УДК 004.7

ОСОБЕННОСТИ ПРИМЕНЕНИЯ БАЙЕСОВСКИХ ГРАФИЧЕСКИХ МОДЕЛЕЙ ДЛЯ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

М. М. Зозуля, аспирант, Воронежский государственный технический университет, Воронеж, Россия; orcid.org/0000-0003-3929-7254, e-mail: csit@bk.ru

Описывается подход к проблеме тестирования программного обеспечения. Подход основан на байесовских графических моделях и представляет формальные механизмы для логического структурирования проблемы тестирования программного обеспечения, вероятностной и статистической обработки неопределенностей, которые необходимо устранить, процесса разработки и анализа тестов, а также включения и применения результатов тестирования. ***Цель работы*** *– разработка механизма динамического представления задач тестирования программного обеспечения на основе построения моделей. Предложенные подходы могут быть использованы для разработки тестов, ответов на вопросы «что, если» и оказания поддержки в принятии решений менеджерам и тестировщикам. Модели отражают знания тестировщика программного обеспечения для дальнейшего использования.*

Ключевые слова: *байесовские графические модели, экспертные оценки, сбор знаний, надежность программного обеспечения, тестирование программного обеспечения, статистические методы, разработка тестов.*

DOI: 10.21667/1995-4565-2022-79-68-80

Введение

В классической работе [1] обсуждается ряд недостатков доступных подходов к тестированию программного обеспечения. В данной работе автор заявляет, что «тестирование может доказать несовершенство, обнаружив одну ошибку, но оно не может доказать совершенство» и «учитывая, что мы никогда не сможем доказать совершенство, мы хотим избежать тестирования за пределами значительно уменьшенной отдачи – при этом все еще достигая желаемого уровня кон-доверие». Автор подчеркивает, что тестирование – это деятельность по управлению рисками, когда «нам нужно тщательно выбирать тестовые примеры, чтобы обеспечить необходимый охват, избегая при этом репликации». Однако нет однозначного ответа на вопрос о том, каков минимальный уровень тестирования, необходимый для обеспечения желаемого уровня уверенности. На самом деле обычно нет знания о том, какой уровень уверенности желателен. Действительно, этот вопрос почти никогда не решается. Подход на основе байесовских графических моделей (BGM), представленный в статье, обеспечивает естественную логическую и вероятностную основу для тестирования программного обеспечения, которая позволяет решить ряд проблем.

Теория BGM [2] привела ко многим новым применениям моделирования в условиях неопределенности, в частности, к сложным проблемам, в которых большое количество факторов вносит свой вклад в общую неопределенность. BGM основаны на байесовской статистической методологии, которая характеризуется предоставлением формальной основы для объединения данных с суждениями экспертов, таких как тестировщики программного обеспечения. Для прикладной области тестирования программного обеспечения будет показано, как должна быть структурирована проблема и как можно использовать полученные модели. Методология проиллюстрирована тематическими исследованиями, возникающими в результате применения подхода к крупномасштабным задачам тестирования программного обеспечения.

Существующие исследования

Общие проблемы надежности программного обеспечения привлекли внимание исследователей статистики [3]. Тем не менее, большая часть этой работы пытается решить проблемы, связанные с надежностью программного обеспечения в рамках существующих математических рамок вместо того, чтобы пытаться тщательно смоделировать фактические неопределенности, возникающие при тестировании программного обеспечения и процессе обучения на основе тестов.

В [4] описана важность тщательного документирования тестовых примеров. Одним из преимуществ является возможность повторения одних и тех же тестов, возможно, другим тестировщиком. Двумя дополнительными заявленными преимуществами являются простота проверки качества тестовых примеров и оценка качества целевого программного обеспечения. Однако не уточняется, как выполнить такую проверку и оценку. Эти проблемы и требования четко решаются с помощью подхода BGM. Еще одним важным тестом является 48-часовой тест непрерывной работы, который направлен на выявление ошибок, связанных с утечкой памяти, взаимоблокировкой и временем ожидания соединения. Важность такого тестирования неоспорима.

В [5] представлен другой подход к количественной оценке надежности программного обеспечения путем размещения функциональной архитектуры программного обеспечения централизованно в модели. Авторы предполагают, что модель «поощряет философию тестирования, направленную на запуск режимов сбоев и устранение связанных с ними неисправностей», но они не предоставляют дополнительных рекомендаций по тестированию на уровне входного раздела. Было бы интересно рассмотреть возможность использования таких или связанных моделей в процессе создания BGM для тестирования программного обеспечения.

В [6] определены две основные цели при тестировании программного обеспечения: достижение надлежащего качества (отладочное тестирование) и оценка существующего качества (оперативное тестирование). Целью отладочного тестирования является проверка программного обеспечения на наличие дефектов, чтобы их можно было устранить. Цель эксплуатационного тестирования – получить уверенность в надежности программного обеспечения. Исследована взаимосвязь между этими целями тестирования с помощью вероятностного анализа, в котором эффективность тестирования основана на надежности программы после тестирования. Оба подхода основаны на субъективных аргументах: отладочное тестирование основывается на понимании того, где могут быть ошибки; оперативное тестирование зависит от знаний и предположений об операционных профилях. Оба подхода имеют преимущества в зависимости от практической ситуации, и оба зависят от разделения входного пространства. Подход BGM также требует разделения входного пространства. Предлагаемое разделение обусловлено сосредоточением внимания на различных программных действиях (software actions, SA), которые являются необходимыми для тестирования сложного программного обеспечения. В [6] тщательно обсуждается сложная проблема определения «неисправностей», которые ответственны за сбои. Формальная обработка «неисправностей» недоступна, и предлагается использовать «области отказов» входного пространства, где одна такая область представляет собой набор точек отказа, которые устраняются изменением программы. Детальное эксплуатационное тестирование позволяет проводить статистический анализ надежности работающего программного обеспечения на основе большого количества тестов. Хотя такое тестирование целесообразно, когда это возможно, ограничения ресурсов для приложений, а также неточные знания об операционных профилях часто мешают применять такое тестирование. Однако некоторые аспекты рабочих профилей отражены в утилитах, которые влияют на дизайн наборов тестов.

Одним из аспектов, общих для большинства практических подходов к тестированию программного обеспечения, является разделение входного домена на поддомены таким обра-

зом, чтобы любой вход в одном и том же поддомене считался оказывающим одинаковое влияние на систему. Затем задача тестирования становится задачей выбора по крайней мере одного представителя из каждого раздела. Затем искусство тестирования превращается в искусство определения разделов. Такие разделы могут быть созданы либо из представления системы в виде черного ящика, использующего знания требований и спецификаций программного обеспечения, с помощью представления в виде белого ящика, использующего структурный анализ кода, либо с помощью комбинации этих двух представлений. Хотя существует ряд методов, которые можно использовать, часто считается, что этот процесс все еще сопряжен с трудностями и проблемами [7]. В [8] обсуждается оценка частоты отказов на основе как случайного, так и секционного тестирования. Исследование сильно зависит от предполагаемого операционного профиля. В предлагаемом подходе также разделяется входная область с использованием группы похожих входных данных, где сходство является субъективной оценкой в отношении общих программных действий.

Регрессионное тестирование – это процесс тестирования программы после внесения в нее изменений, чтобы убедиться, что изменения были эффективными и не привели к дальнейшим ошибкам. Для такого тестирования уже существует набор предыдущих тестовых примеров. Обычно нет возможности повторно выполнить все эти тесты, поэтому необходимо разработать какой-то метод выбора регрессионного теста. В [9] даны оценки современных методов регрессионного тестирования. Все проанализированные методы основаны на информации об исходном коде до и после модификации, и все же ни один из методов не пытается получить экспертные знания о программном обеспечении и тестах. Регрессионное тестирование явно не рассматривается.

Многие статистические методы были предложены или использованы в попытках создать более совершенное программное обеспечение. В [10] описано возможное использование методов классического статистического контроля качества, хотя адаптация к проблемам программного обеспечения довольно расплывчата. В [3] дан обзор активной сферы статистических исследований в области надежности программного обеспечения. В основном это вклад в теорию случайных процессов, связывающий показатели надежности с предполагаемым поведением процессов, при которых происходят сбои как во время тестирования, так и в процессе эксплуатации. Хотя такие подходы потенциально интересны, прямое применение в настоящее время возможно только для программного обеспечения довольно ограниченной сложности.

Классический подход к экспериментальному проектированию может быть полезен в некоторых ситуациях тестирования программного обеспечения при выборе наборов тестов. В [11] обсуждаются факторные конструкции, которые кажутся перспективными для небольших приложений. В сложных ситуациях простая адаптация указанного подхода невозможна. Предложенный алгоритм автоматического проектирования тестов использует простые пошаговые методы для создания эффективных наборов тестов в соответствии с соответствующими критериями. Эти методы значительно улучшают существующую практику, хотя есть возможности для совершенствования, что является предметом продолжающихся исследований. По сравнению с [11] преимущество предложенного подхода заключается в том, что он основан на правильном структурировании и обеспечивает результаты, которые делают тестирование функциональности во многих областях как возможным, так и желательным, и что эти результаты определяют соответствующий экспериментальный дизайн. Характер проведенных тематических исследований таков, что тестируемое программное обеспечение рассматривается как черный ящик, поскольку структуры связаны с подробными знаниями тестировщика. Если доступна подробная информация об анализе кода, она может быть встроена в наш подход.

BGM, также называемые байесовскими сетями доверия (BBN), применяются для решения различных проблем качества программного обеспечения. В [12] их используют для прогнозирования качества программного обеспечения с учетом множества факторов, таких как

усилия и сложность проектирования, навыки людей, участвующих в процессе разработки и тестирования, и затраты. Хотя это интересный подход, позволяющий получить общее представление о плотности дефектов в части программного обеспечения, эти BBN не предназначены непосредственно для оказания помощи тестировщикам. Наиболее типичное применение BBN в этой прикладной области заключается в выводе моделей надежности из больших баз данных. Проект http://www.hugin.dk/serene/ представляет интересные приложения BBN в области оценки безопасности и рисков, и некоторые работы в рамках этого проекта также учитывают программное обеспечение [10], хотя и без фактической поддержки на уровне разработки тестов. Использование BGM, а не BBN, отражает терминологию, используемую в более широкой статистической литературе, и помогает подчеркнуть, что наши суждения тестировщиков моделей BGM не являются результатами вывода моделей из больших БД.

Моделирование работы программного обеспечения

Предположим, что нас в первую очередь интересует вероятность события N, что программное обеспечение не содержит ошибок. Это событие декомпозируем, рассматривая два перекрывающихся события S, что по крайней мере одно из коротких чисел обрабатывается неправильно, и L, что по крайней мере одно из длинных чисел обрабатывается неправильно. Попробуем смоделировать убеждения по S, L, N с использованием BGM.

В BGM представляем каждое неопределенное событие узлом. Некоторые узлы могут быть соединены направленными дугами. Если направленная дуга проходит от узла A к узлу B, то узел A называется родительским для B, а узел B называется дочерним для A. Назовем узел A корневым узлом, если у A нет родителей. Узел B является потомком узла A, если существует направленный путь от A до B. Обычно направленная дуга от узла A до B указывает, что на вероятность для узла B влияет значение узла A. Более того, совокупность родителей узла определяет вероятность в узле в следующем смысле: для любой пары узлов B и C, которые соединены через родительские узлы, если указать результаты всех этих родительских узлов, то B условно независим от C.

Чтобы построить графическую модель, построим ориентированный ациклический граф для представления качественных взаимосвязей между узлами. Затем количественно оцениваем все вероятности событий на графе следующим образом. Сначала определим вероятность наступления каждого события, представленного корневым узлом. Во-вторых, для каждого дочернего узла определим вероятность наступления события, представленного узлом, при условии наступления или отсутствия каждого родительского узла этого узла. Эти спецификации определяют вероятности для всех комбинаций событий, представленных на графе. Поскольку задано полное совместное распределение вероятностей, каждый раз, когда наблюдается событие (например, успешное тестирование программного обеспечения), можно обновлять все вероятности для всех оставшихся событий в модели (например, событие, в котором нет ошибок в программном обеспечении). Кроме того, из-за структуры графической модели легко выполнить такое обновление убеждений даже для больших моделей, используя принципы локальных вычислений. Общие методы BGM можно найти в [2].

В примере есть три события, S, L и N. Один из способов понять взаимосвязь между событиями S и L – это рассмотреть три возможности ошибок в коде: проблемы, которые влияют только на короткие числа, проблемы, которые влияют только на длинные числа, и проблемы, общие для всех чисел. Введем три дополнительных события S*, L* и C*, чтобы выразить эти возможности, так что S* – это событие, при котором в коде возникает проблема, которая может привести к ошибкам для коротких, но не длинных чисел, L* соответствует ошибкам в длинных, но не коротких числах, а C* соответствует ошибкам, которые могут возникнуть в любом числе.

Эти события не поддаются прямому наблюдению, но они полезны для объяснения убеждений, касающихся наблюдаемых событий S и L. Будем рассматривать S*, L* и C* как независимые. Графическая модель представлена на рисунке 1. Существует три независимых кор-

невых узла: S*, C* и L*. Узел S имеет родителей S* и C*, узел L имеет родителей C* и L*, а узел N имеет родителей S и L. Таким образом, S и L независимы, учитывая S*, L* и C*, а N не зависит от S*, C* и L*, учитывая S и L.

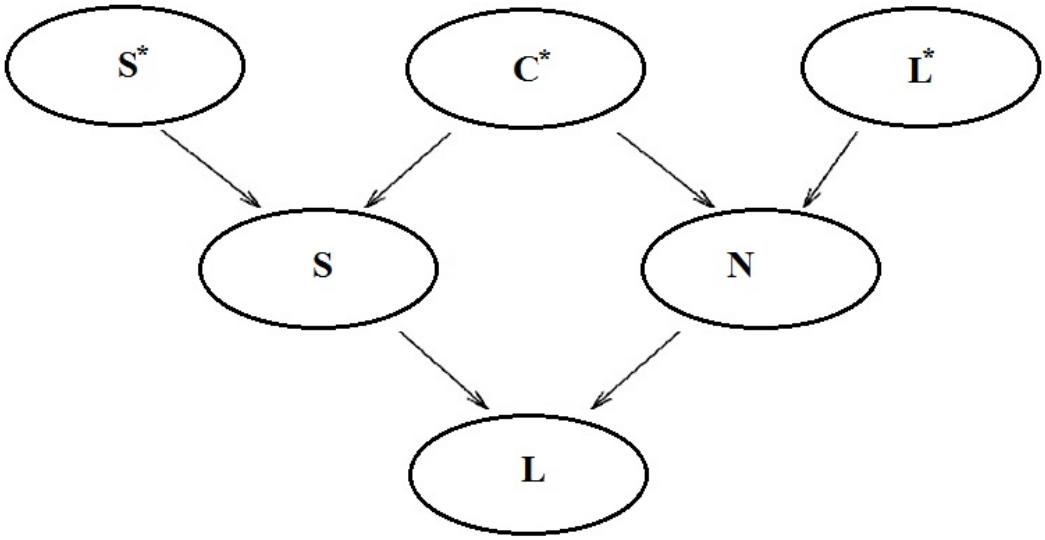

Рисунок 1 – Байесовская графическая модель, показывающая структуру, лежащую в основе обработки ошибок числовых данных и суждений о причинах и их взаимосвязи
Figure 1 – Bayesian graphical model showing the structure underlying the processing of numerical data errors and judgments about the causes and their relationship

Теперь количественно оценим убеждения по модели. Сначала определим вероятности для корневых узлов, начиная с ранжирования относительных вероятностей отказа. Пусть P(C*) – вероятность того, что в программном обеспечении есть хотя бы одна ошибка, соответствующая C*. Предположим, что тестировщик считает, что P(C*) намного больше, чем P(L*), что намного больше, чем P(S*), которая считается небольшой. Хотя может быть трудно дать точные значения для этих вероятностей, обычно можно дать значения, которые кажутся разумными порядками величины. Когда проводится полный вероятностный анализ, проверяется чувствительность выводов путем изменения вероятностных исходных данные. Обычно выводы не будут чувствительны к небольшим изменениям в вероятностных параметрах. Однако, если выводы о надежности программного обеспечения действительно критически зависят от определенных входных значений, то неизбежный вывод состоит в том, что эти значения должны быть тщательно рассмотрены. Таким образом, определяются те аспекты количественной оценки, в которых важно быть точным и которые имеют серьезные последствия для процесса тестирования.

Теперь определим условные вероятности для каждого дочернего узла с учетом каждого родительского узла. В примере предположим, что считаем вероятность того, что S произойдет, равной единице, если произойдет либо S*, либо C*, либо оба, и 0 – в противном случае, и вероятность того, что L произойдет, равна единице, если произойдет либо L*, либо C*, либо оба, и 0 – в противном случае. Аналогично вероятность того, что N произойдет, равна единице, если ни S, ни L не произойдут, и равна 0 – в противном случае. В этом примере все ссылки являются детерминированными, т.е. если происходит родительское событие, то также произойдет дочернее событие. В общем случае, если родительский корневой узел в структуре имеет один дочерний узел, то обычно присваивается вероятность того, что дочерний узел унаследует проблему от родительского. В других случаях связи могут быть вероятностными; например, если общая проблема C* потенциально может повлиять на многие дочерние узлы, то можно было бы определить некоторые дочерние узлы, на которые определенно повлияет общий узел, некоторые из них будут затронуты с высокой вероятностью, поскольку они имеют много общих характеристик с этими узлами, и некоторые, которые будут затронуты с низкой вероятностью, поскольку у них мало общих характеристик.

Привязка результатов тестирования к узлам домена

Чтобы завершить графическую модель, необходимо связать результаты тестирования с графом. Рассмотрим тесты коротких чисел. Если есть хотя бы одна ошибка в обработке короткого числа, то необходимо учитывать долю p_S коротких чисел, которые будут давать правильные ответы при тестировании. Величина p_S неизвестна и, следовательно, должна быть описана вероятностно. Простая, но полезная форма состоит в том, чтобы рассматривать распределение вероятностей для p_S как смесь двух компонентов. Первый компонент представляет собой точку массы q_S при значении $p_S = 0$. Следовательно, q_S – вероятность того, что, учитывая наличие ошибки по крайней мере в одном коротком числе, эта ошибка возникнет в любом коротком числе, которое будет выбрано для проверки. Во многих случаях этого будет достаточно, так как успех или неудача одного теста определит, будет ли узел ошибаться всегда или никогда. Для более общих случаев оставшаяся вероятность $q'_S=1-q_S$ представлена непрерывной плотностью вероятности на (0,1). Естественным выбором было бы предположить, что p_S имеет бета-распределение $Be(\alpha_S, \beta_S)$, так что плотность вероятности p_S есть

$$f(p)=\frac{\Gamma(\alpha_S+\beta_S)}{\Gamma(\alpha_S)\Gamma(\beta_S)}p^{\alpha_S-1}(1-p)^{\beta_S-1},\ 0\le p\le 1,$$

с параметрами α_S, β_S, которые необходимо выбрать, чтобы представить наши убеждения о форме предыдущего распределения.

Величины $\alpha_S = \beta_S = 1$ соответствуют равномерному распределению на [0,1] и изменение отношения $\alpha_S/(\alpha_S + \beta_S)$ перемещает центр распределения от 0 к 1, пока большие значения одного из двух параметров соответствуют малой вариации распределения. Выбор подходящих значений может быть существенно упрощен с использованием компьютерных вычислений.

Выбираем бета-распределение как стандартное распределение для двоичной выборки. Отсюда условное распределение p_S с учетом, что $p_S \neq 0$, если наблюдается набор результатов теста, также является бета-распределением. Следствием прохождения теста является обновление параметров α_S, β_S как $\alpha_S + 1$, β_S, в то время как сбой теста обновляет значения как α_S, $\beta_S + 1$. Это значительно упрощает вычисления для обновления вероятностей. При этом каждый проход теста приближает распределение вероятностей для p_S к единице, поскольку увеличивается общая вероятность наблюдения за успехами теста. Аналогичным образом строится модель для тестов на больших числах путем выбора значений q_L, α_L, β_L.

Предположим, что проводится m_S тестов для разных небольших чисел и m_L тестов для больших чисел. К графу (рисунок 1) добавляются узлы P_S и P_L для представления моделей наблюдаемых тестов для каждого наблюдаемого вместе с узлами фактического теста, что дает рисунок 2. Показаны только узлы для первого и последнего тестов в каждой группе. Поскольку P_S, P_L являются непрерывными случайными величинами, необходимо указать условную вероятность каждого дочернего узла, а именно каждого результата теста, обусловленного каждым возможным значением родительского узла. В данном случае это просто, так как вероятность того, что тест завершится неудачей, например, для короткого числа, равна числовому значению родительского узла P_S. На рисунке 2 единственным родителем, который выбран для P_S, является событие S. В более сложной модели может быть несколько ссылок на P_S.

Преобразование программных действий в BGM

Первоначальное структурирование приведет к определению ряда наблюдаемых узлов, которые назовем узлами домена, каждый из которых состоит из SA и отдельной группы входных данных. Далее необходимо преобразовать эти структуры в BGM в соответствии с общими характеристиками (рисунок 3). Кроме того, необходимо обрабатывать любые SA, которые были оценены как связанные, например, из-за сходства транзакции или из-за суждения тестировщика о том, что SA разделяет общие базовые операционные механизмы, кото-

рые, как считается, не стоит моделировать более подробно. Эти проблемы решаются следующим образом: формируется отдельный кластер SA, которые взаимосвязаны. То есть, если SA A, B и C связаны, а SA D и E связаны, работаем с отдельными кластерами {A, B, C} и {D, E}, где кластеры независимы в том смысле, что SA в одном кластере независимы от SA в любом другом кластере. Рассматриваем каждый кластер как составной SA, который способен выполнять любой или все его компоненты SA, и вводим характеристику маркера для обозначения компонента SA. С этого момента просто обрабатываем характеристику маркера так же, как и любую другую входную характеристику, и разделяем ее соответствующим образом.

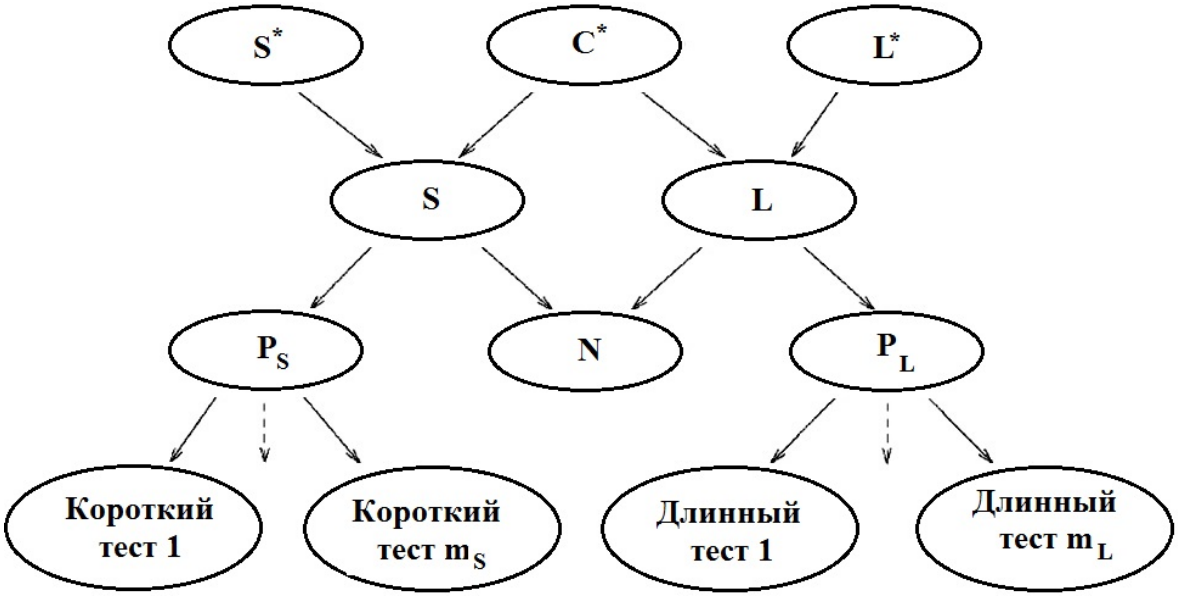

Рисунок 2 – Привязка множества возможных результатов тестирования к базовому BGM
Figure 2 – Linking the set of possible test results to basic BGM

Оценка предыдущих спецификаций

Чтобы обеспечить моделирование суждений тестировщика, необходимо получить спецификации, описанные ниже.

Предварительные распределения для тестов для узлов домена

Для каждого узла домена тестировщик оценивает q_S, вероятность того, что, если предположить, что тест для одного входа завершится неудачей, все остальные входы также завершатся неудачей. Если $q_S < 1$, необходимо выбрать распределение вероятностей f(p) для доли оставшихся входных данных, которые будут существовать. Для этого используем бета-версии дистрибутивов, которые удобны в данной ситуации и обеспечивают достаточную гибкость.

Оценка родительского влияния

Графические модели, которые создаются, можно интерпретировать как имеющие родительские узлы, где возникают ошибки, и дочерние узлы, где обнаруживаются ошибки. Неформально, если предположить, что в родительском узле есть неисправности (но нигде больше нет неисправностей), значение дуги состоит в том, чтобы показать (используя вероятность) потенциал родительского узла, чтобы вызвать неисправность в дочернем узле.

Дуга между родительским и дочерними узлами на графике помечена значением, которое представляет вероятность того, что, если у родителя есть хотя бы одна ошибка, он передаст ошибку дочернему узлу. Чтобы указать таблицы условных вероятностей дочерних узлов с учетом родительских узлов, используем следующую схему, называемую зашумленным ИЛИ-вентилем. Пусть $X = 0$, $X = 1$ – события, соответствующие ситуациям, когда программное

обеспечение узла X работает без сбоев и по крайней мере с одним сбоем соответственно. Для узла X с родителями Y_i, i = 1…k, определим такие вероятности

$$P(X=0|Y_1=0\cap...\cap Y_i=1\cap...\cap Y_k=0)=v_i,$$

что мы считаем $c_i = 1\text{-}v_i$ вероятностью того, что родитель передаст свою ошибку. Затем, для любой конфигурации родителей Y, $P(X=0\,|\,Y)=\prod_{i:Y_i=1} v_i$. Наконец, поскольку все причины сбоя были явно смоделированы, $P(X=0\,|\,Y=0)=0$. Основное предположение заключается в том, что Y_1, …, Y_k не могут передавать свои ошибки независимо.

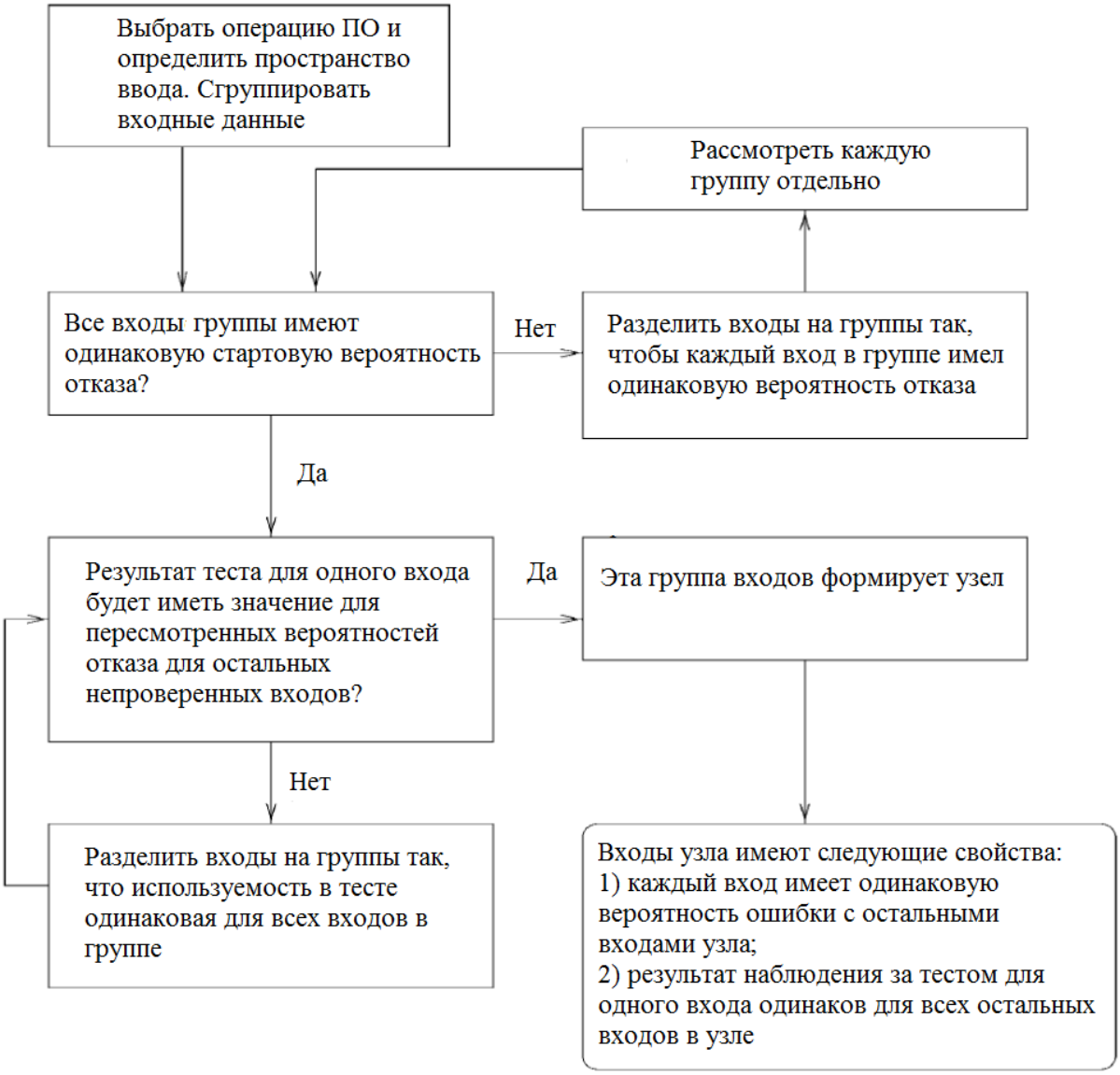

Рисунок 3 – Разделение пространства входов
Figure 3 – Separation of input space

Выявление корневых вероятностей

При определении вероятностей корневых узлов может быть трудно прийти к точным суждениям. Однако калибровка и анализ чувствительности будут выполняться после первоначальной спецификации для достижения большей точности.

Информация, которая имеет непосредственное отношение к процессу выявления, включает историческую информацию, такую как результаты предыдущих тестов или результаты тестов для аналогичных областей программного обеспечения. Дополнительные вопросы, которые необходимо рассмотреть тестировщику, включают следующее.

1. Насколько сложен код для узла? Возможно, чем выше сложность, тем выше вероятность ошибки.

2. Для сравнения, считает ли тестировщик этот код более или менее надежным, чем другие фрагменты кода, для которых есть лучшее представление о надежности?

3. Является ли код старым кодом, который прошел множество предыдущих тестов, или это принципиально новый код для реализации новых функций, или он находится где-то посередине?

4. Какова исторически была надежность автора программного обеспечения, ответственного за этот фрагмент кода? Проверенный производитель программного обеспечения или совершенно новый поставщик?

5. Был ли подобный код протестирован в прошлом и признан надежным или ненадежным?

Измерение последствий и приоритизация неисправностей

Надежность программного обеспечения может быть оценена многими различными способами; например, естественным результатом подхода BGM является получение вероятности того, что узел или любая сеть, сформированная из группы узлов, имеет хотя бы одну неисправность. Другой мерой является ожидаемое количество узлов домена, содержащих по крайней мере одну ошибку. Одним из естественных выражений надежности моделируемого программного обеспечения является шкала полезности (покрытия): байесовский подход обеспечивает естественную основу для сочетания вероятностей и полезностей для принятия решений [13]. Например, каждый узел домена либо проходит, либо не проходит тест, и в случае сбоя конкретного узла владелец программного обеспечения несет ответственность за это. К таким последствиям могут быть приписаны числовые значения, возможно, отражающие ожидаемые затраты компании на использование программного обеспечения, с учетом, что оно содержит эту ошибку.

Возможная классификация ошибок выглядит следующим образом. Приоритетные ошибки оказывают катастрофическое влияние на работу программного обеспечения и могут нанести ущерб, если они останутся незамеченными. Серьезные ошибки оказывают существенное влияние на работу программного обеспечения и могут вызвать некоторое затруднение, если они останутся незамеченными. Незначительные неисправности имеют локальный эффект и могут быть устранены, а могут и не быть устранены при их обнаружении. Незначитльные ошибки обычно не устраняются, если только не будет проведена серьезная доработка программного обеспечения. Для анализа рисков узлы домена могут быть классифицированы в соответствии с приоритетом неисправности и вероятностями отказа, а также с другими сводными показателями, рассчитанными отдельно для каждой категории. Для анализа решений было бы более естественным объединить показатели в масштабе покрытия. Например, владелец программного обеспечения может рассудить, что стоимость для каждой категории неисправностей составляет соответственно 100000, 20000, 1000 и 1 единицу потерь. Такие утилиты играют важную роль в разработке наборов тестов, поскольку они позволяют нам сосредоточиться на более важных ошибках.

Процедура вероятностной спецификации BGM

Опишем формальную процедуру вероятностной спецификации BGM. Особое преимущество этой процедуры заключается в том, что она может генерировать количественную оценку на основе расплывчатых знаний, но также является достаточно гибкой, чтобы обеспечить количественную оценку подробных знаний. Процедура состоит из двух этапов. Во-первых, получаем разумное приближение к начальной спецификации, задав тестировщику несколько простых вопросов для оценки относительной надежности различных областей программной системы. Во-вторых, берем исходную спецификацию и уточняем ее в свете калибровочных анализов и другой соответствующей информации.

Как правило, SA будет разделено на несколько не связанных BGM. Начинаем с того, что просим тестировщика ранжировать все BGM с точки зрения ожидаемой надежности перед тестированием. Затем каждый BGM конструируется путем разделения SA в соответствии с характеристиками, такими как длина номера и начальная цифра. Таким образом, тестировщику предлагается ранжировать в рамках каждого BGM отдельно характеристики с точки зрения ожидаемой надежности перед тестированием. Затем каждая характеристика делится на группы входных данных; например, длина характеристического номера может быть разделена на 8-значные, 9-значные и 10-16-значные числа. Для корневых узлов для построенных BGM обычно необходимо оценивать надежность для общих программных проблем, связанных с характеристикой, и для проблем, возникающих для конкретных элементов характеристики. Таким образом, тестировщику предлагается ранжировать по каждой характеристике отдельно общие и специфические элементы с точки зрения ожидаемой надежности перед тестированием. Наконец, любые корневые узлы, сформированные из комбинаций характеристик, используемых для разделения входного пространства, ранжируются среди общих причин и конкретных величин разделения.

После завершения ранжирования указывается два начальных параметра настройки: p отражает общий уровень ненадежности в наименее надежном BGM и d отражает общий уровень ненадежности в наиболее надежном BGM по сравнению с наиболее ненадежным BGM. Параметр p соответствует вероятности наличия хотя бы одной неисправности в самом ненадежном корневом узле в самом ненадежном BGM; dp соответствует вероятности наличия хотя бы одной неисправности в самом ненадежном корневом узле в самом надежном BGM. Важно отметить, что это первоначальные суждения: они обеспечивают основу для дальнейшей калибровки и анализа чувствительности, и есть все возможности для пересмотра суждений в соответствии с рекомендациями такого анализа.

Для первоначальной количественной оценки BGM необходимо выполнить следующие действия.

Обозначим n независимых графических моделей как G_1, G_2, …, G_n. Обозначим n_i характеристик в модели G_i как G_{i1}, G_{i2}, …, G_{ini}.

Затем обозначим n_{ij} признаков характеристики C_{ij} как L_{ij1}, L_{ij2}, …, L_{ijnij} так, чтобы L_{ijk} было k-м признаком j-й характеристики графической модели. Аналогично пусть r_i, r_{ij}, r_{ijk} будут рангами, присвоенными G_i, C_{ij}, L_{ijk} соответственно. Пусть p_{ijk} - вероятность наличия хотя бы одной неисправности в корневом узле L_{ijk}. Цель выявления состоит в том, чтобы установить вероятности $\{p_{ijk}\}$ для корневых узлов $\{L_{ijk}\}$.

Подход заключается в том, чтобы рассчитать

$$p_i = p - \frac{(r_i - 1)p(1-d)}{n_i - 1},\ i=1...n.$$

$$p_{ij} = p_i - \frac{(r_{ij} - 1)p_i(1-d_i)}{n_{ij} - 1},\ i=1...n,\ j=1...n_i;$$

$$p_{ijk} = p_{ij} - \frac{(r_{ijk} - 1)p_{ij}(1-d_{ij})}{n_{ijk} - 1},\ i=1...n,\ j=1...n_i,\ k=1...n_{ij}.$$

Остается выбрать подходящие значения для d_i и d_{ij}. Для характеристик d_i соответствует d для сетей в том смысле, что это дает приблизительное представление для модели G_i относительно относительной надежности наиболее и наименее ненадежных характеристик в модели G_i. Для элементов внутри характеристик d_{ij} соответствует d для сетей и d_i для характеристик и дает приблизительное указание для характеристики C_{ij} относительно относительного соотношения между наиболее ненадежным и наименее ненадежным элементами в характеристике C_{ij}. Изначально целесообразно выбрать $d_i = d$ для всех и $d_{ij} = d$ для всех i, j. Вероятность для корневого узла, таким образом, состоит из его рейтинга надежности в пределах его локальной области, умноженного на параметр настройки вероятности для этой локальной об-

ласти, модифицированный параметром настройки, выражающим разницу в надежности между наименее и наиболее надежными корневыми узлами для этой локальной области.

Анализ финальной чувствительности

Важно изучить чувствительность в отношении процесса тестирования, в частности, потому, что это помогает принимать обоснованные решения относительно того, когда программное обеспечение будет готово к выпуску, учитывая конкретный набор тестов. Общей особенностью байесовского анализа является то, что различия в предыдущих убеждениях часто в значительной степени устраняются путем наблюдения за данными, и важно то, как изменения в предыдущих убеждениях влияют на последующую вероятность того, что программное обеспечение готово к выпуску.

Рассмотрим ситуацию, когда предусмотрен конкретный набор тестов, либо исторический, либо создаваемый автоматически с использованием BGM, и когда необходимо изучить чувствительность начальной спецификации по отношению к этому набору тестов. Простой подход, основанный на разовых изменениях корневых узлов, заключается в следующем.

1. Выбрать и откалибровать модель.

2. Применить данный набор тестов к модели и рассчитать желаемые последующие результаты, предполагая, что все тесты прошли успешно.

3. Оставляя все остальные корневые узлы неизменными, для каждого корневого узла в модели, в свою очередь, берем вероятность хотя бы одной ошибки и изменяем в соответствующем масштабе.

4. Затем применяется данный набор тестов к измененной модели и вычисляем желаемые последующие результаты.

5. Различия между последующими сводками для исходной и измененной моделей измеряют чувствительность выходов к спецификации вероятности для измененного корневого узла.

Также можно оценить чувствительность в глобальном масштабе, изменяя параметры настройки, такие как p и d, и измеряя изменения в выходных данных BGM. Другие подходы к измерению чувствительности к предыдущим спецификациям обсуждаются, например, в [14].

Заключение

Описан подход к вероятностному графовому моделированию и анализу программных систем. На практике модели, которые создаются, могут быть сложными. Однако они никогда не будут более сложными, чем соображения, которые тестировщик программного обеспечения в любом случае должен учитывать при оценке надежности программного обеспечения, поскольку все различия, которые вводятся в модель, существуют, потому что тестировщик счел эти различия важными. Все, что сделано: это, во-первых, создано описание, определяющее различные отдельные группы тестов, которые можно было бы выполнить, и, во-вторых, сформулировано требование предварительной количественной оценки неопределенностей на графе. В любом случае рекомендуется выполнять первый этап этого процесса при планировании набора тестов программного обеспечения. Что касается количественной оценки, то во многих случаях будет сравнительно просто ранжировать относительные вероятности для различных источников неисправности. Когда размещаются точные данные на графе, может случиться так, что анализ модели будет относительно нечувствителен к изменениям предыдущих значений при условии соблюдения предыдущих ограничений; в этом случае можно быть уверенным, что программное обеспечение было протестировано с высокой степенью надежности. В качестве альтернативы можно обнаружить, что существуют правдоподобные предварительные входные данные, по которым нельзя сделать вывод о надежности программного обеспечения, учитывая результаты тестирования. В таких случаях необходимо либо более тщательно моделировать, либо проводить более тщательное тестирование. Процесс моделирования в сочетании с тщательным анализом чувствительности заставляет задуматься о том, насколько можно защитить суждения о надежности программного обеспече-

ния. Альтернатива, а именно принятие тех же самых суждений о надежности программного обеспечения без основы для анализа неопределенностей, с гораздо большей вероятностью приведет к ошибочным суждениям. Кроме того при построении модели выявляются различные преимущества, вытекающие из вероятностного анализа, такие как автоматическая генерация хороших наборов тестов, количественный подход к анализу рисков для выпуска программного обеспечения и так далее.

Модель, которая описана, использует экспертные суждения тестировщика. Если эти суждения будут чрезмерно упрощенными, то модель не даст хорошего представления о неисправностях в программном обеспечении. В таких случаях модель все равно улучшит анализ тестировщика, но с дополнительным преимуществом, заключающимся в том, что различные предположения тестировщика могут быть явно проанализированы в модели с помощью диагностики модели для BGM. Эта диагностика основана на несоответствиях между фактическим поведением теста и прогнозируемым поведением в соответствии с моделью; например, предсказание нескольких разломов в данной подобласти может быть опровергнуто наблюдением нескольких разломов в этой области.

Наконец, необходимо учесть, что подход BGM отражает, насколько это практически возможно, опыт тестировщика и поддерживает его в пригодной для использования форме в качестве базы знаний. Это представляет собой значительный ресурс для владельца программного обеспечения, который регулярно сталкивается с проблемой потери опыта тестировщика из-за повседневных практических задач, таких как кадровые изменения.

Библиографический список

1. **Redmill F.** Why Systems Go Up in Smoke. The Computer Bulletin, pp. 26-28, Sept. 1999.

2. **Cowell R. G., Dawid A. P., Lauritzen S., Spiegelhalter D. J.** Probabilistic networks and expert systems. Exact computational methods for Bayesian networks. International Statistical Review, 2008, vol. 76.

3. Application of Statistical Methods in Software Engineering: Theory and Practice / **T. Sirqueira, M. A. Miguel, H. Dalpra, J. M. N. David.** University Center Academia, Brazil, 2020. 22 p.

4. **Olsson T., Väänänen K.** How does AI challenge design practice? Interactions, 2021, vol. 28(4), pp. 62-64.

5. **Smidts C., Sova D.** An Architectural Model for Software Reliability Quantification: Sources of Data. Reliability Eng. and System Safety, vol. 64, pp. 279-290, 1999.

6. **Tameem H.** Software evaluation. 2021. https://doi.org/10.6084/m9.figshare.14945583.v1.

7. **Sun C., Dai H. et al.** Adaptive Partition Testing. IEEE Transactions on Computers, 2018, vol. 99. DOI: 10.1109/TC.2018.2866040.

8. **Han M.** Estimation of Failure Rate and its Applications. Advanced Materials Research, 2013, vol. 756-759, pp. 3149-3152.

9. **Yu L., Liu C., Zhang Y.** A multidimensional classification of safe regression test selection techniques. 2012 International Conference on Systems and Informatics (ICSAI2012), 2012, pp. 2516-2520, DOI: 10.1109/ICSAI.2012.6223565.

10. **Kubadet H. M.** The Bayesian Belief Network for Inference. International Journal of Computer Science and Mobile Computing, 2017, vol. 6, iss. 5, pp. 344-346.

11. **Akhtar Y., Phoa F. K. H.** Cost-Efficient Mixed-Level Covering Designs for Testing Experiments. Journal of Statistical Theory and Practice 2020, vol. 14(1), DOI: 10.1007/s42519-019-0062-7.

12. **Olson J. T., Rozenblit J. W., Talarico C., Jacak W.** Hardware/Software Partitioning Using Bayesian Belief Networks. IEEE Transactions on Systems Man and Cybernetics Part A. Systems and Humans, 2007, vol. 37(5), pp. 655-668.

13. Statistical Decision Theory/ S. French, D. Rios Insua. London: Arnold, 2000.

14. **Stokell B.** Computationally Efficient Methods for High-Dimensional Statistical Problems (Doctoral thesis). 2021. https://doi.org/10.17863/CAM.75868.

UDC 004.7

FEATURES OF USING BAYESIAN GRAPHICAL MODELS FOR SOFTWARE TESTING

M. M. Zozulya, Researcher, Voronezh State Technical University, Voronezh, Russia;
orcid.org/0000-0003-3929-7254

The article describes an approach to the problem of software testing. The approach is based on Bayesian graphical models and presents formal mechanisms for logical structuring of software testing problem, probabilistic and statistical processing of uncertainties that need to be eliminated, the process of developing and analyzing tests, as well as the inclusion and application of test results. ***The aim of the work*** *is to provide a mechanism for dynamic representations of software testing problem based on the construction of models. They can be used to develop tests, answer "what if" questions, and provide decision support to managers and testers. The models reflect the knowledge of software tester for further use. The experience of using this approach in case studies is briefly discussed.*

Key words: *bayesian graphical models, expert assessments, knowledge gathering, software reliability, software testing, statistical methods, test development.*

DOI: 10.21667/1995-4565-2022-79-68-80

References

1. **Redmill F.** Why Systems Go Up in Smoke. *The Computer Bulletin*, pp. 26-28, Sept. 1999.

2. **Cowell R. G., Dawid A. P., Lauritzen S., Spiegelhalter D. J.** Probabilistic networks and expert systems. *Exact computational methods for Bayesian networks. International Statistical Review*, 2008, vol. 76.

3. *Application of Statistical Methods in Software Engineering: Theory and Practice* / **T. Sirqueira, M. A. Miguel, H. Dalpra, J. M. N. David.** University Center Academia, Brazil, 2020. 22 p.

4. **Olsson T., Väänänen K.** How does AI challenge design practice? *Interactions*, 2021, vol. 28(4), pp. 62-64.

5. **Smidts C., Sova D.** An Architectural Model for Software Reliability Quantification: Sources of Data. *Reliability Eng. and System Safety*, vol. 64, pp. 279-290, 1999.

6. **Tameem H.** Software evaluation. 2021. https://doi.org/10.6084/m9.figshare.14945583.v1.

7. **Sun C., Dai H. et al.** Adaptive Partition Testing. *IEEE Transactions on Computers,* 2018, vol. 99. DOI: 10.1109/TC.2018.2866040.

8. **Han M.** Estimation of Failure Rate and its Applications. *Advanced Materials Research*, 2013, vol. 756-759, pp. 3149-3152.

9. **Yu L., Liu C., Zhang Y.** A multidimensional classification of safe regression test selection techniques. *2012 International Conference on Systems and Informatics* (*ICSAI2012*), 2012, pp. 2516-2520, DOI: 10.1109/ICSAI.2012.6223565.

10. **Kubadet H. M.** The Bayesian Belief Network for Inference. *International Journal of Computer Science and Mobile Computing*, 2017, vol. 6, iss. 5, pp. 344-346.

11. **Akhtar Y., Phoa F. K. H.** Cost-Efficient Mixed-Level Covering Designs for Testing Experiments. *Journal of Statistical Theory and Practice* 2020, vol. 14(1), DOI: 10.1007/s42519-019-0062-7.

12. **Olson J. T., Rozenblit J. W., Talarico C., Jacak W.** Hardware/Software Partitioning Using Bayesian Belief Networks. *IEEE Transactions on Systems Man and Cybernetics Part A. Systems and Humans*, 2007, vol. 37(5), pp. 655-668.

13. *Statistical Decision Theory*/ S. French, D. Rios Insua. London: Arnold, 2000.

14. **Stokell B.** Computationally Efficient Methods for High-Dimensional Statistical Problems (Doctoral thesis). 2021. https://doi.org/10.17863/CAM.75868.