Волканов Д.Ю., Зорин Д.А.

Исследование применимости моделей оценки надёжности для разработки программного обеспечения с открытым исходным кодом

1. Введение

В настоящее время в мире разрабатывается большое количество программных продуктов с открытым исходным кодом (Open Source). В таких проектах участвует большое количество разработчиков, добавляющих свои фрагменты кода в общий репозиторий (хранилище, где находятся различные версии файлов, из которых состоит программа).

Проектом будем считать одну или несколько программ, созданием которых управляет одна команда разработчиков. Большой проект может состоять из подпроектов, реализующих отдельные части, работающие независимо друг от друга. Подпроекты могут состоять из компонент, которые выполняют определённый набор функций, и не используются отдельно. В проекте участвуют программисты, работающие над одним или несколькими подпроектами, тестировщики, и руководители проекта. Одна из основных задач руководителя проекта — определить, когда программу можно предоставлять конечным пользователям.

Основные особенности проектов с открытым кодом следующие:

- Команда тестировщиков непостоянна и работает не в установленные часы, а по мере своих возможностей. Отчёты об ошибках могут поступать в любое время суток (из-за того, что отправители могут находиться в разных городах мира).
- Не гарантируется, что количество времени и усилий, затраченных на тестирование, равномерно.
- Параллельно с тестированием продолжается разработка, поэтому количество ошибок может увеличиться.

В результате, перед руководителями проекта стоят следующие задачи:

- Оценить количество ошибок, оставшихся в проекте, и их критичность.
- Оценить количество ошибок, оставшихся в выбранном компоненте, и их критичность.

- Оценить время, когда текущая версия станет стабильной, то есть количество ошибок и их критичность будут меньше заданного порога.

Для решения этих задач можно использовать два подхода: первый основан на изучении исходного кода программы, второй — на использовании моделей оценки надёжности (МОН) (англ. software reliability growth model, SRGM) — на данных, полученных при тестировании [1]. Далее в статье подробно рассматривается второй подход.

Основным результатом расчётов в этом случае является *функция количества ошибок* или *функция количества (*англ. mean function) — неубывающая функция времени, показывающая количество ошибок, найденных к заданному моменту времени. При получении этой функции возникают задачи, описанные следующем разделе.

2. Постановка задачи

Введем следующие обозначения:

- *f(t)* – количество ошибок, найденных к заданному моменту: данные, полученные из статистики тестирования.
- *m(t)* – функция количества ошибок, полученная при применении какой-либо модели.
- $m(t)_x$ - функция количества ошибок, полученная в момент времени *x* (на основании статистики до момента *x*).

Изначально задана статистика тестирования: множество ошибок, для каждой из которых обязательно задано время обнаружения и компонент, где она была обнаружена.

Для получения функции *m(t)* из исходных данных результатов тестирования требуется решить следующие задачи:

1. Определить период времени, через который необходимо строить новую функцию *m(t)* для получения более точных оценок, учитывающих вновь поступившие данные.
2. Выбрать наиболее подходящую к данному проекту МОН из нескольких возможных.
3. Оценить, какой объем исходных данных необходим для того, чтобы модели давали результаты с точностью не ниже заданной.

В качестве критериев качества моделей предлагается использовать следующие:

- Сходимость: возможность построения *m(t)* по исходным данным результатов тестирования и выбранной МОН.

- Точность: среднеквадратичное отклонение полученной функции количества ошибок от реальной:

$$acc(t) = \frac{\sum_{t_i} \frac{m(t_i)}{f(t_i)}}{t} * 100\%$$

- Устойчивость: оценка в текущий период мало отличается от оценки за предыдущий период:

$$st(t) = \left| \frac{m(\infty)_{t_n}}{m(\infty)_{t_{n-1}}} * 100\% - 100 \right| < k, k = 10$$

Данная работа посвящена решению задач 1 - 3 для некоторых конкретных проектов с открытым исходным кодом.

3. Классификация моделей оценки надёжности

Суть МОН заключается в том, чтобы построить математическую модель, описывающую процесс обнаружения ошибок. Для этого применяются различные вероятностные модели случайных процессов, при этом основной случайной величиной, распределение вероятностей которой необходимо найти, является либо количество ошибок в некоторый момент или за определённый период, либо время между двумя ошибками. Выбрав общий вид используемой модели (например, неоднородный пуассоновский процесс или цепь Маркова), можно различными статистическими методами оценить его параметры и в результате получить функцию, показывающую наиболее вероятное количество ошибок в некоторый момент, либо оценку для времени до обнаружения следующей ошибки. После этого можно сравнивать функции, полученные в результате применения различных моделей, между собой и выбирать наилучшую. На основании выбранной модели можно строить оценки различных величин, например, ожидаемое количество ошибок в ближайший месяц или общее количество оставшихся ошибок, на основании которых можно принимать решение о готовности программы к выпуску.

Условно можно разделить все модели на три типа:

1. Марковские. В марковских моделях используется предположение, что количество оставшихся ошибок зависит от текущего состояния системы и не зависит от прошлых состояний [1].
2. Пуассоновские. Предполагается, что обнаружение ошибок можно описать моделью «Поток редких событий» [2], если предположить, что ошибки независимы, обнаруживаются по одной, и известен характер изменения частоты их обнаружения.

3. Байесовские. Время между ошибками считается случайной величиной с показательным распределением, но его параметры зависят от времени появления предыдущих ошибок, при этом оценивается только время до ближайшей ошибки, но не общее количество [3].

Возможны также комбинации этих типов, например, в марковскую модель добавляется байесовская модификация.

Применение МОН к проприетарному программному обеспечению достаточно хорошо исследовано (например, в работах [4], [5], [6]). Проприетарное программное обеспечение в большей степени, чем открытое, удовлетворяет указанным выше ограничениям, так как команда разработчиков и тестировщиков централизованная, и поэтому процесс разработки и тестирования является более равномерным. Кроме того, МОН изначально создавались без учёта возможности применения к проектам с открытым исходным кодом, так как в то время (70-е годы) современный способ разработки проектов с открытым кодом ещё не существовал. Поэтому применимость МОН к открытому программному обеспечению требует дополнительного исследования.

В данном исследовании были проанализированы наиболее подробно описанные в литературе и часто используемые модели оценки надежности:

- Goel-Okumoto (GO) [1]
- S-Shaped (SS) [1]
- Logarithmic (Log) [5]
- Jelinski-Moranda (JM) [4]
- Littlewood-Verrall (LV) [7]

Основной вопрос, появляющийся при использовании SRGM — какую модель использовать в том или ином проекте. Однозначный ответ на этот вопрос найти в общем случае достаточно трудно, более того, нельзя гарантировать, что для одного и того же проекта в течение всего времени разработки будет верна одна модель. Перечислим основные факторы, из-за которых МОН могут работать неточно:

- Малое количество данных. Причиной может быть, например, то, что данные тестирования группируются по неделям или месяцам. Из-за малого количества данных точность оценки параметров снижается.
- Неравномерность тестирования, особенно в сочетании с малым количеством данных
- Стандартные ограничения моделей, то есть наиболее часто встречающиеся допущения о следующих свойствах проекта [7].:

  1. Равномерность: тестирование равномерно, то есть ПО тестируется одинаково до и после построения оценок МОН.

2. Однородность: все ошибки (по крайней мере, в рамках анализируемой группы) равноценны и вероятность их обнаружения одинакова.
3. Независимость: ошибки не зависят друг от друга
4. Идеальность: после обнаружения ошибка немедленно исправляется, и при этом новых ошибок не появляется.
5. Бесконечность: общее число ошибок считается бесконечным, нет возможности получить его оценку напрямую.

Чтобы можно было решить, какую из моделей выбрать, необходимы критерии выбора. В следующем разделе будет предложен алгоритм выбора модели.

4.Алгоритм выбора модели оценки надёжности

В этом разделе предложен алгоритм, позволяющий сравнивать несколько заданных МОН[8]. Алгоритм состоит из следующих шагов:

1. На первом шаге рассчитываются параметры для нескольких моделей (методом максимального правдоподобия). После первого шага остаются только те модели, для которых метод сходится. Сходимость может отсутствовать из-за того, что модель в принципе не подходит (например, взята S-изогнутая кривая, а в реальности она строго выпуклая)
2. Исследуется точность моделей. Для этого применяется любой из существующих методов (минимизация среднеквадратичного отклонения, критерий хи-квадрат и другие). В программе, использовавшейся для исследований, рассчитывается среднеквадратичное отклонение, так как этот способ прост и универсален.
3. Проверка на устойчивость (по формулам из раздела 2).
4. Если на предыдущем шаге не оказалось ни одной устойчивой модели, то необходимо продолжить тестирование и сбор данных ещё на один отчётный период. Если же есть устойчивые модели, то на основе их можно уже решать, является ли текущая версия стабильной.

