МОСКОВСКИЙ
ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
им. М.В.ЛОМОНОСОВА

ФАКУЛЬТЕТ ВМиК
КАФЕДРА АСВК
ЛАБОРАТОРИЯ ВЫЧИСЛИТЕЛЬНЫХ КОМПЛЕКСОВ

Курсовая работа на тему:

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

Выполнил студент
322 группы
Зорин Д.А.

Научный руководитель
ассистент Волканов Д.Ю.

Москва,
Апрель 2009

Аннотация

Моделями оценки надежности (англ. Software reliability growth models) называются специальные математические модели, позволяющие по статистике тестирования за определенный период времени оценить количественные характеристики надежности проекта. Несмотря на то, что в литературе рассмотрено множество моделей оценки надёжности, критерии выбора этих моделей для конкретного проекта не определены четко. Соответственно, для применения моделей оценки надежности и получения адекватных результатов необходимо найти способ сравнивать модели между собой и выбирать наиболее подходящую.

Выбор моделей невозможен в отрыве от конкретной предметной области. В настоящее время широко распространена разработка программного обеспечения с открытым исходным кодом. Автору известно лишь две работы, посвященных проблеме выбора модели оценки надежности для таких проектов.

В данной работе рассматриваются различные модели оценки надежности, с целью определения применимости моделей к проектам с открытым исходным кодом и уточнения механизма их использования описываются эксперименты с некоторыми крупными проектами с открытым исходным кодом. В результате экспериментов получены различные характеристики надежности рассматриваемых проектов и сформулированы гипотезы об общих свойствах проектов с открытым исходным кодом.

Оглавление

1. Введение

В настоящее время в мире разрабатывается большое количество программных продуктов с открытым исходным кодом. В таких проектах участвует большое количество разработчиков, добавляющих свои фрагменты кода в общий репозиторий (хранилище, где находятся различные версии файлов, из которых состоит программа). В определенный момент возникает необходимость выпустить стабильную версию. При этом встает вопрос о том, как оценить насколько версия надежна, то есть узнать, с какой вероятностью программа работает в соответствии со спецификацией.

Наиболее распространенный способ поиска ошибок — тестирование. При этом тестировать программу можно как централизовано, силами определенной команды тестировщиков, так и распределенно, собирая так называемые «отчеты об ошибках» от разработчиков, независимых тестировщиков или конечных пользователей. Второй способ более удобен при разработке программного обеспечения с открытым исходным кодом. Основной недостаток тестирования — сколько бы программу ни тестировали на различных входных данных и в разных условиях, невозможно доказать формально, что ошибок больше нет. Поэтому возникает необходимость косвенно оценить различные формальные параметры системы, такие как количество оставшихся ошибок или надежность [8].

Под *надежностью* программного обеспечения понимают вероятность того, что, начиная с момента времени T, программная система будет работать по спецификации в течение заданного времени [14]. Часто рассматривают предел этой величины при бесконечно больших T. В дальнейшем мы будем рассматривать процесс разработки программы, и, соответственно, обнаружение ошибок в коде при тестировании.

Под «*ошибкой*» будем понимать работу программы не по спецификации, вызванную неверно написанным или отсутствующим кодом.

Так как измерить надежность непосредственно можно не во всех ситуациях, используются различные оценки, основанные, как правило, на данных о периодах между обнаружением ошибок.

Для оценки числа ошибок в системе, можно использовать две методики: первая основана на изучении исходного кода программы, вторая — использование моделей оценки надежности (англ. software reliability growth model, SRGM) — на данных, полученных при тестировании [8].

Суть SRGM заключается в том, чтобы подобрать модель, описывающую процесс обнаружения ошибок с помощью известных вероятностных распределений. Выбрав общий вид используемого распределения, 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.

Условно можно разделить все рассматриваемые далее модели на три типа:

1. Пуассоновские. Предполагается, что обнаружение ошибок можно описать моделью «Поток редких событий» [16], если предположить, что ошибки независимы, обнаруживаются по одной и известен характер изменения частоты их обнаружения.

2. Марковские. В марковских моделях используется предположение, что количество

оставшихся ошибок зависит от текущего состояния системы, и не зависит от прошлых состояний.

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 [1].

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

Данная классификация основана на разделении моделей по используемому базовому математическому аппарату, в литературе встречается также другая классификация, основанная на некоторых используемых в моделях ограничениях. Подробнее эта классификация описана в [3].

В разных моделях используется, вообще говоря, разный набор ограничений [6]. Приведенные ниже — наиболее часто встречающиеся.

а) Дефекты исправляются сразу после обнаружения

б) При исправлении новых ошибок не появляется

в) В период тестирования не появляется нового кода

г) Тестирование централизовано, то есть в каждый момент одна версия программы тестируется одной группой тестирощиков.

д) Ошибки не зависят друг от друга.

Все эти предположения, конечно, на практике, в точности не выполняются, поэтому одной из актуальных задач является создание моделей, позволяющих либо избавиться от каких-то из указанных ограничений, либо сделать его влияние на результат меньше.

После выбора происходит расчет: по исходным данным (как правило, даты обнаружения ошибок) с помощью методов математической статистики выводится оценка необходимых характеристик: общего количества ошибок, времени до ближайшей ошибки.

В результате применения моделей оценки надежности возможно получить следующие характеристики процесса тестирования [18]:

- *Функция количества ошибок* или *функция количества (*англ. mean function) — неубывающая функция одной вещественной неотрицательной переменной t, показывающая количество ошибок, найденных к моменту времени t.
- *Интенсивность потока ошибок* или просто *интенсивность*[1] *(*англ. Failure intensity*)* — производная функции количества по t, показывающая среднее количество ошибок, обнаруживаемых в данное время.
- *Ожидаемое время до ошибки* (англ. Mean time to failure) — оценка времени с текущего момента t0 до момента t1>t0, когда функция количества m(t) примет ближайшее к m(t0) целое значение. Эта характеристика особенно важна в моделях, не позволяющих оценить общее количество ошибок.

Применение моделей оценки надежности к проприетарному программному обеспечению достаточно хорошо исследовано (например, в работах [2], [13], [19]). Проприетарное программное обеспечение в большей степени, чем открытое, удовлетворяет указанным выше ограничениям, так как команда разработчиков и

1 Не следует путать интенсивность потока ошибок с интенсивностью пуассоновского потока, которая есть математическое ожидание случайной величины, имеющей распределение Пуассона, а в рамках пуассоновской модели по сути эквивалентна функции количества.

тестировщиков централизованная, и поэтому процесс разработки и тестирования является более равномерным. Кроме того, модели оценки надежности изначально создавались без учета возможности применения к проектам с открытым исходным кодом, так как в то время (70-е годы) современный способ разработки проектов с открытым кодом (Open Source) еще не существовал. Поэтому применимость моделей оценки надежности к открытому программному обеспечению требует дополнительного исследования.

Структура работы такова: в главе 2 формулируется цель работы, в главе 3 приводится неформальное описание предметной области. Далее в главе 4 будут рассмотрены различные часто используемые модели оценки надежности и некоторые подробности их реализации и применения. В главе 5 описан формальный алгоритм выбора наилучшей модели оценки надежности на основе наблюдений за процессом тестирования в течение некоторого периода времени. В главе 6 приведено описание программы, используемой для расчетов. В главе 7 описаны результаты применения методик SRGM к реальным проектам программного обеспечения с открытым исходным кодом.

2. Цель работы

Цель работы - оценить применимость моделей оценки надёжности для проектов по разработке программного обеспечения с открытым исходным кодом, разработать и реализовать средство, позволяющее оценить количественные характеристики надёжности разрабатываемого ПО и оценить наиболее приемлемый период времени, по которому следует группировать данные.

Для достижения цели необходимо решить следующие задачи:

I. Написать обзор предметной области. Для этого необходимо:

1. Изучить общий математический аппарат, применяющийся при исследованиях, связанных с моделями оценки надежности.
2. Рассмотреть различные существующие модели, обращая внимание на формальную математическую постановку, данные, получающиеся в результате расчетов, ограничения, которым должен соответствовать проект, чтобы модель была применима и преимущества/недостатки перед другими моделями.

II. Подготовиться к практическому исследованию:

1. Выбрать из рассмотренных моделей несколько по сформулированным в обзоре критериям.
2. Сформулировать требования для статистики по проекту.
3. Описать алгоритм выбора наиболее подходящей модели оценки надежности.
4. Выбрать проекты для анализа.

III. Разработать программное средство с необходимой для исследования функциональностью, удовлетворяющее следующим требованиям:

1. Позволяет рассчитывать оценки по выбранным моделям.
2. Позволяет рассчитывать точность моделей и выбирать наилучшую.
3. Выводит результаты в наглядном виде (график, XML).
4. Содержит отдельный модуль, позволяющий загружать статистику из средства отслеживания ошибок Bugzilla.
5. Является переносимым на различные платформы (Windows, Linux, Mac OS)

IV. Провести экспериментальное исследование:

1. Загрузить статистику проектов с открытым исходным кодом, использующих средство отслеживания ошибок Bugzilla
2. Примененить реализованные модели на практике.
3. Сравнить модели на примере выбранного проекта и сделать вывод о применимости той или иной модели в рассматриваемой ситуации.
4. Выбрать период, по которому следует группировать данные.
5. Сформулировать гипотезы о том, какие модели подходят лучше.

3. Неформальное описание предметной области

Прежде, чем перейти к формальному математическому описанию моделей оценки надежности, необходимо уточнить, как происходит разработка программного обеспечения с открытым исходным кодом и какие задачи, стоящие перед руководителем проекта, решаются с использованием моделей оценки надежности.

Проектом будем считать одну или несколько программ, созданием которых управляет одна команда, а текущие разработки хранятся в общем репозитории. Большой проект может состоять из подпроектов, реализующих отдельные части, работающие независимо друг от друга. Подпроекты могут состоять из компонент, которые выполняют определенный набор функций, и не используются отдельно. В проекте участвуют разработчики, работающие над одним или несколькими подпроектами, тестировщики, а также существует некое общее управление. Одна из основных задач руководителя проекта — решить, когда программу можно предоставлять конечным пользователям. В случае проприетарного программного обеспечения, это обычно называют релизом, для проектов с открытым исходным кодом — стабильной версией, которую могут соответствующим образом оформить и распространять. Таким образом, необходимо оценить количество ошибок, оставшихся в программе. Для этого можно анализировать либо сами исходные коды, либо статистику тестирования, которая в случае проектов с открытым исходным кодом составляется, как правило, силами большого количества не связанных между собой тестировщиков либо пользователей промежуточных версий, и представляет собой данные о баг-репортах, которые могут быть упорядочены с помощью специальных средств отслеживания ошибок, таких, как например, Bugzilla. С его помощью собираются исходные данные для моделей оценки надежности: информация обо всех ошибках, их времени, критичности, компоненте.

В результате, перед руководителями проекта стоят следующие задачи:

- Оценить количество ошибок, оставшихся в проекте, и их критичность.
- Оценить количество ошибок, оставшихся в выбранной компоненте, и их критичность.
- Оценить время, когда текущач версия станет стабильной, то есть количество ошибок и их критичность будут меньше заданного порога.

Третий вопрос может быть весьма существенным, так как разработка различных компонент может вестись по-разному и, соответственно, описываться разными моделями.

Использование моделей оценки надежности позволяет получить ответ на эти вопросы. На основе исходных данных строится функция количества ошибок, определенная в главе 1. При получении этой функции возникают следующие задачи:

1. Определить период времени для уточнения оценок количества ошибок в проекте.
2. Выбрать наиболее подходящую к данному проекту модель оценки надежности из нескольких возможных.
3. Оценить, какой объем исходных данных необходим для того, чтобы модели давали результаты с точностью не ниже заданной.

Эти задачи касаются внутреннего устройства метода моделей оценки надежности, поэтому желательно, чтобы при практическом применении моделей либо уже был ответ на них, либо ответ получался в результате некой формальной процедуры, не требующей дополнительного исследования.

Данная работа посвящена решению задач 1 - 3 для некоторых конкретных проектов.

В следующем разделе подробно рассматриваются различные модели оценки надежности.

4. Краткий обзор моделей оценки надежности

Суть моделей оценки надежности 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 определенный 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.

Примером самой тривиальной модели оценки надежности может служить модель Nelson [17].

$$P(t)=1-\frac{n}{N}$$

Здесь *n* — число успешных «испытаний», а *N* — общее число «испытаний», *P(t)* — вероятность появления ошибки на интервале *(0, t)*. Эта модель, предложенная в 1970-е годы, была одной из первых, и сейчас практически не используется, так как дает только точечную оценку по минимальному количеству данных, и как следствие, очень низкую точность. Модели, описанные далее в этом разделе, намного более сложные, но позволяют получить больше результатов с лучшей точностью.

В данном обзоре рассмотрены три основных типа моделей оценки надежности: пуассоновские, марковские и байесовские. Для программной реализации были выбраны не все модели, а наиболее распространенные (подробно описанные в литературе, ка часто применявшиеся на практике) модели различных типов.

Далее будут описаны способы оценки параметров моделей, рассмотрены отдельные модели различных типов и проведено их сравнение.

4.1. Статистическая оценка параметров

После выбора модели, исходя из каких бы то ни было рассуждений, необходимо вычислить параметры, так как модель задает лишь вид функции, зависящей от нескольких (1 или 2) произвольных постоянных, подлежащих определению. Чтобы найти эти постоянные, используются стандартные статистические методы [13]:

а) Метод максимального правдоподобия

На основе выборки имеющихся наблюдений можно построить функцию правдоподобия, являющуюся произведением плотностей отдельных случайных величин из выборки (предполагается, что тип распределения известен). По сути функция правдоподобия представляет из себя вероятность получить заданную выборку при заданном значении параметра. Соответственно, если найти максимум этой функции, то это значение параметра можно взять в качестве точечной оценки, так как при таком значении параметра вероятность получить имеющуюся выборку максимальна. На

практике, при расчете параметров в пуассоновских моделях, задача сводится не к поиску максимума функции, а к уравнению, которое можно решить численно. Это связано с дискретностью наблюдений.

б) Линейная регрессия

Суть: найти такие значение параметров, при которых полученная кривая будет наиболее близка к наблюдениям. За меру близости принимается сумма квадратов расстояний.

Основываясь на полученных точечных оценках, далее можно построить доверительные интервалы для искомых параметров. При этом метод максимального правдоподобия имеет некоторое преимущество, так как в нем на оценки накладывается меньше ограничений, связанных с предположением о том, что ошибки нормально распределены (а на больших выборках это становится малосущественным в силу центральной предельной теоремы), кроме того, при расчетах вторым методом могут получатся результаты, противоречащие реальности, например, нижняя граница доверительного интервала может оказаться меньше текущего числа ошибок.

Помимо доверительных интервалов, при практическом применении SRGM часто используют факты из теории проверки статистических гипотез, так как таким образом проще проверять соответствие полученной кривой и реальных данных. Самый простой подход — расчет среднеквадратичного отклонения, при его применении возникает проблема выбора порога, начиная с которого гипотеза отвергается.

Более сложный способ — использование критериев согласия. Опишем кратко один из самых распространенных критериев — критерий Пирсона (хи-квадрат).

Обозначим через X исследуемую случайную величину. Пусть требуется проверить гипотезу о том, что эта случайная величина подчиняется закону распределения $F(x)$. Для проверки гипотезы произведём выборку, состоящую из n независимых наблюдений над случайной величиной X. По выборке можно построить эмпирическое распределение $F^*(x)$ исследуемой случайной величины. Сравнение эмпирического $F^*(x)$ и теоретического распределений производится с помощью специально подобранной случайной величины — критерия согласия.

Для проверки критерия вводится статистика:

$\chi^2 = N \sum \frac{(P_i^{theor} - P_i^{emp})^2}{P_i^{theor}}$, где $P_i^{theor} = \int_{x_{i-1}}^{x_i} F(x)\,dx$ — предполагаемая вероятность попадания в i-й интервал, $P_i^{emp} = \frac{n_i}{N}$ — соответствующее эмпирическое значение.

Эта величина в свою очередь является случайной(в силу случайности X) и должна подчиняться распределению хи-квадрат.

Если полученная статистика превосходит квантиль закона распределения χ^2 заданного уровня значимости α с k — 1 или с k — p - 1 степенями свободы[2], где k — число наблюдений или число интервалов (для случая интервального вариационного ряда), а p — число оцениваемых параметров закона распределения то гипотеза отвергается. В противном случае гипотеза принимается на заданном уровне значимости α .

2 Распределение хи-квадрат с n степенями свободы - это распределение суммы n независимых стандартных нормальных случайных величин. Подробности о свойствах этих распределений - [16]

При реализации моделей оценки надежности в средстве, описанном в разделе 5 данной работы, использовался метод максимального правдоподобия как наиболее универсальный и не сложный в реализации.

4.2. Пуассоновские модели.

Пуассоновский процесс — семейство случайных величин, заданных на одном вероятностном пространстве, зависящих от неотрицательного параметра t, обладающее следующими свойствами: 1) Независимые приращения: случайные величины $\xi_0, \xi_1-\xi_0, \xi_2-\xi_1 \cdots$ независимы 2) Однородность по времени 3) $\xi_0=0$ 4) P(ξ > 1) стремится к нулю при h->0

Теорема. Если X — пуассоновский процесс, то для любого t случайная величина ξ_t имеет распределение Пуассона с параметром l*t

В SRGM рассматривается неоднородный процесс, где интенсивность потока ошибок в заданный момент времени не является линейной функцией от t. По виду функции количества *m(t)* модели делят на :

а) Выпуклые (Concave). Предполагаются, что количество обнаруживаемых ошибок уменьшается равномерно с ростом объема тестирования (и количества исправленных ошибок)

б) S-изогнутые. Предполагается, что тестирование на ранних этапах разработки менее эффективно, например, потому что в начале разработки одни ошибки влекут за собой другие, и их исправление возможно только совместно, а также потому, что часть времени уходит на исправление параметров системы для корректной работы программы.

в) Неограниченные (Infinite failure). Используется предположение, что количество ошибок в программе бесконечно (и соответствующее распределение). Несмотря на кажущуюся абсурдность такой модели, в реальности есть основания их использовать, если считать, что все ошибки исправить к одному релизу невозможно.

Основные выпуклые модели:

- Экспоненциальная (п. 3.2.1)
- Gompetz (п. 3.2.2)
- Вейбулла (п.3.2.3)
- Парето (п. 3.2.7)
- Гиперэкспоненциальная (п.3.2.7)

Основные S-изогнутые модели:

- Yamada (п. 3.2.4)
- Параметризованная (п. 3.2.5)

Самая распространенная неограниченная модель — логарифмическая (п. 3.2.8).

4.2.1. Экспоненциальная модель (Goel-Okumoto)

Формула $m(t)=a(1-e^{-bt})$ задает функцию ошибок. Параметр *a* есть общее количество ошибок, параметр *b* задает скорость обнаружения ошибок.

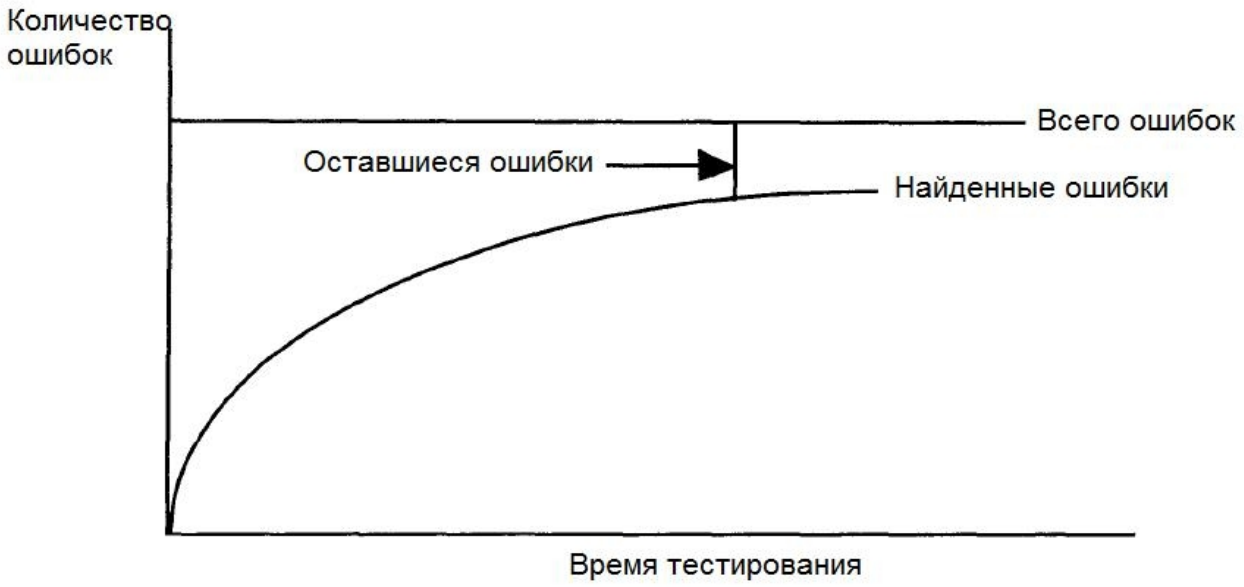

Рисунок 1: Экспоненциальная модель: график m(t)

Суть [4]: полагаем, что число ошибок, обнаруженных в интервале *dt*, пропорционально числу оставшихся ошибок, считая, что общее число ошибок конечно и равно a. Получаем дифференциальное уравнение

$$m'(t)=ab-bm(t)$$

Решением этого уравнения является указанная выше функция *m(t)*.

Вид выпуклой кривой показан на рис.1

Эта модель очень распространена. Оценка параметров методом максимального правдоподобия в этом случае равносильна решению уравнения [3]:

$$\sum_{i=1}^{w}\left((f_i-f_{i-1})\frac{t_i e^{-bt_i}-t_{i-1}e^{-bt_{i-1}}}{e^{-bt_i}-e^{-bt_{i-1}}}\right)=\frac{f_w t_w}{1-e^{bt_w}}$$

Где *w* — количество тестов (периодов тестирования), *t* — время тестирования к концу данного периода, *f* — количество ошибок, обнаруженных к концу данного периода.

4.2.2. Модель Gompertz

$$m(t)=ak^{b^t}.$$

В данной модели кривая также является выпуклой. Относительным преимуществом является большая гибкость модели, так как она зависит от трех параметров. Для этой модели также есть модификация, делающая кривую S-изогнутой [14].

4.2.3. Обобщенная экспоненциальная модель (модель Вейбулла)

$$m(t)=a(1-e^{-bt^k})$$

Параметр k показывает качество тестирования. При k=1 получаем экспоненциальную модель. По сути, это обобщение экспоненциальной модели с целью сделать кривую более «настраиваемой» на данные.

4.2.4. Модель Yamada (S-Shaped)

Еще одно обобщение экспоненциальной модели, суть модификации заключается в том, что кривая становится S-изогнутой, и происходящее можно рассматривать как модель для тестирования, при котором умения тестировщика постепенно увеличиваются. Вид кривой показан на рис. 2.

$$m(t)=a(1-(1+bt)e^{-bt})$$

Параметры a и b находятся из системы уравнений [3]:

$$\sum_{i=1}^{n} f_j = a(1-(1+bt_n)e^{-bt_n})$$

$$at_n^2 e^{-bt_n} = \sum_{i=1}^{n} \left(\frac{(\sum_{j=1}^{i} f_j - \sum_{j=1}^{i-1} f_j)\cdot(t_i^2 e^{-bt_i} - t_{i-1}e^{-bt_{i-1}})}{((1+bt_{i-1})e^{-bt_{i-1}} - (1+bt_i)e^{-bt_i})}\right)$$

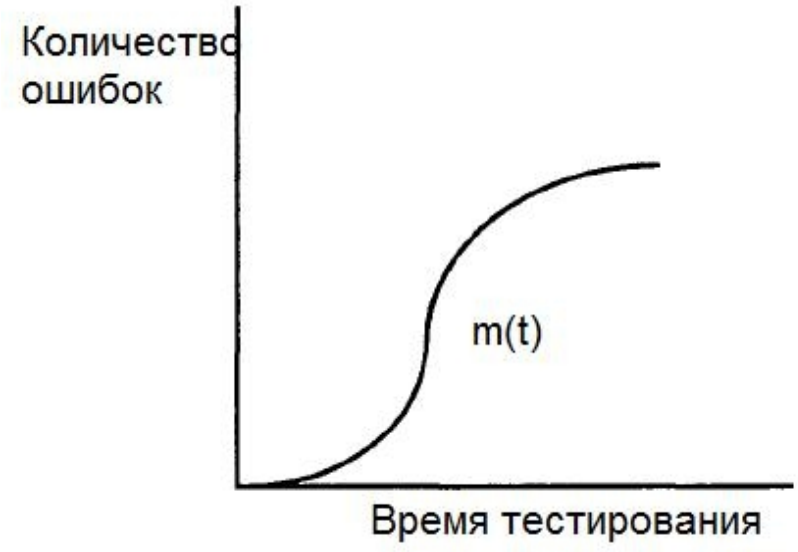

Рисунок 2: Вид графика m(t) для S-изогнутых моделей

4.2.5. Параметризованная S-изогнутая модель

Данная модель позволяет избавиться от ограничения на зависимость ошибок друг от друга.

Формула:

$$m(t)=a\frac{1-e^{-bt}}{1+\psi(r)e^{-bt}}$$

Здесь $\psi(r)=\frac{1-r}{r}$. Параметр *r* есть отношение количества ошибок, которые можно обнаружить в текущий момент (то есть не зависящих от еще не найденных), к общему количеству ошибок. При *r=1* получаем стандартную экспоненциальную модель.

4.2.6. Модель Парето.

В этой модели считается, что ошибки неравноценны, и наиболее серьезные исправляются первыми. Параметр *a*, как и раньше, показывает общее количество ошибок, два других параметра задают форму кривой.

$$m(t)=a\left(1-(1+t/\beta)^{1-\alpha}\right)$$
$$a\geq 0\,;\,\beta>0\,;\,0\leq\alpha\leq 1$$

4.2.7. Модель с двумя типами ошибок (гиперэкспоненциальная).

Если ошибки неравноценны, то можно разделить их условно на две группы — легкие и трудные для обнаружения, и оценивать отдельно. Для конкретного проекта «легкие» и «трудные» в таком случае необходимо определять отдельно. Уравнение выглядит следующим образом:

$$m(t)=a\left(p_1(1-e^{-b_1 t})+p_2(1-e^{-b_2 t})\right)$$
$$a>0, 0<b_2<b_1<1,\ p_1+p_2=1, 0<p_{1/2}<1$$

Здесь p_1, p_2 - «веса» групп ошибок. Эту модель можно обобщить, чтобы использовать не только для двух типов ошибок, но и для большего количества.

4.2.8. Логарифмическая пуассоновская модель

$$m(t)=b_0\ln(b_1 t+1)\,;\,b_0>0\,;\,b_1>0$$

Параметры находятся из следующей системы:

$$b_0=\frac{n}{\ln(1+b_1 t_n)}$$
$$\frac{1}{b_1}\sum_{i=1}^{n}\frac{1}{1+b_1 t_i}=\frac{nt_n}{(1+b_1 t_n)\ln(1+b_1 t_n)}$$

Функция количества не ограничена на бесконечности. Данная модель не позволяет получить оценку для общего количества ошибок. На практике, если это необходимо, можно произвести расчет по другой модели (например, экспоненциальной) и найти, сколько ошибок будет обнаружено по логарифмической модели к моменту, когда по экспоненциальной модели будет найден определенный процент ошибок, либо искать значение функции количества в момент времени, когда интенсивность становится меньше некоторого достаточно малого значения. [9]

В заключение отметим некоторые достоинства и недостатки пуассоновских моделей.

(+) По пуассоновским моделям много работ, их применяли в разных проектах множество раз, и получали адекватные результаты [13]

(+) Возможность исследовать несколько параллельных независимых источников ошибок (например, ошибки программы и оборудования) в силу простоты сложения двух пуассоновских случайных величин

(-) Изначально, пуассоновские процессы использовались для моделирования отказов оборудования. При тестировании программ время, будь то астрономическое время запуска тестов или процессорное время, дискретно, и в этом отношении пуассоновские модели применяются с ограничениями. В частности, может получиться так, что оценка для Mean time to failure не несет содержательного смысла из-за того, что процесс тестирования не является непрерывным. [7].

4.3. Марковские модели

Случайный процесс называется цепью Маркова, если его состояние в будущем зависит только от настоящего, и не зависит от предыдущих состояний.

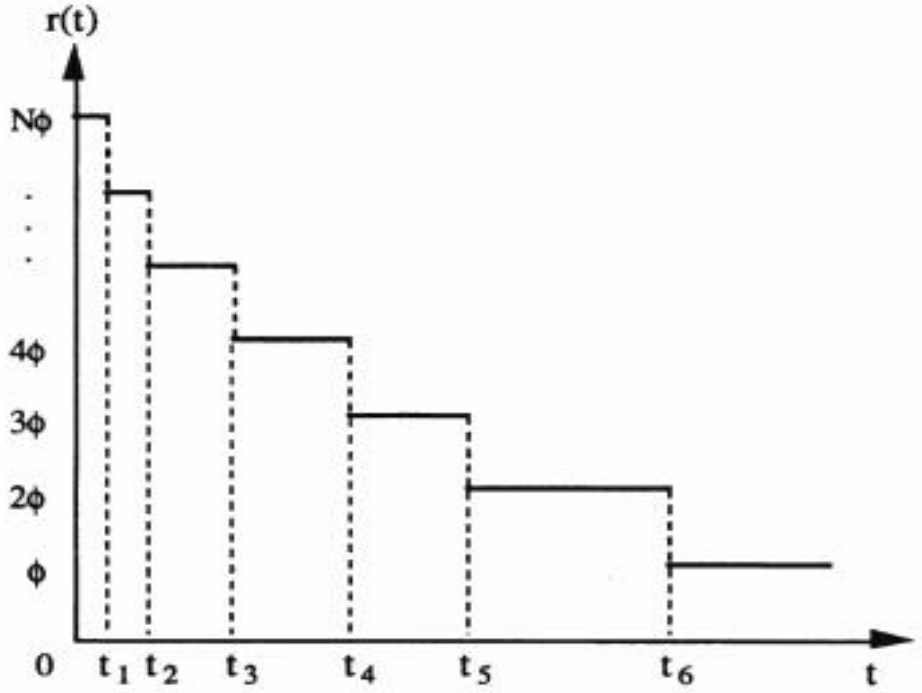

Рисунок 3: Зависимость интенсивности от времени в JM-модели

Модель Jelinski-Moranda рассматривает процесс тестирования как марковскую цепь, с неизвестным начальным состоянием. Полагая, что надежность линейно зависит от количества ошибок, можно рассматривать время между обнаружением двух ошибок случайной величиной с экспоненциальным распределением.

В этом отношении данная модель похожа на пуассоновскую (в обеих присутствует поток редких событий), однако, используя JM-модель, мы смотрим на происходящее с другой стороны: если в пуассоновских моделях рассматривались вероятности появления того или иного количества ошибок к заданному моменту времени, то в марковских моделях ищутся вероятности появления очередной ошибки за некоторый момент времени, то есть, если в пуассоновской модели функция, описывающая «надежность» была непрерывна, а сама модель описывала дискретные вероятности, то

в марковских «надежность» дискретна, но оценивается время, являющееся непрерывным [2].

Пусть в программе *a* ошибок, в начальный момент интенсивность равна $a*b$, где *b* – положительный параметр. T_i – время между появлением *i*-й и *i-1* ошибок имеет показательное распределение с параметром интенсивности $(a-i+1)b$. Параметры *a* и *b* можно оценить методом максимального правдоподобия:

$$b=\frac{n}{a\sum_{i=1}^{n}(t_i-t_{i-1})-\sum_{i=1}^{n}(i-1)(t_i-ti-1)}$$

$$\frac{\sum_{i=1}^{n}1}{a-i+1}=\frac{n}{a-\frac{\sum_{i=1}^{n}(i-1)(t_i-t_{i-1})}{\sum_{i=1}^{n}(t_i-t_{i-1})}}$$

JM-модель имеет много ограничений, в частности предполагается, что ошибки исправляются моментально и не зависят друг от друга. Наиболее серьезным ограничением является предположение о линейной зависимости надежности и числа обнаруженных ошибок, или, что то же самое, предположение об одинаковом «весе» всех ошибок. Чтобы избавиться от этого ограничения, вводится функция *λ(i)*, описывающая частоту нахождения ошибок. *λ(i)* является монотонно убывающей функцией: т.к. математическое ожидание экспоненциально распределенной случайной величины равно *1/λ*, то получается, что со временем отказы происходят реже.

Одно из возможных решений — взять в качестве *λ(i)* степенную функцию [14]:

$$\lambda(i)=b(a-i+1)^k$$

Так как *λ* в начале должна резко убывать, *k>1*.

Более сложный вариант — экспоненциальный рост *λ*:

$$\lambda(i)=b(e^{-k(a-i+1)}-1)$$

Другой подход предложен [11] — можно ввести функцию φ(t), описывающую изменения в характере тестирования на протяжении некоторого времени. Тогда

$$\lambda(i)=\phi(t)(a-i+1)$$

Если *φ(t)* — константа, получаем обычную JM-модель.

В заключение — о преимуществах и недостатках марковских моделей.

(+) Так как вид кривой в марковских моделях достаточно простой (кусочно-постоянная функция), то основным преимуществом марковских моделей является простота расчета.

(-) Наличие априорных предположений о «весе» дефектов.

(-) Низкая точность аппроксимации при небольшом количестве тестовых данных (по сравнению, например, с пуассоновскими моделями).

По этим причинам часто марковские модели используют не в чистом виде, а с байесовскими модификациями [20].

4.4. Байесовские модели

В теории вероятностей доказывается формула Байеса: $P(H_k|A)=\frac{P(H_k)P(A|H_k)}{\sum_i P(H_i)P(A|H_i)}$, позволяющая вычислить вероятности гипотез, основываясь на результатах наблюдений.

Вообще, *байесовскими* называют различные модели, учитывающие новые данные для корректировки существующих гипотез. Если в описанных выше моделях используются некоторые предположения о характере распределения априорно, то при использовании байесовских методов эти параметры меняются в зависимости от новых измерений. К примеру, если с какого-то момента ошибки резко участятся, то и распределения, описывающие предположения о надежности программы, изменятся таким образом, что вероятность новых ошибок повысится.

Формализуем этот подход. Пусть в выбранной модели есть несколько параметров, которые необходимо оценить, и они задаются в виде вектора X из некоторого линейного пространства. Если согласно предыдущим измерениям вектор X имеет плотность распределения g(X), то байесовская оценка будет иметь следующий вид:

$$h(X|t)=\frac{f(t|X)g(X)}{\int_\Omega f(t|X)g(X)dX},\ X\in\Omega$$

Здесь t — некоторый вектор, содержащий новые данные, f(t|X) — функция правдоподобия, h(X|t) — апостериорная оценка плотности распределения вектора X.

В общем случае получить аналитически формулу для апостериорного распределения невозможно, вместо этого применяются численные алгоритмы расчета кратных интегралов. Однако для некоторых распределений, обладающих некоторыми удобными свойствами, это возможно, и именно они используются в простых моделях. Таковы, в частности, Гамма и Бета распределения.

Свойства байесовских моделей:

(+) Любую SRGM теоретически можно модифицировать и сделать байесовской, достаточно для каждого параметра ввести функцию априорного распределения [3]. Для соблюдения логичности и корректности модели, тем не менее, необходимо осознавать содержательный смысл выбираемого распределения.

(-) Введение байесовских оценок усложняет процесс расчета параметров.

(-) Трудности с применением на больших объемах данных.

Рассмотрим отдельно наиболее популярную байесовскую модель.

Модель LV (Littlewood — Verrall)

Модель LV является одной из самых распространенных байесовской моделью. Изначально предполагаем, что время между ошибками имеет экспоненциальное распределение:

$$f(t_i|\lambda_i)=\lambda_i\exp-\lambda_i t_i$$

Однако теперь функция ошибок является случайной величиной, имеющей Гамма-распределение:

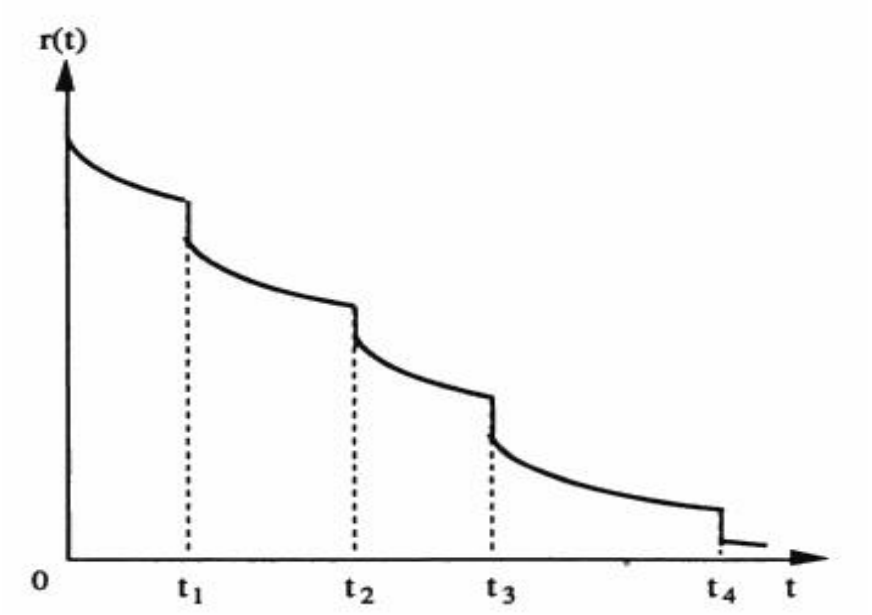

Рисунок 4: Интенсивность в LV-модели в зависимости от времени

$$f_{\lambda_i}(\alpha,\psi(i))=\frac{[\psi(i)]^{\alpha}\lambda_i^{\alpha-1}e^{-\psi(i)\lambda_i}}{\Gamma(\alpha)}$$

Параметр α характеризует форму кривой, а *ψ(i)* — параметр масштаба, он зависит от количества найденных ошибок. Чаще всего считают, что *ψ* описывает качество тестирования, и, соответственно, монотонно возрастает. Из этого следует, что *λ(i)* убывает стохастически, то есть

$$P(\lambda_i \le \lambda) \ge P(\lambda_{i-1} \le \lambda)$$

В частности, можно взять в качестве *ψ(i)* линейную функцию, тогда коэффициенты прямой — два параметра для оценки.

Интенсивность в LV модели равна:

$$\lambda(t)=\frac{\alpha-1}{\sqrt{b_0^2+2b_1(\alpha-1)t}}$$

Три неизвестных параметра находятся из системы трех уравнений:

$$\frac{n}{\alpha}+\sum_{i=1}^{n}\ln(b_0+b_1 i)-\sum_{i=1}^{n}\ln(t_i-t_{i-1}+b_0+b_1 i)=0$$

$$\alpha\sum_{i=1}^{n}\frac{1}{b_0+b_1 i}-(\alpha+1)\sum_{i=1}^{n}\frac{1}{t_i-t_{i-1}+b_0+b_1 i}=0$$

$$\alpha\sum_{i=1}^{n}\frac{i}{b_0+b1\,i}-(\alpha+1)\sum_{i=1}^{n}\frac{i}{t_i-t_{i-1}+b_0+b_1 i}=0$$

4.5 Ограничения на применение моделей оценки надежности. Сравнение моделей

В разделе 1 были отмечены некоторые наиболее часто встречающиеся ограничения на применение SRGM. В этом пункте данные ограничения будут систематизированы и формализованы.

Ниже охарактеризованы наиболее часто встречающиеся ограничения. [10]

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

b. Однородность: все ошибки (по крайней мере, в рамках анализируемой группы) равноценны и вероятность их обнаружения одинакова.

c. Независимость: ошибки не зависят друг от друга

d. Идеальность: после обнаружения ошибка немедленно исправляется, и при этом новых ошибок не появляется.

e. Бесконечность: общее число ошибок считается бесконечным, нет возможности получить его оценку напрямую.

Модель	Равно-мерность	Однородность	Независимость	Идеальность	Бесконечность
Экспоненциальная (Пуассоновская)	-	-	-	-	+
Jelinski-Moranda (Марковская)	-	-	-	-	+
Littlewood-Verrall (Байесовская)	-	+	-	+	-
S-Shaped (Пуассоновская)	-	-	-	-	+
Логарифмическая (Пуассоновская)	-	-	-	-	-

Таблица 1: Сравнение моделей оценки надежности

В таблице 1 приведены характеристики пяти распространенных моделей, которые реализованы и использованы при исследовании (п. 5-6). Знак «-» в соответствующей графе указывает на то, что данное ограничение действует при применении данной модели.

Кроме того, каждая модель использует базовые предположения о виде вероятностного распределения случайных величин и самого случайного процесса, которые используются в модели. В случае использования пуассоновского или показательного распределения эти ограничения являются следствиями вышеописанных.

Из таблицы видно, что меньше всего ограничений накладывает модель LV, поэтому можно ожидать, что результаты по ней будут наиболее точными. Экспоненциальная, S-изогнутая и Jelinski-Moranda модели в плане ограничений эквивалентны, поэтому точность их зависит от конкретного проекта. Чтобы можно было решить, какую из

моделей выбрать, необходимо иметь какие-либо критерии отбора. Например, можно по прошествии некоторого времени сравнивать теоретические оценки с реальными данными. В следующем разделе этот подход будет развит, и будет предложен формальный алгоритм выбора модели.

5.Алгоритм выбора модели оценки надежности

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. Систематический подход к выбору модели предполагает сравнение результатов применения различных моделей в течение тестирования для выбора наиболее подходящей, и, возможно, для определения момента, когда можно оставить лишь одну лучшую.

Перечислим основные факторы, из-за которых SRGM 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, перечисленные в пункте 3.5

Ни одна модель не подходит идеально, поэтому нужно как-то оценивать точность, сходимость и так далее.

Следует отметить, что на практике при применении SRGM возникает необходимость привязывать результаты ко времени, так как, в отличие от теории, время тестирования не бесконечно и некоторым образом связано с внешними факторами (длина рабочей недели, близость срока сдачи проекта и прочее). Идеальный вариант — учитывать только процессорное время, то есть время, в течение которого непосредственно проходило тестирование. Изначально, когда SRGM использовались для прогнозирования отказов оборудования, такой подход был вполне реалистичен, однако, когда речь идет о программном обеспечении, собрать информацию о процессорном времени малореально. Поэтому, как правило, используется календарное время, и данные группируются по неделям (или месяцам).

При применении SRGM в проприетарных проектах (например, в [13]) такой подход полностью оправдан, так как время тестирования в течение каждого периода одинаковое — команда тестировщиков работает в установленные часы, меняться может только эффективность их работы, что допускается многими моделями.

В случае же программного обеспечения с открытым исходным кодом ситуация совершенно другая [20]:

- Команда тестировщиков непостоянна и работает не в установленные часы, а по мере своих возможностей. Отчеты 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.
- В течение тестирования продолжается разработка, поэтому количество ошибок может увеличиться.

Далее рассмотрим относительно простой подход, позволяющий сравнивать несколько заданных SRGM [12].

1. На первом шаге рассчитываются параметры для нескольких моделей (например, методом нелинейной регрессии). 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, описанной в разделе 5, рассчитывается среднеквадратичное отклонение, так как этот способ прост и универсален.

3[3]. Проверка на устойчивость. Под устойчивостью оценки будем понимать то, что оценка за текущий период отличается от оценки за прошлый период не более, чем на k %, где k не очень велико, например, равно 10.

4. Если на предыдущем шаге не оказалось ни одной устойчивой модели, то необходимо продолжить тестирование и сбор данных еще на один отчетный 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 расчеты, 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. В следующем разделе описано программное средство, позволяющее проводить такие исследования.

3 Между 2 и 3 пунктом возможна проверка, не получилось ли так, что оценка общего количества ошибок меньше числа реально найденных к текущему моменту ошибок. Если это так, то модель также отвергается исходя из соображения о том, что недооценить число ошибок намного хуже, чем переоценить его [12]. Поскольку при реализованном в программе способе расчета параметров такая ситуация не возникает, этот пункт опущен.

6. Описание программной реализации

6.1 Общая характеристика

Программа работает с «проектами», список которых хранится в конфигурационном файле. Каждый проект имеет свой список используемых моделей, задаваемый пользователем, и к каждому проекту при создании должен быть приложен XML-файл с исходными данными. Формат входного файла следующий:

```
<errors>
<error>
<time>[int]</time>
<programmer>[int]</programmer>
<severity>[int]</severity>
<item>[int]</item>
</error>
<error>...</error>
</errors>
```

Здесь:

- time — время фиксации ошибки
- programmer — идентификатор программиста, реализовавшего фрагмент, содержащий ошибку
- severity — условная фатальность ошибки. В проекте Eclipse, например, введены следующие уровни серьезности ошибки:
  - Blocker (препятствует дальнейшей разработке).
  - Critical (серьезные ошибки, приводящие к падению программы).
  - Major (серьезная ошибка).
  - Minor (малосущественная ошибка).
  - Trivial (мелкое изменение, в основном в оформлении).
  - Enhancement (запрос на разработку).
  - Normal (значение по умолчанию).
- item — модуль, где обнаружена ошибка.

Все атрибуты, кроме time, необязательны. Реализована возможность добавлять данные, как по одному пункту через специальную форму, так и загрузив новый XML-файл, который объединяется с первым.

Также существует возможность фильтровать данные по указанным выше характеристикам (программист, фатальность, модуль)

По полученным данным программа рассчитывает оценку общего количества ошибок и времени до ближайшей ошибки по следующим моделям, описанным в разделе 3:

- Goel-Okumoto (Экспоненциальная)
- S-Shaped (Yamada, S-изогнутая)
- Jelinski-Moranda (Марковская)
- Littlewood-Verrall (Байесовская)
- Logarithmic (Логарифмическая)

Также на экран выводится график функция количества ошибок m(t) или интенсивности m'(t).

В программе реализован описанный в пункте 4 алгоритм выбора моделей. По умолчанию при добавлении любых данных в проект считается, что эти данные собраны

на новом этапе, и это приводит к пересчету устойчивости моделей относительно предыдущего этапа.

6.2 Подробное описание функциональности

Программная реализация курсовой работы представляет собой приложение на языке Python. Выбор языка продиктован тем, что:

- Python достаточно удобен для реализации сложных математических вычислений (по сравнению, например, с C++)
- Желательна переносимость программы на разные платформы
- В Python есть возможность удобной разработки графического интерфейса — PyQT
- В Python есть широкие возможности для работы с шаблонными функциями.

Для запуска необходимо наличие стандартной установки интерпретатора Python и PyQT 4.

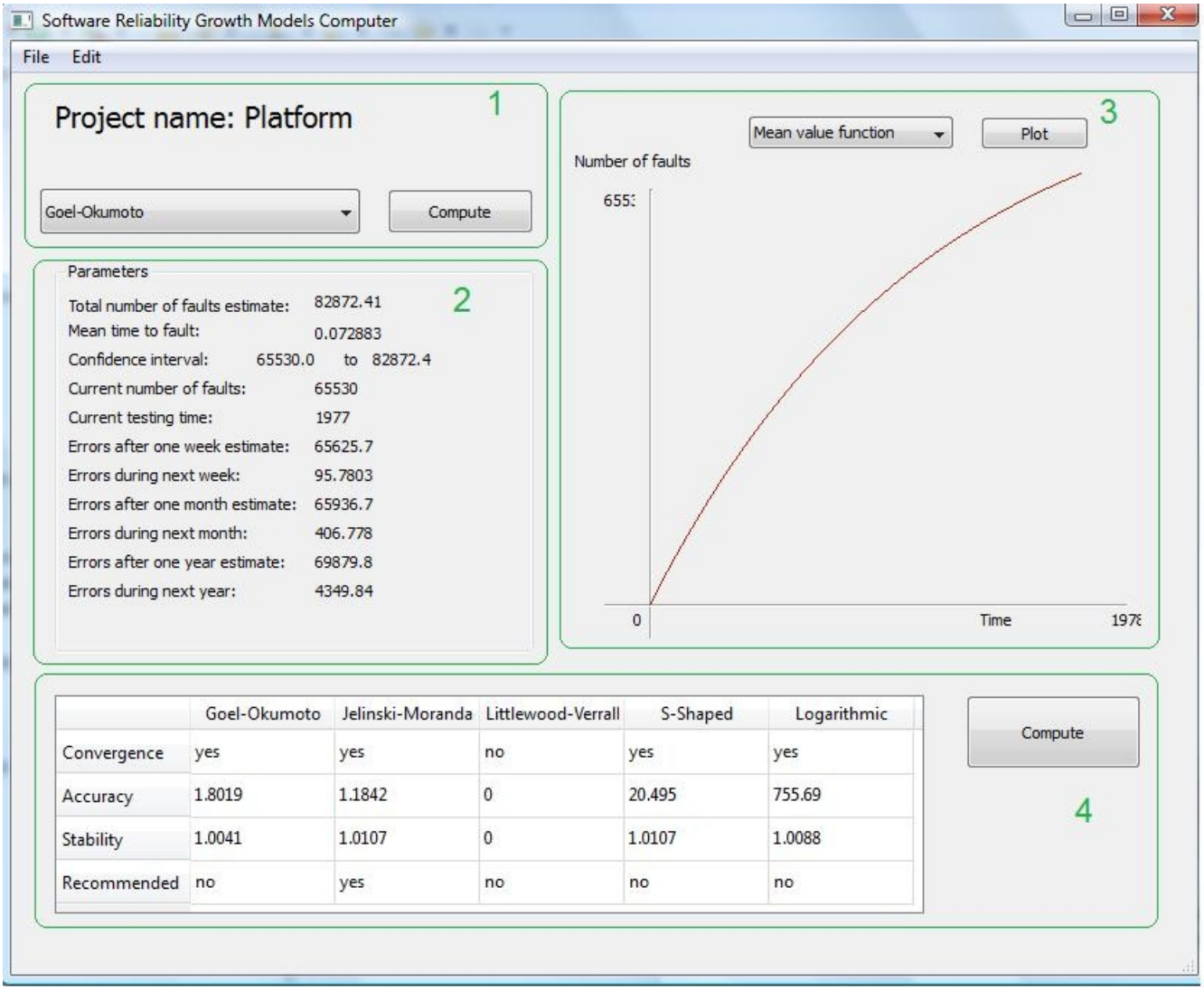

Рисунок 5: Главное окно программы

На рисунке 5 показано главное окно программы. Выделены основные области:

1. Название текущего проекта, раскрывающийся список, содержащий все выбранные модели, кнопка Compute, запускающая расчет.

2. После нажатия на кнопку Compute в этом поле появляются результаты расчетов. Если параметр не может быть оценен (в силу ограничений модели), то вместо него остается слово «Undefined», если уравнения не решаются (модель не подходит), то появляется слово «Diverge»

3. График функции количества либо интенсивности, в зависимости от пункта, выбранного в раскрывающемся списке. Масштаб по осям различен, чтобы уместить все графики в окно. Число возле оси указывает на максимальное значение соответствующей переменной на показанном в окне фрагменте графика.

4. Таблица, после нажатия на кнопку Compute заполняющаяся данными о сравнении моделей: Convergence — сходимость, Accuracy — точность, Stability — устойчивость, Recommended — подходит к проекту.

Пункты главного меню:

File — New Project. Вызывает wizard создания нового проекта, запрашивающий название проекта, файл с исходными данными и список моделей для расчета.

File — Open Project. Открывает проект, выбранный пользователем из раскрывающегося списка.

File — Delete Project. Удаляет текущий проект.

File — About. Показывает информацию о версии программы.

File — Exit. Закрывает программу.

Edit — Change models list. Запрашивает новый список моделей для расчета.

Edit — Load new XML file. Заменяет текущие данные данными из указанного пользователем файла.

Edit — Setup data selection. Определяет ограничения, применяемые при отборе данных из XML-файла. Позволяет указать рассматривать только данные об ошибках, содержащие определенный идентификатор программиста, модули или серьезности.

Edit — Add data to current project. Позволяет добавить к текущим данным данные об одной ошибке.

Edit — Batch data load. Загружает данные из указанного XML-файла и объединяет их с уже имеющимися в проекте.

6.3 Внутренняя структура программы.

В программе два основных класса: MyMainWindow и Computer и несколько небольших классов.

MyMainWindow является наследником QtGui.QMainWindow и реализует главное окно. Сама форма нарисована с помощью Qt Designer и преобразована в специальный служебный класс стандартными средствами PyQt. Функции-члены класса MyMainWindow реализуют обработчики событий, связанных с нажатием на кнопки или выбором пунктов в меню. Кроме того, в классе хранятся члены-данные, прежде всего это преобразованный в ассоциативный массив XML-файл с входными параметрами, переменные, определяющие текущий проект, и логи, которые при закрытии программы записываются в текстовый файл в формате Pickle.

Класс Computer реализует непосредственный расчет оценок по различным моделям. Содержит следующие важные функции:

__init__ - конструктор. При создании экземпляра класса в конструкторе из специального файла загружаются непосредственно числа, соответствующие датам

обнаружения ошибок.

GoelOkumoto() производит расчет по экспоненциальной модели. Возвращает 5 чисел: оценку N (числа ошибок), оценку b (параметр изгиба), оценку MTTF (время до ближайшей ошибки) и две границы доверительного интервала для N.

JelinskiMoranda() производит расчет по марковской модели. Возвращает 3 числа: оценку N (числа ошибок), оценку phi (параметр интенсивности), оценку MTTF.

SShaped() производит расчет S-изогнутой модели. Возвращает 3 числа: оценку N (числа ошибок), оценку b (параметр изгиба), оценку MTTF.

Logarithmic() производит расчет логарифмической модели. Возвращает оценку MTTF и параметры кривой b1, b2

LittlewoodVerrall() производит расчет байесовской LV модели. Внутри этой функции реализован и расчет параметров методом простой итерации. Возвращает 4 числа: оценку MTTF и параметры a, b0, b1 (п. 3.4).

Solve(f, a, b) находит решение уравнения f(x) = 0 на отрезке [a,b] методом дихотомии и возвращает его в качестве результата

7. Экспериментальное исследование

7.1 Цель исследования

Задача исследования — на данных реальных проектов опробовать реализованные модели, и получить решение для задач 1-3 из раздела 3:

1. Определить промежуток времени между применениями алгоритма из раздела 5.
2. Выбрать наиболее подходящую к данному проекту модель оценки надежности из нескольких возможных.
3. Оценить, какой объем исходных данных необходим для того, чтобы модели давали результаты с точностью не ниже заданной.

7.2 Выбор проектов для исследования

Проекты выбирались исходя из следующих требований:

- Большой объем программы (сотни тысяч строк)
- Длительное время тестирования, обширная база данных о тестировании
- 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, выбирались проекты, использующие эту систему для хранения статистики тестирования. В качестве основного был взят проект Eclipse (http://www.eclipse.org/), в качестве дополнительных — Mozilla и Linux Kernel.

Проект Eclipse был выбран в качестве основного, так как содержит самую большую базу (около 260 000 баг-репортов) с наиболее подробной статистикой: указаны дата, модуль, фатальность ошибки, текущий статус для каждого бага.

7.3 Методика экспериментального исследования.

Производятся следующие шаги:

1. По выбранным проектам загружается статистика тестирования.
2. Для этих проектов и, выборочно, их компонент и подмножеств ошибок одинаковой серьезности (severity), получены следующие данные:
   - Оценка количества ошибок на ближайший период и сравнение этой оценки с реальными данными (по периодам за некоторое время). Такие расчеты производились для того, чтобы продемонстрировать соотношение прогнозов и реальных данных, а также для того, чтобы эмпирически обнаружить закономерности, которым следует процесс разработки конкретного проекта.
   - Точность и устойчивость рассматриваемых сходящихся моделей, в соответствии с алгоритмом из раздела 5.
3. Используя эти данные, с помощью алгоритма из раздела 5 получается ответ на задачи 2 и 3. Оценивая близость полученных результатов к реальным данным при разных промежутках времени между применениями алгоритма выбора модели, можно получить ответ на задачу 1.

7.4 Результаты расчетов

В Приложении приведены таблицы с результатами расчетов по некоторым моделям.

Расчеты производились с помощью описанного в разделе 5 программного средства. Так как статистика по различным проектам весьма обширная, в Приложении приведены только отдельные примеры, иллюстрирующие применение моделей и их сравнение.

На рис. 6-11 приведены данные из таблиц Приложения в более наглядном виде. Некоторые графики начинаются с момента времени t=10 периодов (недель или месяцев), чтобы сильные отклонения в масштабе в самом начале не мешали. В случае, если оценка не определена в некоторый период времени, график принимает значения 0.

Рисунки 6-7 показывают точность и устойчивость трех моделей, подробно рассмотренных в разделе 4: экспоненциальной (Goel-Okumoto, GO), Jelinski-Moranda (JM) и логарифмической (Loarithmic). Наилучшей точностью и устойчивостью является 100, соответственно графики показывают отклонение данных величин от 100 на протяжении некоторого промежутка времени.

Рисунки 8-11 показывают оценки количества ошибок, полученные в результате применения моделей оценки надежности к различным подпроектам и компонентам в сравнении с реальными данными (Data) за тот же период.

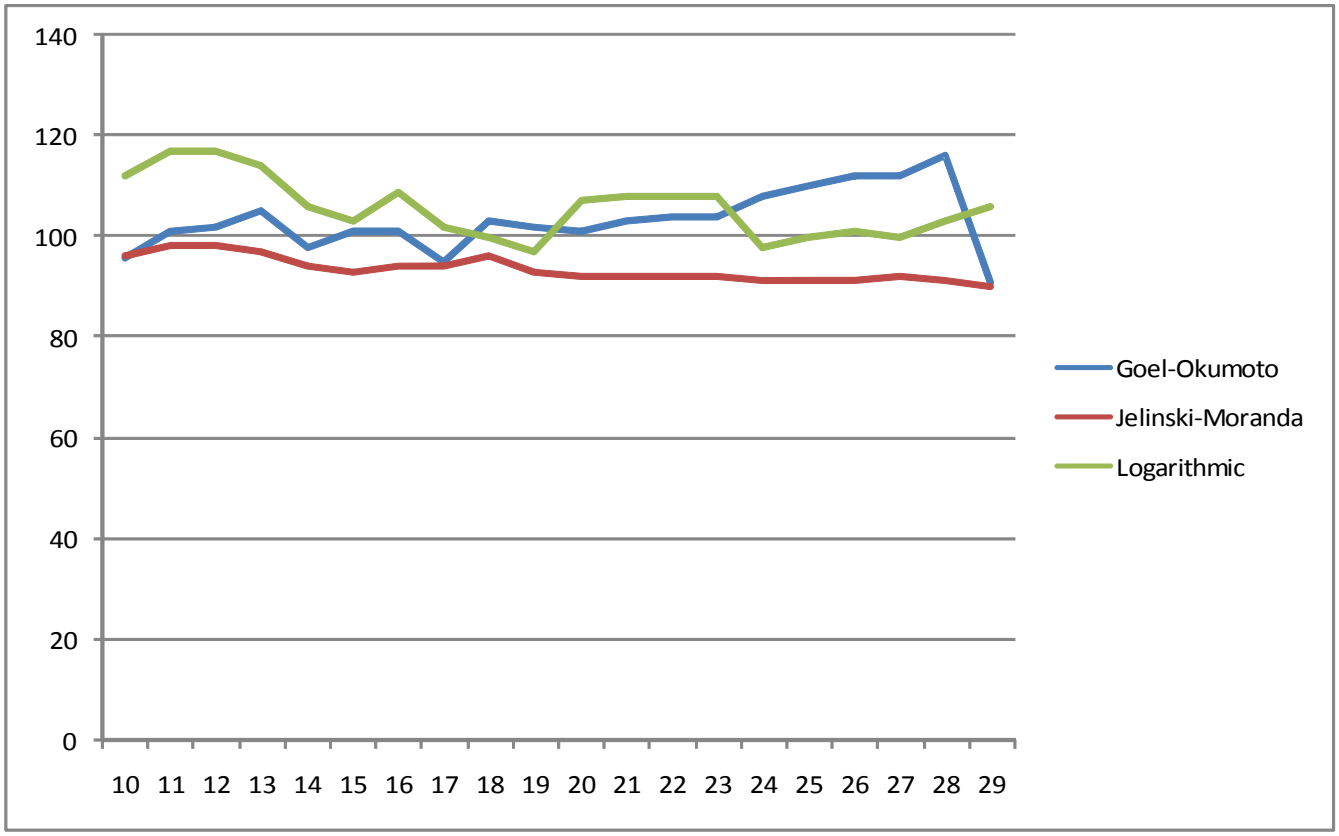

Рисунок 6: Проект JDT - Точность по трем моделям

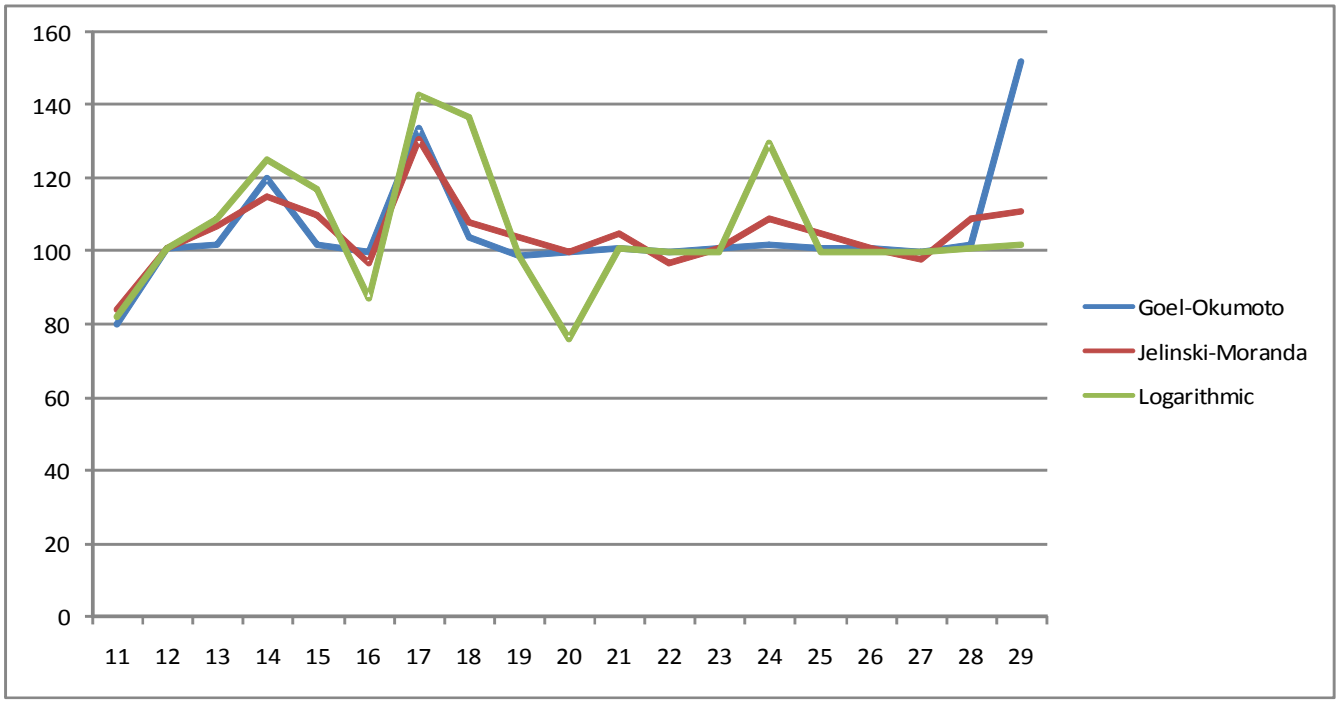

Рисунок 7: Проект JDT - Устойчивость по трем моделям

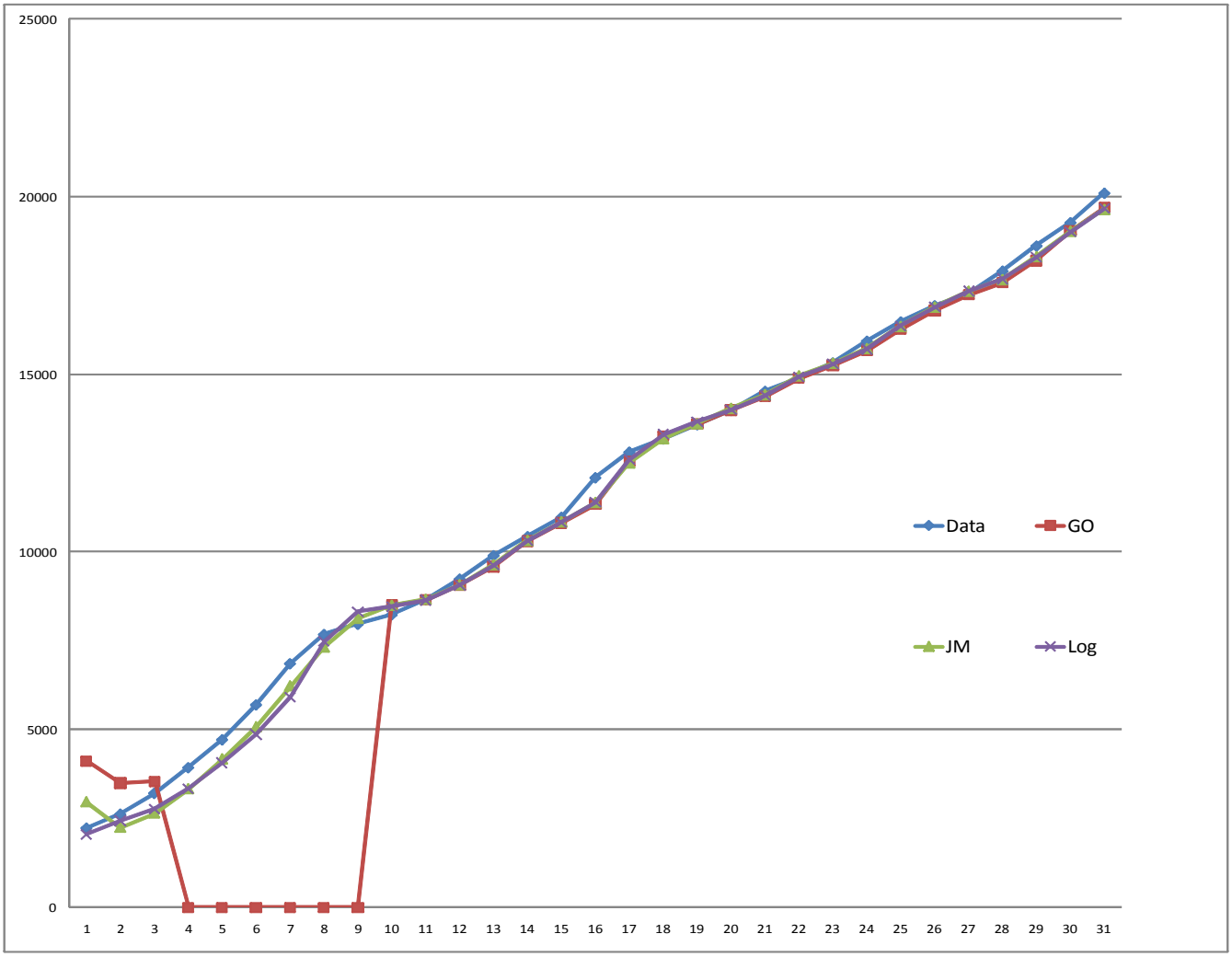

Рисунок 8: Проект JDT - количество ошибок к концу месяца и его оценки

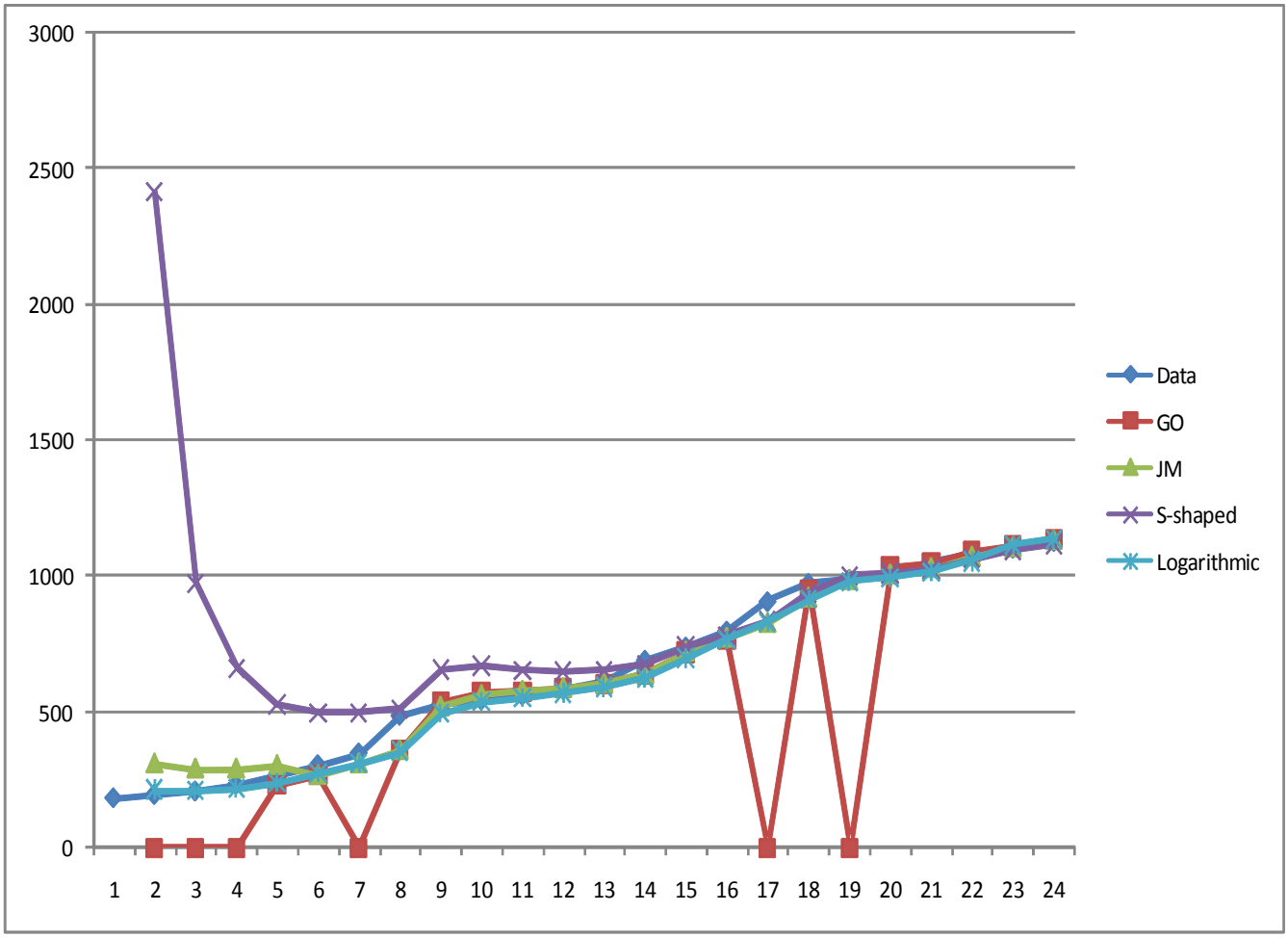

Рисунок 9: Количество ошибок с уровнем серьезности "major"

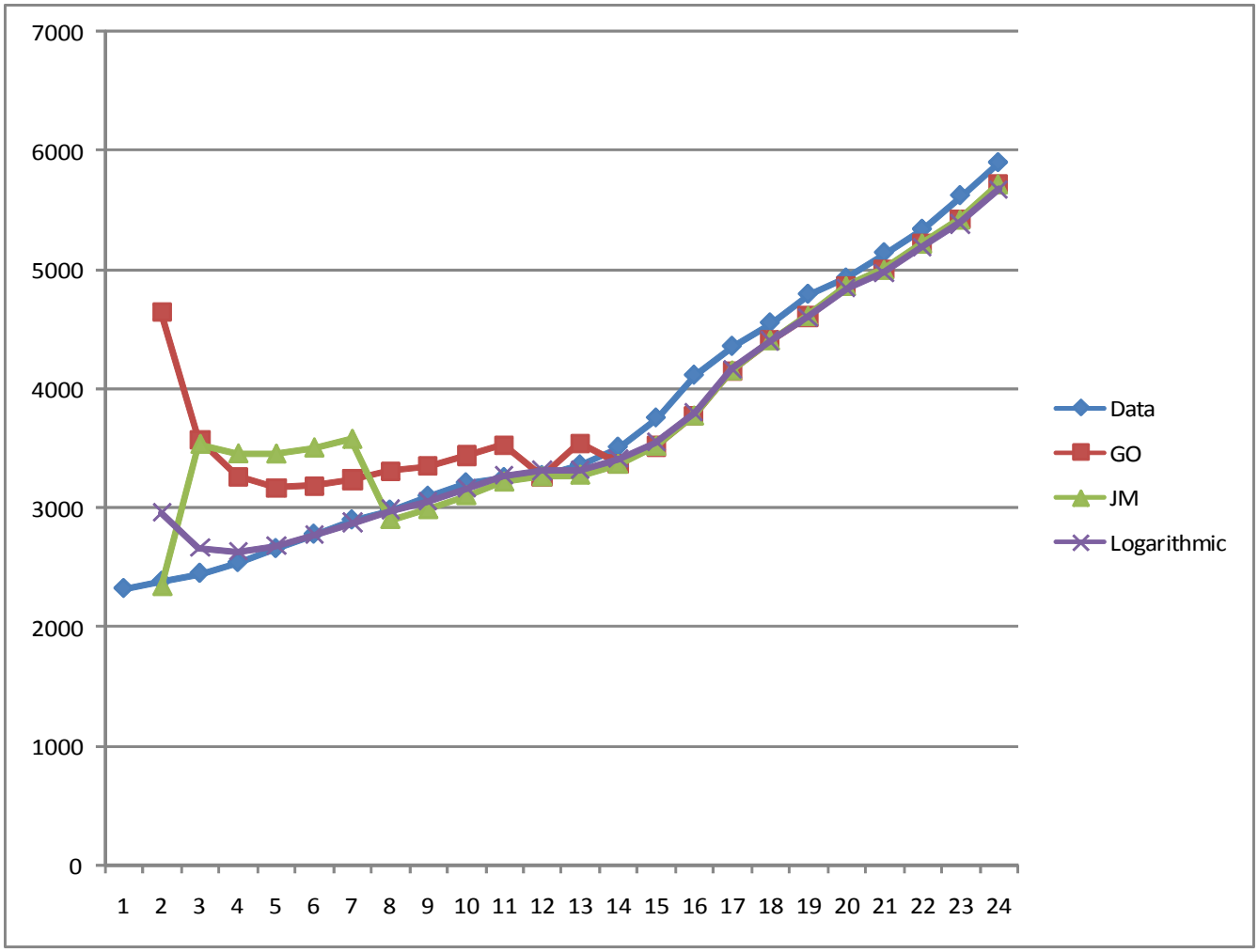

Рисунок 10: Количество ошибок к концу недели - данные и оценки

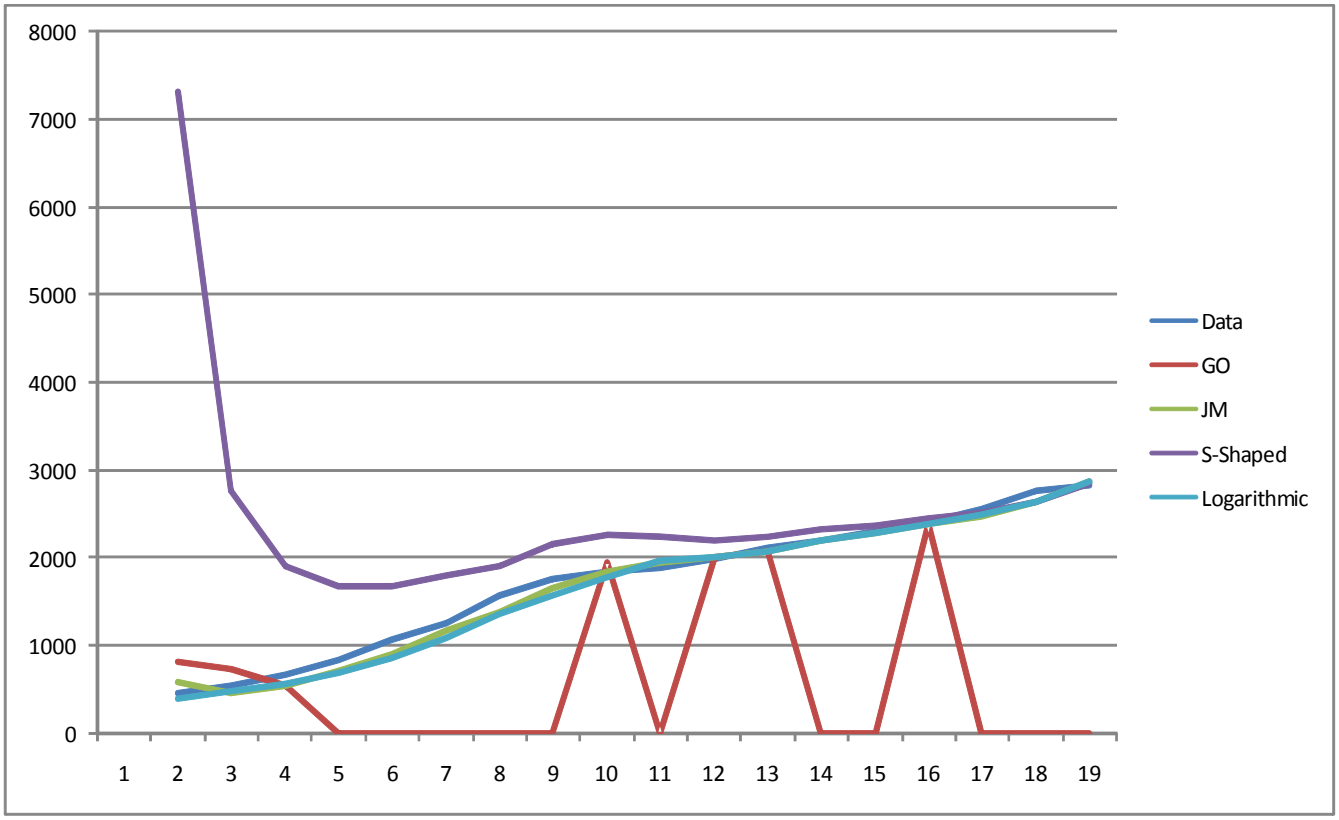

Рисунок 11: Количество ошибок в отдельном модуле

7.5 Выводы

Из приведенных в приложении таблиц (таблицы 2-3, например) видно, что точность и устойчивость начиная приблизительно с 10-15 периода, стабильно отклоняются от 100 не более, чем на 10 процентов, что можно считать приемлемым. Из этого можно сделать вывод о том, что в целом модели оценки надежности применимы к проектам с открытым исходным кодом, несмотря на децентрализованность тестирования и совмещения тестирования с разработкой.

Из экспериментов также сделан вывод, что наиболее приемлемым периодом для перерасчета является один месяц. При сравнении данных в таблицах 2, 4 и 7 приложения 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%). Разница видна также и на рисунках 8 и 10 — при разбиении по неделям расхождения оценок и результатах более явные. При разбиении по годам же данных слишком мало и уменьшения количества ошибок не прослеживается. Эмпирически такие результаты можно объяснить тем, что для расчета по годам / полугодиям нужно слишком много данных, а при расчете 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 перерасчете 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 расчете же по неделям точность сильно понижается. Тем не менее, вопрос о более точном определении расчетного периода остается открытым, возможно, стоит найти формулу или алгоритм для определения оптимальной величины расчетного периода, если имеется достаточно большой объем статистики.

Видно также, что приемлемая точность аппроксимаций и оценок достигается примерно к 10-15 периоду, 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. Таблицы 2, 5, 6 приложения свидетельствуют, что спустя некоторое время (те же 10-15 периодов) 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: таблица 5 и рисунок 11. 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.

Одну универсальную модель, которая могла бы хорошо подходить ко всем проектам с открытым исходным кодом, найти невозможно даже в пределах нескольких подпроектов одного большого сообщества. Более правильный подход — искать наилучшую модель для каждого отдельного проекта или даже его компоненты. Для этого можно применять описанный ранее алгоритм сравнения, либо руководствоваться априорными соображениями. Совмещение априорных данных о проекте и апостериорных оценок по имеющейся статистике и расчетам за прошедшие периоды — возможное направление для исследований в данной области.

8. Заключение

В рамках данной работы получены следующие результаты:

- Сделан обзор основного математического аппарата моделей оценки надежности и распространенных пуассоновских, марковских и байесовских моделей
- Из рассмотренных моделей выбраны 5 наиболее универсальных и распространенных (Goel-Okumoto, Jelinski-Moranda, S-Shaped, