность продолжения варианта использования: не возможно, возможно, не определено.

2) Высказывание s — это объект, содержащий информацию, объясняющую, почему заключение приняло данное значение v. Высказывание преобразуется программой в форму, понятную для актанта, например в текст. Возможны и другие способы предоставления высказываний актанту, например, в форме звукового сообщения или графического изображения.

Валидация программы заключается в вычислении предиката, который переводит значения переменных, определяющих состояние программы, в заключение. В свою очередь, этот предикат может быть составным, т.е. логической формулой, состоящей из предикатов, формирующих частные заключения о переменных программы. Заключение о состоянии какого-то параметра программы, которое не делится на составные части, для краткости назовем фактом. Каждый факт формулируется так, как если бы этот факт был единственным заключением для осуществления валидации. Например, если валидируется поле с указанием состояния здоровья на форме ввода данных о пациенте для регистрации в больнице, то факт со значением true должен быть (true, Болен), а для формы ввода данных о космонавте, отправляющемся в полёт, факт со значением true для такого же поля должен быть (true, Здоров).

В трёхзначной логике могут использовать различные полные наборы операций [16]. Однако естественными и понятными логическими операциями, которыми для валидации пользуются разработчики программ, являются отрицание ~, конъюнкция ⋀ и дизъюнкция ⋁ [17]. В дополнение к ним предлагается использовать логические операции возможности ◊ и необходимости □ [18]. Операция возможности позволяет кратко записать предикат с допустимым неопределённым значением, а с помощью операции необходимости записывается предикат, не допускающий неопределённое значение.

Для того чтобы можно было сформировать высказывание по результату логического вывода, предлагается сохранять в объекте заключения дерево разбора применённых операций и фактов. Листьями дерева разбора являются факты. Вхождение любой одноместной операции в дерево разбора будем перемещать до листьев дерева при помощи известных преобразований. С этой целью для операции ~ применяются формулы де Моргана:

$$\sim(x \vee y) \leftrightarrow \sim x \wedge \sim y, \quad (1)$$

$$\sim(x \wedge y) \leftrightarrow \sim x \vee \sim y. \quad (2)$$

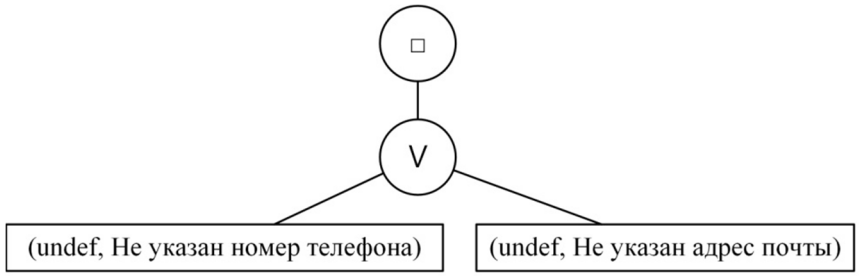
Для операций ◊ и □ применяются следующие формулы:

$$\Diamond(x \vee y) \leftrightarrow \Diamond x \vee \Diamond y, \quad (3)$$

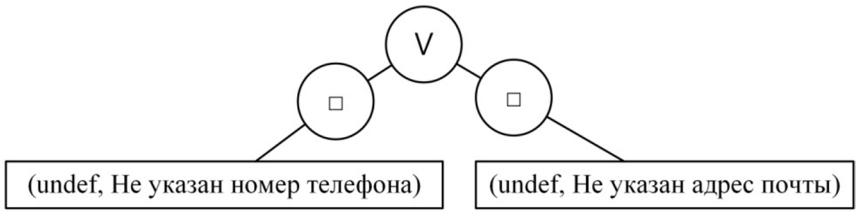
$$\Diamond(x \wedge y) \leftrightarrow \Diamond x \wedge \Diamond y, \quad (4)$$