Описанный алгоритм носит эмпирический характер: модель выбирается после некоторого периода, в течение которого производятся различные промежуточные расчёты, тогда как изначально по свойствам проекта никаких предположений не делается. Для того, чтобы проверить работоспособность алгоритма и адекватность выбранных характеристик точности, необходимо испытать модели и алгоритм на реальных данных. Для этих целей было создано программное средство, позволяющее делать расчёты, и в следующем разделе показаны результаты проведённых с его помощью экспериментов.

5. Экспериментальное исследование

5.1 Цель исследования

Задача исследования — на данных тестирования реальных проектов исследовать различные описанные в литературе модели, и получить решение для задач 1-3 из раздела 2:

1. Определить период времени, через который необходимо строить новую функцию *m(t)* для получения более точных оценок, учитывающих вновь поступившие данные.
2. Выбрать наиболее подходящую к исследуемому проекту МОН из нескольких возможных.
3. Оценить, какой объем исходных данных необходим для того, чтобы модели давали результаты с точностью не ниже заданной.

5.2 Выбор проектов для исследования

Проекты выбирались исходя из следующих требований:

- Большой объем программы (сотни тысяч строк)
- Длительное время тестирования, обширная база данных о тестировании (несколько лет).
- Большое количество разработчиков (несколько сотен)
- Состоит из отдельных компонент.

Для практического исследования были выбраны несколько проектов с открытым исходным кодом. Так как модуль, собирающий статистику, ориентирован на систему Bugzilla [9], выбирались проекты, использующие эту систему для хранения статистики тестирования. В качестве основного был взят проект Eclipse [10], в качестве дополнительных — Mozilla [11] и Linux Kernel [12].

Проект Eclipse был выбран в качестве основного, так как содержит самую большую базу (около 260 000 отчётов об ошибках) с наиболее подробной статистикой: указаны дата, модуль, фатальность ошибки, текущий статус для каждой ошибки.

5.3 Методика экспериментального исследования.

1. По выбранным проектам из системы отслеживания ошибок загружается статистика тестирования.
2. Для этих проектов и, выборочно, их компонент и подмножеств ошибок одинаковой критичности, получены следующие данные:
   - Оценка количества ошибок на ближайший период и сравнение этой оценки с реальными данными (по периодам за некоторое время).
   - Точность и устойчивость рассматриваемых сходящихся моделей, в соответствии с алгоритмом из раздела 4.

3. Используя эти данные, с помощью алгоритма из раздела 4 получается ответ на задачи 2 и 3. Оценивая точность полученных результатов на разных промежутках времени между применениями алгоритма выбора модели, можно получить ответ на задачу 1.

5.4 Эксперименты

Полученные в результате исследования данные показывают, что точность и устойчивость, начиная приблизительно с 10-15 месяца, стабильно отклоняются от 100 не более, чем на 10 процентов (рисунок 1-2), что можно считать приемлемым. Из этого можно сделать вывод о том, что в целом МОН применимы к проектам с открытым исходным кодом, несмотря на децентрализованность тестирования и совмещения тестирования с разработкой.

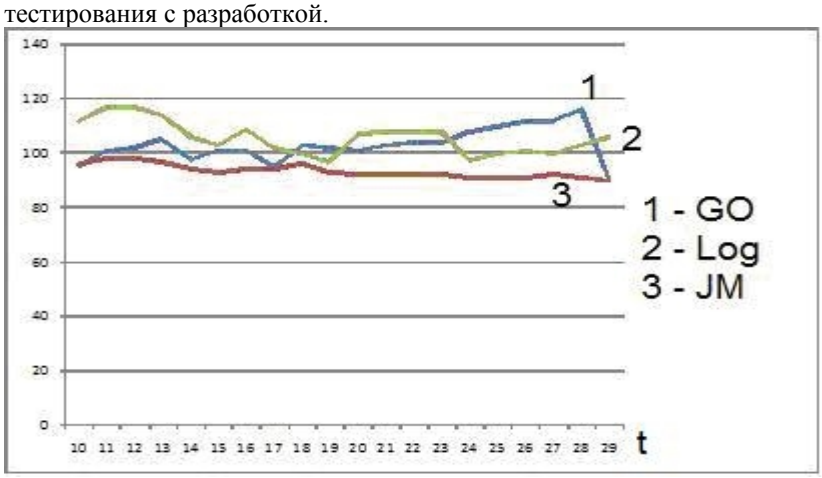

Рисунок 1:Подпроект JDT – функция ac(t)

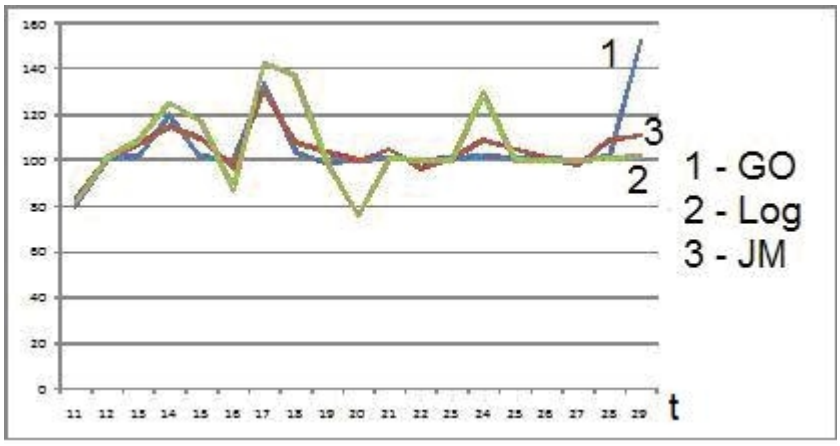

Рисунок 2: Подпроект JDT – функция st(t)

Из экспериментов также сделан вывод, что наиболее приемлемым периодом для перерасчёта является один месяц. При сравнении данных, полученных при разной частоте пересчёта, обнаруживается, что близость оценки к реальным данным при разбиении по месяцам значительно выше, чем при разбиении по неделям (в среднем 10-15% против 30-40%). При разбиении по неделям расхождения оценок и результатах более явные. При разбиении по годам же данных слишком мало и уменьшения количества ошибок не прослеживается. Такие результаты можно объяснить тем, что для расчёта по годам / полугодиям нужно слишком много данных, а при расчёте каждую неделю слишком большое влияние оказывают факторы, не связанные с тестированием непосредственно — нерабочие дни, релизы новых версий и так далее. Кроме того, так как при перерасчёте по годам виден почти линейный рост, можно отметить, что за достаточно большой период набирается достаточно нового кода, содержащего новые ошибки (по этой же причине предположение о бесконечном количестве ошибок в коде не является существенным ограничением в данном случае). При расчёте же по неделям точность сильно понижается. Тем не менее, вопрос о более точном определении расчётного периода остается открытым, возможно, стоит найти формулу или алгоритм для определения оптимальной величины расчётного периода, если имеется достаточно большой объем статистики.

При этом также можно отметить, что полученные оценки достаточно точны в течение более длительного времени, чем один месяц, вплоть до 3-4 месяцев (рисунок 3).

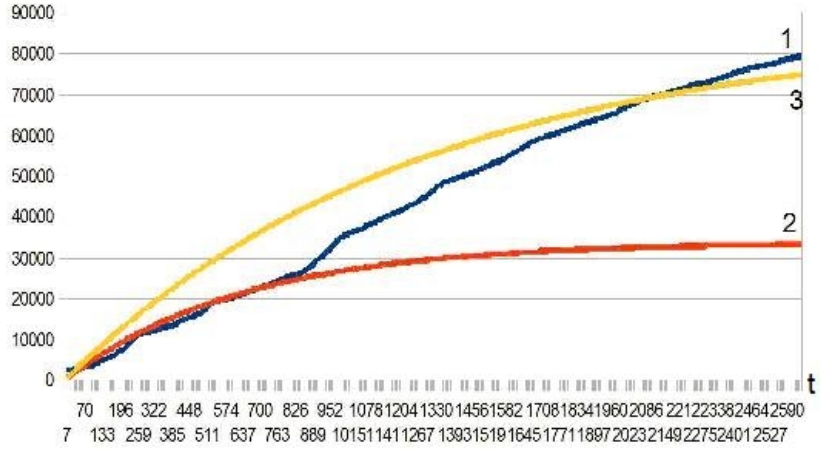

Рисунок 3: 1 – функция f(t), 2 – функция $m(t)_{100}$, *3 — функция* $m(t)_{300}$.

Видно также, что приемлемая точность аппроксимаций и оценок достигается примерно к 10-15 месяцу после начала тестирования, в начале же она очень низкая. Это связано, в первую очередь с особенностями самих моделей — до тех пор, пока тестирование не дойдет до того момента, на котором кривая начинает приближаться к асимптоте, очевидно, что стремление к точности аппроксимации будет снижать точность оценок. Кроме того, в случае с Eclipse у данного проекта есть особенность — в начале в Bugzilla, по всей видимости загрузили статистику за некоторый период, поставив близкие даты.

Что касается сравнения моделей между собой, то показатели моделей Goel-Okumoto, Jelinski-Moranda и Logarithmic в целом довольно близкие. Проведенные расчеты свидетельствуют, что спустя некоторое время (те же 10-15 месяцев) разница между оценками по этим моделям оказывается не выше 5%. Минусом GO модели является то, что она иногда не сходится — постоянно применять только ее не всегда возможно. Модель JM можно считать наиболее устойчивой, но не самой точной, кроме того она делает скорее заниженные оценки, тогда как из практических соображений более приемлемыми следует считать несколько завышенные.

Наиболее подходящей кривой для проектов с открытым исходным кодом, как можно было предположить, является S-изогнутая кривая, что хорошо видно на примерах статистики по отдельным модулям. Причиной этого является то, что у каждого небольшого

модуля имеется относительно небольшая команда разработчиков, которые другими модулями почти не занимаются. Как следствие, после обнаружения большого количества ошибок в начале качество тестирования начинает повышаться. В более глобальном масштабе, на уровне целого проекта уровня JDT или Platform (количество ошибок за весь период имеет порядок нескольких десятков тысяч) S-изогнутая модель работает плохо, что можно объяснить неравномерностью разработки по разным модулям.

5.5 Выводы

В результате проведенного исследования были сделаны следующие выводы:

- Лучше всего строить новые оценки каждый месяц.
- Высокая точность и устойчивость достигаются через 10-15 месяцев после начала тестирования.
- Наиболее подходящая МОН для проектов с открытым исходным кодом – S-Shaped, при этом ее лучше применять не к целым проектам, а к подпроектам.

6. Заключение

В рамках данной работы получены следующие результаты:

- Предложен алгоритм сравнения МОН.
- Реализовано программное средство для сравнения моделей и выбора наиболее подходящей.
- Показана общая эффективность МОН применительно к выбранным проектам с открытым кодом.
- Выбран наиболее подходящий промежуток времени для разбиения данных — один месяц.

Сформулированы следующие гипотезы о применении МОН к проектам с открытым исходным кодом:

- Приемлемая точность и устойчивость достигается через 10-15 месяцев после начала тестирования.
- Наиболее подходящей для проектов с открытым исходным кодом является S-изогнутая пуассоновская модель.

В качестве дальнейшего развития данного подхода можно выделить следующие направления:

- Автоматизация выбора наилучшего периода для группировки данных (с возможностью выбора произвольного периода, а не только «круглого»).
- Проверка указанных выше гипотез статистически, например с помощью критерия Пирсона. Для этого необходимо получить результаты для большего количества проектов.

- Определение периода, на котором оценки верны с точностью не ниже заданной.

Литература

1. M.Lyu Software Reliability Engineering: A Roadmap // 2007 Future of Software Engineering. Washington, DC, USA, 2007. p. 153 — 170.
2. Ширяев А. Н. Вероятность. М.: МЦНМО, 2004.
3. R.Al-Ekram Software Reliability Growth Modeling and Prediction. Waterloo: Dept. of Electrical and Computer Engineering, University of Waterloo, 2002
4. S.Alam Software Reliability Using Markov Chain Usage Model. Dhaka, Bangladesh: Department of Computer Science and engineering Bangladesh University of Engineering and Technology, 2005
5. A.Wood Software Reliability Growth Models. Technical Report, 1996
6. H.Umeda The development of a program drawing reliability growth curves and implementing reliability growth models. Tottori University of Natural Sciences, Department of Infromation Systems, 2008
7. M.Limnios, N.Nikulin Recent Advances in Reliability Theory. Birkhauser, 2000
8. C.Stringfellow An Empirical Method for Selecting Software Reliability Growth Models // Empirical Software Engineering 2002. 7. №4. p. 319 — 343.
9. Mozilla project. Bugzilla bug tracker http://www.bugzilla.org/about/
10. Eclipse project. http://bugs.eclipse.org/bugs/
11. Mozilla Project. https://bugzilla.mozilla.org/
12. Linux Kernel project. http://bugzilla.kernel.org/