УДК 681.12

**Д.Л. ПАРНАС, С.А. ВИЛКОМИР**

***Университет г. Лимерик, Ирландия***

# ИСПОЛЬЗОВАНИЕ ТОЧНОЙ ДОКУМЕНТАЦИИ ДЛЯ ПОВЫШЕНИЯ УРОВНЯ ДОВЕРИЯ К КРИТИЧЕСКОМУ ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ

Часто возникают сомнения, можно ли доверять программному обеспечению в критических приложениях. Обсуждаются причины этого недоверия и предлагаются пути улучшения документации для повышения уровня доверия к программному обеспечению. Показано, как табличные выражения могут быть использованы для создания точной программной документации, облегчающей разработку, инспекции и тестирование программного обеспечения.

**критическое программное обеспечение, документация, уровень доверия, тестирование**

## 1. Программное обеспечение: чрезвычайно полезная технология, которой мы не доверяем

В данной секции рассматриваются преимущества и недостатки технологии, использующей программное обеспечение (ПО).

**Почему ПО является чрезвычайно полезной технологией?** Не нужно слишком долго искать изделия, которые не будут практичными без использования ПО. Начиная от телефонов, фото и видеокамер, автомагнитол и заканчивая электростанциями – всюду использование программируемых процессоров дает значительные преимущества. Программируемые технические средства являются массовой продукцией. Как результат, процессоры значительно дешевле в производстве, чем специализированные технические средства, которые процессоры заменяют. Программируемые процессоры могут быть специально приспособлены для заказчика, чтобы выполнять именно те функции, которые нужны для конкретного приложения. Программируемость также означает, что изделия могут быть улучшены без изменения используемого технического обеспечения (ТО). Часто ПО включает программы, которые загружают и инсталлируют свои новые версии. Более того, для таких приложений как системы управления электростанциями, мы можем реализовывать усовершенствованные режимы работы, т.е. вводить более детальные различия в режимах работы, чем для функций, которые реализуются на аппаратном уровне. ПО также позволяет собирать и отображать больше оперативной информации, чем это было возможно для прежних технологий.

**Почему мы не доверяем ПО?** Не приходится слишком долго искать ответ, почему люди настороженно относятся к ПО. Не так давно, эксперты в критическом программном обеспечении говорили, что они не стали бы летать самолетами, которые управляются ПО. Сегодня публикуются статьи других экспертов, которые считают, что ПО нельзя доверять даже такую простую работу как подсчет голосов при голосовании. Причины отсутствия доверия к ПО многочисленны и сложны; мы перечислим только некоторые из них.

**А.** *История отказов ПО в прошлом.* ПО заслужило наше недоверие [1, 11, 13, 14]. Отказы ПО происходили и происходят достаточно часто, и поэтому ПО должно часто модернизироваться. Даже разработчики ПО бывают иногда удивлены поведению их программ в неожиданных обстоятельствах.

**Б.** *Трудности понимания и инспектирования ПО*. Опыт десятилетий показывает, что людям трудно понимать длинные программы. Пытаясь понять и проверить корректность длинной программы, мы должны декомпозировать ее на маленькие части и предварительно связать каждую такую часть с вы-

полняемой этой частью функцией. Мы должны убедить себя в том, что (1) если каждая часть выполняет свою функцию, вся программа будет работать корректно, (2) каждая часть выполняет свою функцию.

Мы часто обнаруживаем, что наши предварительные допущения не совсем точно отражают намерения программиста или работу программы. Тогда, после пересмотра нашего начального разделения программы на части и функций этих частей, мы должны повторить весь процесс. В принципе, такой итеративный процесс должен подойти к концу и показать, является ли данная программа корректной или нет. На практике мы обычно отказываемся от продолжения до того, как приходим к полному пониманию о работе программы. Процесс обрывается, когда истекает отпущенное время или истощается наше терпение. Данный инспекционный процесс является ненадежным также и потому, что он зависит от неточностей в описании функций компонентов, сделанных допущений и памяти инспектора.

Люди не доверяют изделиям, конструкция которых настолько сложна, что никто ее не может понять. Пока мы не улучшим наши возможности в понимании и инспектировании ПО, мы не сможем ему доверять.

**В.** *Трудности тестирования ПО*. Одним из самых цитируемых высказываний пионера программирования Эдсгера Дейкстра является «Тестирование программ может служить для доказательства наличия ошибок, но никогда не докажет их отсутствия» [2]. Это замечание базируется на факте, что число возможных тестов даже для простых программ является слишком большим и всегда возможно найти программу, которая удовлетворяла всем предыдущим тестам, но откажет при каждом следующем. Следовательно, в большинстве случаев мы не знаем, достаточно ли проведенного тестирования или нет. По контрасту со старыми технологиями, интерполяция не работает должным образом для дискретных функций, которые описывают поведение ПО. Как результат, пользователи находят ошибки в ПО даже после интенсивного тестирования. Никто не может быть полностью уверен, что все ошибки были найдены, или когда проявится следующая ошибка.

Люди не будут доверять ПО, пока они не убедятся, что оно было должным образом протестировано.

**Г.** *Отсутствие взаимного согласия о желаемых функциях ПО*. Большинство людей находят очень трудным сформулировать, какое поведение они ожидают от компьютерной системы. Часто ответ на просьбу сформулировать требования к системе будет примерно таким: «Я буду знать, когда я увижу систему». И после того, как система готова, мы услышим: «Это не то, что нужно».

Люди не доверяют ПО, если при его использовании они продолжают обнаруживать новые требования к ПО.

**Почему, в отличие от ПО, мы доверяем самолетам?** Хотя мы полностью понимаем, что наши жизни зависят от результатов работы большой группы инженеров, большинство из нас входит в самолет без больших волнений. Мы знаем, что данное изделие является очень сложным и что отказ даже небольшой его части может быть опасным, но мы доверяем этому изделию. Можно сформулировать некоторые причины нашего доверия:

– хорошая история результатов эксплуатации механических и электрических систем;

– хотя данные системы являются сложными, они имеют понятную структуру, которая позволяет выделить части системы с четкими спецификациями и проинспектировать каждую часть в отдельности;

– тестирование механических и электрических систем хорошо разработано и основано на математических и физических методах и законах. Свойства, которые должны быть подтверждены, четко определены и используемые для этого методы понятны;

– нет никаких сомнений в том, какие функции ожидаются от самолета и от каждой из его компонент. Данные функции тщательно задокументированы, проверены и используются как основа для проверок результатов проектирования самолета.

Если мы хотим увеличить уровень доверия к ПО, мы должны улучшить нашу способность инспектировать, тестировать и специфицировать программные системы и их компоненты.

## 2. Как хорошая документация может сделать ПО более заслуживающим доверия?

Если мы рассмотрим процесс, используемый для обеспечения уверенности в механических и электрических инженерных изделиях, мы увидим, что документирование проекта, функций и компонент этих изделий играет существенную роль в этом процессе. Профессиональная инженерная документация, используемая в этом процессе, является точной, абстрактной и используется более для ссылок во время работы, чем для первоначального описания изделия.

Подобная документация может использоваться и для ПО. Хорошая программная документация помогает более эффективно инспектировать и тестировать ПО. Точные документы позволяют инженерам подтвердить, что, если каждый программный компонент удовлетворяет установленным спецификациям, весь продукт в целом можно считать удовлетворительным. В свою очередь то, что каждый компонент удовлетворяет спецификациям, мы можем подтвердить, комбинируя тестирование и математические методы. В данной секции мы обсудим требования к подобной документации.

**Документация должна быть точной.** Точные документы не позволяют различных интерпретаций. Такие утверждения как «Система должна вызвать останов, если температура слишком долго превышает лимит» очевидно является неточным. Однако, хотя утверждение «Система должна вызвать останов, если средняя температура в течение более 3 секунд превышает 89 градусов» намного более точное, оно не определяет, что понимается под «средним» значением – среднее арифметическое, среднеквадратичное или медианное значение. Обычно точность (в инженерном понимании) требует использования математических выражений.

**Документация должна быть абстрактной.** Мы рассматриваем документацию как абстрактную, если она содержит только поддающуюся наблюдению информацию и абстрагируется (т.е. не упоминает) от всего, что пользователь или пользовательская программа наблюдать не могут.[1] Абстрактная документация для устройств с памятью должна иметь возможность ссылаться как на текущие значения входных и выходных переменных, так и на их значения в прошлом.

**Документация должна носить справочный характер**. Справочные документы не являются вводной или обзорной документацией, они подразумевают общее знакомство пользователей с изделием и его окружением[2]. Отдельная документация должна быть доступна для первоначального ознакомления с общей природой и структурой изделия. Синтаксическая информация и неформальные комментарии, идущие вместе с ПО, также не заменяют собой справочную документацию.

Справочные документы должны создаваться как удобные для поиска информации репозитарии и обеспечивать легкий доступ к детальным данным о поведении программы. Использование справочной документации должно позволять быстро находить ответы на такие вопросы, как:

Как данный компонент реагирует на входную последовательность . . . ?

При каких обстоятельствах выходное значение будет равным . . . ?

Как данный компонент выполняет . . . ?

Справочная документация должна выпускаться в соответствии со строгими правилами и описывать каждый факт в определенном и только в одном месте. Такие документы предназначены не для чтения от начала и до конца, но для легкого, быстрого и точного ответа на специальные конкретные вопросы.

Как профессиональная документация повышает уровень доверия к ПО? Секрет того, как сделать любой продукт заслуживающим доверия, заключается в древнем изречении «разделяй и властвуй». Мы долж-

---

[1] Это специальный случай более общего определения абстрактности как представления одновременно многих различных вещей. Абстрактная документация представляет все возможные реализации, которые имеют описываемое в документации поведение.

[2] В компьютерных науках свойственно смешивать справочную документацию с вводными материалами.

ны разделять программный продукт на множество небольших компонентов таким образом, что из нашей уверенности в каждом компоненте вытекает доверие ко всему продукту в целом. Для комплексного ПО это означает, что каждый компонент должен быть точно специфицирован, и что корректность компонента должна быть верифицирована на основе только его спецификации и спецификаций использующих его компонентов. Именно такие спецификации обсуждаются в данной статье. Ясно, что аккуратность, точность и полнота документов являются необходимыми условиями. Абстракность документов значительно облегчит весь процесс, поскольку объем рассматриваемой информации при этом значительно уменьшается.

## 3. Как определить содержание основных документов ПО?

Многих удивляет, что содержание программной документации может быть определено математически. Каждый документ имеет свое индивидуальное назначение и целевую группу пользователей, но все документы могут быть представлены с помощью отношений (множеств упорядоченных пар). Большинство этих отношений являются функциями. Рассмотрим, например, следующие необходимые документы:

Документ **«Системные требования»** должен представлять множество отношений между временными функциями, определяюшими историю функционирования системы, и значениями ее выходных параметров. Область определения отношения должна включать все возможные последовательности входных и выходных значений [6].

Документ **«Интерфейс компонента ПО»** должен представлять отношение между описателями событий /event descriptions/ (трассами /traces/) и текущими значениями выходных параметров компонента.

Документ **«Функция программы»** (для программ, завершающихся после выполнения) должен представлять LD-отношение между начальными и конечными состояниями программы. Область определения отношения должна включать все начальные состояния программы, для которых завершение ее работы возможно. Множество компетентности /competence set/ LD-отношения включает все начальные состояния программы, для которых завершение ее работы гарантировано [16].

Документ **«Внутренний проект компонента ПО»** должен представлять абстрагированное отношение и программное отношение для основных проргамм компонента. Область определения абстрагированного отношения должна включать все достижимые внутренние состояния компонента. Область значений отношения является множеством трасс, включающем все возможные варианты функционирования компонента.

Документ рассматривается как представляющий точное описание, если каждая упорядоченная пара, которая может возникнуть, включена в отношение, и все пары, включенные в отношение, могут возникнуть в реальности. Подробнее данный вопрос рассмотрен в [18].

## 4. Как создавать четкие описания кусочно-непрерывных отношений?

Приведенные выше описания содержания документов являются простыми, но недостаточно конкретными. Для конкретизации нам необходимо иметь возможность создавать легкочитаемые представления для используемых в данных документах отношений. При использовании традиционной математической нотации результат часто получается неудобочитаемым. Например, на рис. 1 представлено четкое описание поведения программного компонента для тестирования клавиатур компьютеров, используемого компанией Делл в Лимерике, Ирландия. Значения отношений и предикатов в данном выражении описаны в [3], но для данного рассмотрения эти значения не являются важными.

Ясно, что хотя данное выражение является полным, точным и абстрактным, для справочной документации оно не является полезным. Проверять корректность такого выражения крайне сложно, даже простой подсчет скобок затруднителен.

```
(N(T)=2∧keyOK∧(¬(T=_)∧N(p(T))=1)) ∨
(N(T)=1∧(T=_∨(¬(T=_)∧N(p(T))=1)) ∧
(¬keyOK∧¬prevkeyOK∧¬prevkeyesc))
∨((¬(T=_)∧N(p(T))=1) ∧((¬keyOK∧keyesc∧
¬prevkeyesc) ∨(¬keyOK∧keyesc∧
prevkeyesc∧prevexpkeyesc)) ∨
((N(T)=N(p(T))+1) ∧(¬(T=_)∧
(1<N(p(T))<L)) ∧(keyOK)) ∨
((N(T)=N(p(T))-1)) ∧(¬keyOK∧¬keyesc∧
(¬prevkeyOK∧prevkeyesc∧preprevkeyOK)∨
prevkeyOK) ∧((¬(T=_)∧(1<N(p(T))<L)) ∨
(¬(T=_)∧N(p(T))=L))) ∨((N(T)=N(p(T))) ∧
(¬(T=_)∧(1<N(p(T))≤L))∧((¬keyOK∧
¬keyesc∧(¬prevkeyOK∧prevkeyesc∧
¬preprevkeyOK)) ∨ (¬keyOK ∧¬prevkeyOK
∧ ¬prevkeyesc) ∨(¬keyOK∧keyesc∧
¬prevkeyesc) ∨ ¬keyOK∧keyesc∧
prevkeyesc∧prevexpkeyesc)) ∨
((N(P(T)=Fail) ∧(¬keyOK∧keyesc ∧
prevkeyesc∧¬prevexpkeyesc) ∧
(1≤N(p(T))≤L)) ∨ ((N(P(T)=Pass) ∧(¬(T= _)∧
N(p(T))=L) ∧(keyOK))
```

Рис. 1. Пример логического описания поведения ПО

Полная семантическая проверка крайне трудна и потребует больших затрат времени от разработчиков ПО. Можно пытаться улучшить формат и построение такого выражения, однако, как показывает наш опыт, разработчики ПО не проявляют желания читать и писать документацию в такой форме и считают такой подход не стоящим усилий.

Тем не менее, первый автор данной статьи, начиная с 1977 года ([7]) использовал новую форму представления выражений, а именно табличные выражения. Табл. 1 представляет табличное выражение, которое эквивалентно традиционному выражению на рис.1.

Табличные выражения позволяют проводить их структурный анализ, т.е. проверять их полноту, непротиворечивость и корректность. Для того, чтобы проверить полноту и корректность, необходимо убедиться, что условия в заголовках таблицы являются

Таблица 1

Табличное выражение

| **N(T)=** | | | **T = _** | **¬(T = _ ) ∧** | | |
|---|---|---|---|---|---|---|
| | | | | **N(p(T))=1** | **1<N(p(T))<L** | **N(p(T))=L** |
| keyOK | | | | 2 | N(p(T))+1 | Pass |
| ¬keyOK∧ | ¬keyesc∧ | (¬prevkeyOK ∧ prevkeyesc ∧ preprevkeyOK) ∨ prevkeyOK | | | N(p(T))-1 | N(p(T))-1 |
| | | ¬prevkeyOK ∧ prevkeyesc ∧ ¬preprevkeyOK | | | N(p(T)) | N(p(T)) |
| | | ¬prevkeyOK ∧ ¬prevkeyesc | 1 | 1 | N(p(T)) | N(p(T)) |
| | keyesc∧ | ¬prevkeyesc | | 1 | N(p(T)) | N(p(T)) |
| | | prevkeyesc ∧ ¬prevexpkeyesc | | Fail | Fail | Fail |
| | | prevkeyesc ∧ prevexpkeyesc | | 1 | N(p(T)) | N(p(T)) |

взаимно исключающими и покрывают все возможные случаи. Затем проверяется коррекность каждой ситуации (каждой ячейки таблицы) по отдельности. Наш опыт работы с подобными табличными выражениями показывает, что практические работники предпочитают такие выражения длинным текстовым описаниям, так как они могут легко и быстро найти необходимую информацию. Наш опыт также свидетельствует, что при использовании табличных выражений появление ошибок менее вероятно.

Когда табличные выражения были впервые предложены, некоторые называли их «полуфор-

мальными». На самом деле эти выражения полностью формальны. Их смысл определялся различными способами, мы определяем табличные выражения как эквивалент традиционных выражений. Табл. 1 представляет только один из возможных типов табличных выражений, которые могут применяться на практике. Одно и то же отношение может быть описано различными способами, и разработчики могут выбрать наиболее подходящий для их нужд способ[3].

## 5. Пример точной документации - TFM спецификации

Как пример точной документации, мы рассмотрим Trace Function метод (TFM) для описания и специфицирования компонентов (модулей) ПО. TFM был успешно применен в двух индустриальных проектах [3, 21] и также использовался Лабораторией качества программного обеспечения (SQRL) Университета г. Лимерик в нескольких внутренних проектах.

TFM спецификации определяют значения каждого выходного параметра компонента ПО как функцию от всех прошлых событий, которые воздействуют на данный компонент. Событием является момент времени, когда компонент читает или изменяет значения глобальных переменных. Каждое событие описывается с помощью «дескриптора события», который содержит имена и значения глобальных переменных до и после события. Мы будем называть последовательность дескрипторов событий «трассой» (trace).

Для представления указанных функций используются табличные выражения. Даные таблицы могут создаваться систематически шаг за шагом, определяя на каждом шаге условия, которые разделяют трассы на взаимно исключающие и дополняющие друг друга подмножества.

[3] Наилучший способ для тех, кто составляет таблицы, может не быть лучшим для тех, кто их использует как справочную документацию. Для перехода от одной формы таблиц к другой были разработаны специальные алгоритмы [23].

Как пример, илюстрирующий применение TFM, рассмотрим спецификации для программируемого электронного календаря. Даный календарь показывает текущую дату (день и месяц) и имеет две кнопки «set» и «up» для установки начальной даты (рис. 2):

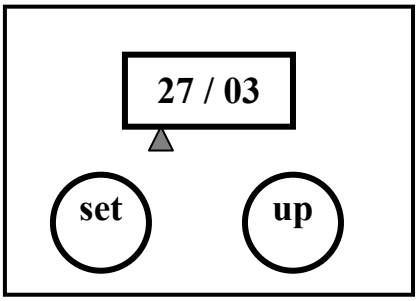


Рис. 2. Электронный календарь

Кнопка «set» определяет режим (т.е. что будет устанавливаться) и переключается между «день» и «месяц». Маленький треугольник на рисунке показывает текущий режим. Нажатие на кнопку «up» увеличивает выбранную величину на единицу до тех пор, пока «день» не станет равным 31 или «месяц» равным 12, и затем начинает цикл заново с 1. Начальной датой является 1/1.

Для данного примера, трасса описывает последовательность нажатий на кнопки, например,

*T1=set.up.up.up.up* or *T2=set.up.set.set.up.set.up.up.up*.

Для каждой трассы *T*, выходная функция возвращает вектор *(S(T), D(T), M(T))*, где $S(T) \in$ *{«день», «месяц»}* является текущим режимом, $D(T) \in$ *{1, 2. ... 31}* – показываемый день и $M(T) \in$ *{1, 2, ... 12}* – показываемый месяц. Чтобы отразить историю работы (предыдущие нажатия), мы используем две стандартных функции: *r(T)* – самый последний дескриптор события в трассе *T*; *p(T)* – трасса *T* с удаленным последним дескриптором события. Например, если *T=set.up.set*, то *r(T)=set* и *p(T)= set.up*. Мы можем описать требуемое поведение электронного календаря, используя только одно табличное выражение (табл. 2).

Для реальных больших программных систем TFM таблицы более сложны, часто необходимо использовать дополнительные таблицы для вспомогательных функций.

Таблица 2

TFM спецификации для электронного календаря

(S(T), D(T), M(T)) ≡

| | | | | |
|---|---|---|---|---|
| T =_ | | | | (day, 1, 1) |
| ¬ (T =_) ∧ | r(T)=set ∧ | S(p(T))=day | | (month, D(p(T)), M(p(T))) |
| | | S(p(T))=month | | (day, D(p(T)), M(p(T))) |
| | r(T)=up ∧ | S(p(T))=day ∧ | D(p(T))=31 | (S(p(T)), 1, M(p(T))) |
| | | | ¬ (D(p(T))=31) | (S(p(T)), D(p(T))+1, M(p(T))) |
| | | S(p(T))=month ∧ | M(p(T))=12 | (S(p(T)), D(p(T)), 1) |
| | | | ¬ (M(p(T))=12) | (S(p(T)), D(p(T)), M(p(T))+1) |

Однако даже для сложных программ TFM спецификации имеют на удивление небольшой размер. Это объясняется абстрактным характером данной документации; спецификации не содержат деталей реализации системы.

## 6. Как использовать документацию для проверки результатов проектирования ПО?

Одним из путей повышения уровня доверия к механическим и электрическим системам является строгая экспертиза проекта перед его реализацией. Используя математический аппарат, такой как дифференциальные уравнения, можно оценить численные значения потребляемой мощности, выработки теплоты, коэффициента искажений и других величин на основе принципиальной схемы и других проектных документов. При проектировании ПО обычно невозможно провести детальный анализ до разработки кода, так как документация, если и выпускается, то не является достаточно точной

Точная математическая документация, обсуждаемая в данной статье, способна изменить ситуацию. Теоретические исследования в компьютерных науках, особенно исследования в области отношений [22], позволяют составить уравнения, которые справедливы, если проект ПО корректен. Однако, без документов по внутреннему проекту и спецификациям интерфейсов модуля, данные возможности остаются лишь теоретическими. Имея-же подобные документы, возможно проверить проект до перехода к его реализации. Это предохранит от значительных потерь, возникающих при реализации ошибочного проекта.

## 7. Как использовать документацию для инспектирования ПО?

Хотя полезность проверенной проектной документации очевидна, однако для доверия программному продукту, проверенным на корректность должен быть прежде всего програмный код. Вполне возможно иметь очень хороший проект, но дефектную реализацию. Инспектирование кода для устранения часто встречающихся мелких ошибок является трудной задачей. Даже когда инспекция успешна и многие ошибки обнаружены, часть ошибок может остаться ненайденной. Для ПО характерно множество тонких взаимодействий между компонентами и для нахождения ошибок необходимо держать в голове множество деталей.

Предыдуший опыт ([15, 17]) показал, что точная документация позволяет применять философию «разделяй и властвуй» для инспекций ПО. Для этого ПО разлагается на компоненты и каждый компонент описывается множеством «представлений» (displays). Используя описание компонентов с помощью представлений и спецификаций интерфейсов, мы можем инспектировать представления по отдельности, зная, что если каждое представление корректно, то и вся система будет корректной.

Данный подход более детально обсуждается в [15].

## 8. Как использовать точную документацию для тестирования ПО?

Тестирование является одним из главных инструментов для повышения доверия к программным продуктам. Однако тестировщики ПО встречаются с существенными проблемами:

– из-за отсутствия точной документации подготовка тестовых примеров не начинается до того, как будет готов программный код, что приводит к работе в спешке и под давлением;

– критерии тестового покрытия в основном зависят от кода ПО и поэтому применяются обычно только при «white box» тестировании;

– по окончанию тестирования возможны разногласия в оценке результатов, т.е. считать поведение ПО корректным или нет.

Точная документация помогает смягчить данные проблемы, так как:

– подготовка тестовых примеров может начаться сразу же, когда проектные документы будут разработаны и одобрены;

– точная документация полностью определяет какае результаты тестирования являются приемлемыми и не оставляет места для споров о корректности продукта;

– точная документация предоставляет основу для измерения покрытия при «black box» тестировании, например, мы должны убедиться, что каждая ячейка табличного выражения протестирована в достаточной мере;

– мы можем использовать точную документацию для тестирования ситуаций, в которых, судя по прошлому опыту, ошибки наиболее вероятны, например, если мы знаем, что некоторая функция обращается в ноль или имеет особые точки, мы должны протестировать эти «места интереса» [5]; другими «местами интереса» являются границы между различными строками и столбцами в табличных спецификациях [10];

– точная документация является полезной также для статистического тестирования. Целью статистического тестирования является оценка надежности ПО. Для оценки надежности необходимо знать операционный профиль, т.е. вероятностные характеристики ожидаемого использования ПО [4, 12]. Данный профиль используется для тестирования каждой ситуации с частотой, пропорциональной вероятности появления данной ситуации при использовании ПО. Точная документация может использоваться как основа для описания операционного профиля, задавая вероятности появления ситуаций, определенных каждой ячейкой в табличном выражении.

Использование точной документации для улучшения тестирования рассматривалось также в [19, 20, 25].

## 9. Как опыт подтверждает практичность предлагаемого подхода?

Идеи, обсуждаемые в данной статье, были порождены практическими проблемами в индустрии и получили дальнейшее развитие на солидной математической основе в ходе академических исследований. Впервые табличные выражения были использованы для документирования требований к ПО военных систем реального времени [7]. Военно-морская Исследовательская Лаборатория США, где эти идеи впервые возникли, разработала программные утилиты и продолжает работать с подрядчиками над практическими проблемами. Данные идеи были затем применены для систем телефонии в Лаборатории Белла [8], где они были переняты и для других проектов. Методы инспекций ПО, описанные в [15] и [17], в последующем применении доказали свою высокую эффективность – пропущенных ошибок не было обнаружено за 15 лет использования [27].

Недавние совместные работы с компаниями Делл [3, 25] и Эриксон [21] показали эффективность новых форм документирования. Мы обычно получаем смешанные отклики от наших индустриальных партнеров. Инженерам нравится краткость и точность документов, но они чувствуют, что у них нет достаточного времени для подготовки таких доку-

ментов. Менеджерам нравится абстрактная идея выпуска точной проектной документации, однако обычно у них нет желания делать предварительные инвестиции, необходимые для выпуска такой документации. Выпуск подобных документов требует затрат времени для ответов на вопросы, на которые обычно отвечают только на этапе кодирования. Некоторым это время кажется потраченным непродуктивно. Наш опыт показывает, что если подобные инвестиции сделаны, то этап кодирования укорачивается и время на тестирование уменьшается. Однако, если менеджеры заранее считают, что документирование и применение формальных методов на ранних этапах разработки не имеет большой ценности, то непросто убедить их в будущей окупаемости затраченных усилий.

## 10. Как улучшить стандарты для критического ПО?

Мы убеждены, что предлагаемые методы должны в первую очередь должны применяться к критическому ПО, либо важному для безопасности [24, 28], либо критическому для коммерческих целей. Наш опыт показывает, что в области безопасности в качестве первого шага должны быть введены более требовательные стандарты для документирования, тестирования и инспекций. Регулирующие требования для критического с точки зрения безопасности ПО также должны быть точными, формальными и стандартизированными [26]. Поскольку большинство существующих стандартов не требует наличия точной абстрактной документации, они фактически являются «беззубыми». Математические методы могут существенно повысить эффективность таких стандартов.

## Литература

1. Аджиев В. Мифы о безопасном ПО: уроки знаменитых катастроф // Открытые системы. – № 6. – 1998.

2. Дейкстра Э. Заметки по структурному программированию // У. Дал, Э. Дейкстра, К. Хоор. Структурное программирование. – М.: Мир, 1975. – С. 7-97.

3. Baber R., Parnas D.L., Vilkomir S.A., Harrison P., O'Connor T. Disciplined Methods of Software Specification: A Case Study // Proceedings of the International Conference on Information Technology Coding and Computing (ITCC 2005), April 4-6, 2005, Las Vegas, NV, USA, IEEE Computer Society. – P. 428-437.

4. Brown J.R., Lipow M. Testing for software reliability // Proceedings of the International Conference on Reliable Software, Los Angeles, California, 1975. – P. 518-527.

5. Clermont M., Parnas D.L. Using information about functions in selecting test cases // ICSE 2005 Workshop on Advances in Model-Based Software Testing (A-MOST), St. Louis, Missouri, USA, May 15-16, 2005, IEEE Computer Society.

6. Heninger K.L. Specifying Software Requirements for Complex Systems: New Techniques and their Application // IEEE Transactions Software Engineering, Vol. SE-6, January 1980. – P. 2-13.

7. Heninger K.L., Kallander J., Parnas D.L., Shore J.E. Software Requirements for the A-7E Aircraft, NRL Memorandum Report 3876, United States Naval Research Laboratory, Washington D.C., November 1978, 523 pp. and subsequent versions published by the U.S. Naval Research Laboratory.

8. Hester S.D., Parnas,D.L., Utter D.F. Using Documentation as a Software Design Medium // Bell System Technical Journal. –October 1981. – 60, 8. – P. 1941-1977.

9. Hoffman D.M., Weiss D.M. (eds.) Software Fundamentals: Collected Papers by David L. Parnas. – Addison-Wesley, 2001. – 664 p. – ISBN 0-201-70369-6.

10. Jeng B., Weyuker E. A simplified domain-testing strategy // ACM Transactions on Software Engineering and Methodology (TOSEM). – July 1994. – V. 3, n.3. – P. 254-270.

11. Leveson N., Turner C. An Investigation of the Therac-25 Accidents // IEEE Computer. – July 1993. – Vol. 26, No. 7. – P. 18-41.

12. Musa J.D. Operational profiles in software-reliability engineering // IEEE Software. – March 1993. – Vol. 10 Issue 2. – P. 14-32.

13. Neumann P.G. Computer-Related Risks, ACM Press / Addison Wesley, 1995, ISBN 0-201-55805-X. – 384 p.

14. Nuseibeh, B. Ariane 5: Who Dunnit? // IEEE Software. – 1997. – Vol. 14, No. 3. – P. 15-16.

15. Parnas D.L. Inspection of Safety Critical Software using Function Tables // Proceedings of IFIP World Congress 1994. – August 1994. – Vol. III. – P. 270-277 (также Chapter 19 in [9]).

16. Parnas D.L. Precise Description and Specification of Software // Mathematics of Dependable Systems II, edited by V. Stavridou, Clarendon Press. – 1997. – P. 1-14 (также Chapter 5 in [9]).

17. Parnas D.L., Asmis G.J.K., Madey J. Assessment of Safety-Critical Software in Nuclear Power Plants // Nuclear Safety. – April-June 1991. – Vol. 32, no. 2. – P. 189-198.

18. Parnas D.L., Madey J. Functional Documentation for Computer Systems Engineering, Science of Computer Programming (Elsevier) vol. 25, number 1, October 1995, pp 41-61. Также в Lecture Notes in Computer Science (75), Information Systems Methodology, Proceedings ICS, Venice, 1978, Springer Verlag, pp. 211-236. Также в Software Specification Techniques edited by N. Gehani & A.D. McGettrick, AT&T Bell Telephone Laboratories, 1985. – P. 111-130 (QA 76.6 S6437).

19. Peters D., Parnas D.L. Using Test Oracles Generated from Program Documentation // IEEE Transactions on Software Engineering. – March 1998. – Vol. 24, No.3. – P. 161-173.

20. Peters D., Parnas D.L. Requirements-based monitors for real-time systems // IEEE Transactions on Software Engineering. – Feb. 2002. – Vol. 28, Issue 2. – P. 146-158.

21. Quinn C., Vilkomir S.A., Parnas D.L., Kostic S. Specification of Software Component Requirements Using the Trace Function Method // Proceeding of the International Conference on Software Engineering Advances (ICSEA 2006), October 29 - November 1, 2006, Tahiti, French Polynesia.

22. Schmidt G., Ströhlein T. Relations and Graphs-Discrete Mathematics for Computer Scientists. – EATCS Monographs on Theoretical Computer Science. Springer, 1993. – 301 p.

23. Shen H., Zucker J.I., Parnas D.L. Table Transformation Tools: Why and How // Proceedings of the Eleventh Annual Conference on Computer Assurance (COMPASS ‘96), IEEE. – June 1996. – P. 3-11.

24. Storey N. Safety-Critical Computer Systems. – Addison-Wesley, 1996, ISBN 0-201-42787-7.

25. Vilkomir S.A, Tips P., Parna, D.L., Monaha, J., O'Connor T. Evaluation of Automated Testing Coverage: a Case Study of Wireless Secure Connection Software Testing // Supplementary Proceedings of the 16th IEEE International Symposium on Software Engineering Reliability (ISSRE 2005), November 8-11, 2005, Chicago, Illinois, USA. – P. 3.123-3.134.

26. Vilkomir S.A., Bowen J.P., Ghose A. Formalization and assessment of regulatory requirements for safety-critical software // Innovations in Systems and Software Engineering - A NASA Journal. – December 2006. – Vol. 2, Num. 3-4. – P. 165-178.

27. Wassyng A. Private Communication.

28. Wichmann B.A. Software in Safety Related Systems, NPL, Wiley, 1992, ISBN 0471-93474-7.

*Поступила в редакцию 22.02.2007*

**Рецензент:** д-р техн. наук, проф. В.С. Харченко, Национальный аэрокосмический университет им. Н.Е. Жуковского «ХАИ», Харьков.