$$\Box(x \vee y) \leftrightarrow \Box x \vee \Box y, \quad (5)$$

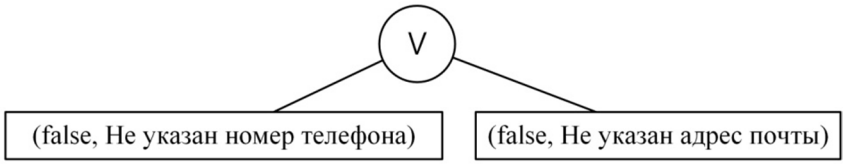
$$\Box(x \wedge y) \leftrightarrow \Box x \wedge \Box y. \quad (6)$$



а) исходное дерево разбора выражения



б) эквивалентно преобразованное дерево после распределения унарной операции по операндам



в) дерево после применения унарной операции к фактам

Рис. 1. Редуцирование унарных операций.

В результате указанного перемещения все одноместные операции оказываются в дереве разбора перед фактами, а их применение к фактам изменяет логическое значение, связанное с заключением. Этот подход позволяет свести составное высказывание к понятным пользователю операциям конъюнкции и дизъюнкции, связывающим элементарные высказывания без упоминания названий унарных операций. На рис. 1 приводится пример изменения дерева выражения □ ((undef, Не указан номер телефона) ∨ (undef, Не указан адрес почты)), в котором операция необходимости из корня перемещается к листьям — фактам, после чего оба факта заменяются заключениями со значениями false.

В таблицах 1, 2, 3 показано как формируется составное высказывание, получаемое при использовании логических операций. Двуместная операция (конъюнкция или дизъюнкция) возвращает значение, соответствующее операции трёхзначной логики над значениями аргументов. В высказывание результата операции попадает высказывание того аргумента, который определил результат. Например, в операции (true, Правильный номер телефона) ∨ (undef, Не указан адрес почты), значение результата определяется первым аргументом, поэтому результирующее заключение будет (true, Правильный номер телефона). Если же результат двуместной операции определяется значениями обоих аргументов, то высказывание результата составляется из высказываний обоих аргументов со связкой, соответствующей операции. Например, результат операции (false, Неправильный формат номера телефона) ∧ (false, Неправильный формат адреса почты) есть высказывание со значением false, которое

Таблица 1. Заключения, порождаемые операцией ⋀

Оператор ⋀	**(true, s2)**	**(undef, s2)**	**(false, s2)**
(true, s1)	(true, s1 ⋀ s2)	(undef, s2)	(false, s2)
(undef, s1)	(undef, s1)	(undef, s1 ⋀ s2)	(false, s2)
(false, s1)	(false, s1)	(false, s1)	(false, s1 ⋀ s2)

Таблица 2. Заключения, порождаемые операцией ⋁

Оператор ⋁	**(true, s2)**	**(undef, s2)**	**(false, s2)**
(true, s1)	(true, s1 ⋁ s2)	(true, s1)	(true, s1)
(undef, s1)	(true, s2)	(undef, s1 ⋁ s2)	(undef, s1)
(false, s1)	(true, s2)	(undef, s2)	(false, s1 ⋁ s2)

Таблица 3. Заключения, порождаемые одноместными операциями

x	***~x***	***◊x***	***□x***
(true, s)	(false, s)	(true, s)	(true, s)
(undef, s)	(undef, s)	(true, s)	(false, s)
(false, s)	(true, s)	(false, s)	(false, s)

определено значениями и первого, и второго операндов. Поэтому результирующее заключение будет (false, Неправильный формат номера телефона ⋀ Неправильный формат адреса почты). Одноместная логическая операция изменяет только логическое значение заключения, а высказывание остаётся без изменения, т.к. именно то, что оно выражает, является объяснением результата. Подобным образом в работе [19] определяется информативность высказываний при формировании вывода в экспертной системе, основанной на трёхзначной логике.

Рассмотрим в качестве примера ввод данных о сотруднике в некоторой системе учёта кадров. Пусть заполняются следующие поля формы:

1) NA – фамилия сотрудника;

2) PH – номер телефона;

3) EM – адрес электронной почты.

Предикаты, с помощью которых формируются факты – частные заключения о состоянии элементов формы, запишем следующим образом:

1) f1(NA) возвращает: если не задано NA, то (undef, Не указана фамилия), если NA содержит от 2 до 20 символов, то (true, Правильная фамилия), иначе (false, Ошибка в фамилии).

2) f2(PH) возвращает: если не задано PH, то (undef, Не указан номер телефона), если PH содержит 11 цифр, то (true, Номер телефона указан), иначе (false, Не правильный формат номера телефона).

3) f3(EM) возвращает: если не задано EM, то (undef, Не указан адрес почты), если EM содержит символ @, то (true, Адрес почты указан), иначе (false, Неправильный формат адрес почты).

Правило для валидации программы на шаге сохранения значений полей формы в базу данных в нашем примере можно сформулировать следующим образом. Обязательно должно быть задано имя сотрудника, без которого невозможно идентифицировать всю вводимую информацию. Этому требованию соответствует операция необходимости □f1(NA) для соответствующего поля. Остальные поля могут быть либо не указаны, либо указаны в правильном формате. Этому соответствует конъюнкция с операцией возможности для каждого поля: ◊f2(PH) для номера телефона и ◊f3(EM) для электронной почты. При этом желательно, чтобы был задан или номер телефона, или адрес электронной почты, тогда хотя бы по одному каналу связи можно связаться с сотрудником. Это даёт составляющую f2(PH) ⋁

Таблица 4. Примеры заключений

Варианты параметров	**NA**	**PH**	**EM**	**Значение**	**Результирующее высказывание без приведения к ДНФ**
1. Все параметры не определены	null	null	null	false	Не указана фамилия
2. Все параметры ошибочные	И	123	x	false	Ошибка в фамилии ⋀ Неправильный формат номера телефона ⋀ Неправильный формат адреса почты ⋀ (Неправильный формат номера телефона ⋁ Неправильный формат адреса почты)
3. В одном параметре ошибка	Иванов	123	x@x.x	false	Неправильный формат номера телефона
4. Неопределённость результата	Иванов	null	null	undef	Не указан номер телефона ⋁ Не указан адрес почты
5. Без ошибок и неопределённости	Иванов	12345678900	null	true	Фамилия указана ⋀ Номер телефона указан ⋀ Не указан адрес почты ⋀ Номер телефона указан

f3(EM). В результате получаем предикат валидации в следующем виде:

$$p(NA, PH, EM) = \Box f1(NA) \wedge \Diamond f2(PH) \wedge \wedge \Diamond f3(EM) \wedge (f2(PH) \vee f3(EM)). \quad (7)$$

В таблице 4 показаны варианты заключений, которые формируются предикатом p(NA, PH, EM) при различных фактических значениях полей формы. Для прослеживания появления высказываний из фактов в результирующем высказывании, в таблице не выполнено приведение к дизъюнктивной нормальной форме (ДНФ).

Для булевой алгебры известно большое количество методов оптимизации записи выражений с разными критериями оптимальности. Оптимизация символической записи выражений для трёхзначной логики менее исследована, хотя и здесь имеются определённые достижения, например, [20, 21]. Однако на практике при проектировании программных систем избегают зависимости одного шага варианта использования от большого количества фактов. Большое количество фактов с высокой вероятностью будет приводить к тому, что пользователь не сможет правильно воспринять предлагаемое ему заключение и будет совершать ошибки при принятии решения. Поэтому оптимизацию логических выражений в данном случае можно не использовать. В то же время для пользователя основную информацию несёт текст элементарных высказываний, а не логическая структура. Поэтому на практике можно показывать пользователю список элементарных высказываний без указания операций, с которыми они вошли в результирующее высказывание. При этом дубликаты элементарных высказываний следует убрать из списка.

В примере результирующие высказывания в вариантах 2, 4 и 5 таблицы 4 представляют собой достаточно сложные для понимания пользователем конструкции с различными логическими операциями. После удаления знаков логических операций и дубликатов фактов в варианте 2 пользователю будет представлена следующая информация:

- Ошибка в фамилии;
- Неправильный формат номера телефона;
- Неправильный формат адреса почты.

Пользователю будет понятно, что ошибка в прохождении шага варианта использования будет устранена, если изменить перечисленные факты.

В примере 4 из таблицы 4 пользователю предоставляется список из следующих фактов:

- Не указан номер телефона;
- Не указан адрес почты.

Значение заключения undef означает, что система не в состоянии определить, можно ли продолжать выполнение варианта использования. Пользователь должен выбрать один из двух вариантов продолжения работы:

1) Указать системе, что продолжать выполнение варианта использования нельзя, нужно вернуться назад так, чтобы можно было изменить факты: указать номер телефона и/или адрес почты.

2) Указать системе, что можно продолжить работу с имеющимися фактами дальше.

В примере 5 из таблицы 4 список фактов будет таким:

- Фамилия указана;
- Номер телефона указан;
- Не указан адрес почты.

Здесь значение заключения true показывает, что система продолжает выполнение варианта использования без ошибок. Обычно в такой ситуации список фактов не показывают пользователю т.к. от него не требуется вмешиваться в работу программы.

Таким образом, предложенный подход к формализации правил валидации на основе трёхзначной логики и получения итогового высказывания из элементарных высказываний, позволяет упростить разработку программ, требования к которым сформулированы в виде вариантов использования. Для этого авторами создана соответствующая библиотека на языке C#. В ней используется достаточно удобная форма записи предикатов на основе перегрузки операторов [22]. Библиотека внедрена в процесс разработки программ в компании Инрэко ЛАН.

Литература

1. *Lhotka R.* Expert C# Business Objects. Apress, 2004. 840 p.

2. *Эванс Э.* Предметно-ориентированное проектирование (DDD). Структуризация сложных программных систем. Пер. с англ. М.: Вильямс, 2011. 448 с.

3. *Lutowski R.* Software Requirements: Encapsulation, Quality, and Reuse. Auerbach Publications, 2005. 264 p.

4. *Арлоу Дж., Нейштадт А.* UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е издание. СПб.: Символ-Плюс, 2013. 624 с.

5. *Cohn M.* User Stories Applied: For Agile Software Development. Addison Wesley Longman Publishing Co., Inc., USA, 2004. 304 p.

6. *Коберн А.* Современные методы описания требований к системам. М.: Лори, 2002. 263 с.

7. *Rosenberg D., Stephens M.* Use Case Driven Object Modeling with UML: Theory and Practice. Springer, 2007. 438 p.

8. *Ларман К.* Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку. 3-е изд. М.: Вильямс, 2007. 727 с.

9. *Yue T., Briand L.C., Labiche Y.* A systematic review of transformation approaches between user requirements and analysis models // Requirements Engineering. 2011. Vol. 16. Pp. 75–99.

10. *Troya, J., Moreno, N., Bertoa, M.F., Vallecillo, A.* Uncertainty representation in software models: a survey // Software and Systems Modeling, 20(4). 2021. Pp. 1183–1213.

11. *Console, M., Guagliardo, P., Libkin, L.* Propositional and predicate logics of incomplete information // Principles of Knowledge Representation and Reasoning: Proceedings of the 16th International Conference, KR 2018, Pp. 592–601.

12. *Kazimirov, A.S., Panteleyev, V.I., Semicheva, N.L.* Generalized interpretation of variables as a decision-making model in conditions of uncertainty, Proceedings of 2017 20th IEEE International Conference on Soft Computing and Measurements, SCM 2017, 2017. Pp. 512–513.

13. *Epstein, G.* Multiple-valued logic design: An introduction. Boca Raton. Routledge, 2017. 370 p.

14. *Камкин А.С.* Тестирование в условиях неполной информации. Подход к разработке спецификаций и генерации тестов // Труды Института системного программирования РАН. 2006. Том 10. С. 143–166.

15. *Mikheyenkova M.A.* On the Logical Approach to the Rationality of an Intelligent Agent // CEUR Workshop Proceedings. 2021. Vol. 3044. Pp. 40–47.

16. *Cheng, D., Feng, J., Zhao, J., Fu, S.* On adequate sets of multi-valued logic // Journal of the Franklin Institute. 2021. Vol. 358(13). Pp. 6705–6722.

17. *Bessmertny, I., Sukhikh, N., Vedernikov, J., Koroleva, J.* Ternary Logics in Decision Making // Reliability and Statistics in Transportation and Communication. RelStat 2020. Lecture Notes in Networks

and Systems. Vol. 195. Springer, Cham, 2021. Pp. 411–419.

18. *Карпенко А.С.* Развитие многозначной логики. М.: Издательство ЛКИ, 2010. 448 с.

19. *Бессмертный И. А.* О построении интеллектуальных систем в троичной логике // Программирование. 2014. № 1. С. 31–35.

20. *Поспелов Д.А.* Логические методы анализа и синтеза схем. М.: Энергия, 1974. 368 с.

21. *Reps T., Loginov A., Sagiv M.* Semantic minimization of 3-valued propositional formulae // Proceedings 17th Annual IEEE Symposium on Logic in Computer Science. 2002. Pp. 40–51.

22. *Албахари Б., Албахари Дж.* C# 7.0. Справочник. Полное описание языка. М.: Вильямс, 2016. 1023 с.

Поступила 15 августа 2022 г.

English

VALIDATION RULES OF PASSING STEPS FOR USE CASE BASED ON THREE-VALUED LOGIC

Ilya Roidovich Dubov — Grand Dr. in Engineering, Professor, Federal State Budgetary Educational Institution of Higher Professional Education “Vladimir State University named after A.G. and N.G. Stoletovs”.

E-mail: dubov@vlsu.ru

Konstantin Vladimirovich Kulikov — PhD, the Head of Computer Engineering and Control Systems Department, Federal State Budgetary Educational Institution of Higher Professional Education “Vladimir State University named after A.G. and N.G. Stoletovs”.

E-mail: kulikov@vlsu.ru

Address: 602264, Russian Federation, Vladimir, Gorky street. 87.

Abstract: Validation rules are implemented as methods of program classes in the simplest case. Another approach involves using "Specification" template, in which rules are formed as Boolean algebra predicates. The paper proposes to link validation rules with steps of use cases, through which functional requirements for software system are described. Validation can result in one of three situations at the execution step of use case: (1) the step was completed correctly, (2) an error occurred at step execution leading to an alternative path, (3) uncertainty that requires the actant decision to continue execution of the current path or to move towards an alternative path. Therefore, three-value logic is proposed to use for validation, while the values "true", "false" and "uncertainty" correspond to the above situations. The system status at validation moment is described by facts – atomic logical values reflecting status of program elements. A statement is attributed to each such value explaining the meaning in a fashion understandable to the actant in natural language. Combination of value and statement is conclusion. Thus, the purpose of validation is to obtain a final conclusion with respect to the predicate formulated for the step of the use case. Statement of the operation result is formed when performing a two-place logical operation on operands. The statement for the value that defined the operation logical result is included into the result. The statement after applying one-place operation on interium conclusion remains unchanged. "NOT", "AND", "OR" are proposed to additionally be included into the set of operations traditionally used in program development, one-place operations of possibility and necessity known in three-value logic. One-place operations are moved to leaves in parse tree of validation predicate according to known formulas. Recommendation while implementing validation are to provide the final conclusion to the actant without mentioning operations’ names and without duplicate statements. The proposed approach to validation is implemented by the authors as programming library.

Keywords: three-value logic, statement logic, use case, program validation.

References

1. *Lhotka R.* Expert C# Business Objects. Apress, 2004. 840 p.

2. *Evans E.* Domain-oriented design (DDD). Structuring of complex software systems. Transl. from Engl. Moscow: Williams, 2011. 448 p.

3. *Lutowski R.* Software Requirements: Encapsulation, Quality, and Reuse. Auerbach Publications, 2005. 264 p.

4. *Arlow J., Neustadt A.* UML 2 and the Unified Process. Practical Object-oriented Analysis and Design, 2nd edition. St. Petersburg: Simvol-Plyus, 2013. 624 p.

5. *Cohn M.* User Stories Applied: For Agile Software Development. Addison Wesley Longman Publishing Co., Inc., USA, 2004. 304 p.

6. *Coburn A.* Modern methods of describing system requirements. Moscow: Lori, 2002. 263 p.

7. *Rosenberg D., Stephens M.* Use Case Driven Object Modeling with UML: Theory and Practice. Springer, 2007. 438 p.

8. *Larman K.* Application of UML 2.0 and design patterns. Introduction to Object-oriented analysis, design and iterative development. 3rd ed. Moscow: Williams, 2007. 727 p.

9. *Yue T., Briand L.C., Labiche Y.* A systematic review of transformation approaches between user requirements and analysis models. Requirements Engineering. 2011. Vol. 16. Pp. 75–99.

10. *Troya, J., Moreno, N., Bertoa, M.F., Vallecillo, A.* Uncertainty representation in software models: a survey. Software and Systems Modeling, 20(4). 2021. Pp. 1183–1213.

11. *Console, M., Guagliardo, P., Libkin, L.* Propositional and predicate logics of incomplete information. Principles of Knowledge Representation and Reasoning: Proceedings of the 16th International Conference, KR 2018, Pp. 592–601.

12. *Kazimirov, A.S., Panteleyev, V.I., Semicheva, N.L.* Generalized interpretation of variables as a decision-making model in conditions of uncertainty, Proceedings of 2017 20th IEEE International Conference on Soft Computing and Measurements, SCM 2017, 2017. Pp. 512–513.

13. *Epstein, G.* Multiple-valued logic design: An introduction. Boca Raton. Routledge, 2017. 370 p.

14. *Kamkin A.S.* Testing in conditions of incomplete information. Approach to the development of specifications and test generation. Proceedings of the Institute for System Programming of the RAS. 2006. Vol. 10. Pp. 143–166.

15. *Mikheyenkova M.A.* On the Logical Approach to the Rationality of an Intelligent Agent. CEUR Workshop Proceedings. 2021. Vol. 3044. Pp. 40–47.

16. *Cheng, D., Feng, J., Zhao, J., Fu, S.* On adequate sets of multi-valued logic. Journal of the Franklin Institute. 2021. Vol. 358(13). Pp. 6705–6722.

17. *Bessmertny, I., Sukhikh, N., Vedernikov, J., Koroleva, J.* Ternary Logics in Decision Making // Reliability and Statistics in Transportation and Communication. RelStat 2020. Lecture Notes in Networks and Systems. Vol. 195. Springer, Cham, 2021. Pp. 411–419.

18. *Karpenko A.S.* Development of multivalued logic. Moscow: Izdadel'stvo LKI, 2010. 448 p.

19. *Bessmertny I. A.* On the construction of intelligent systems in ternary logic. Programming and Computer Software. 2014. No. 1. Pp. 31–35.

20. *Pospelov D.A.* Logical methods of analysis and synthesis of circuits. Moscow: Energiya, 1974. 368 p.

21. *Reps T., Loginov A., Sagiv M.* Semantic minimization of 3-valued propositional formulae. Proceedings 17th Annual IEEE Symposium on Logic in Computer Science. 2002. Pp. 40–51.

22. *Albahari B., Albahari J.* C# 7.0. Handbook. Full description of the language. Moscow: Williams, 2016. 1023 p.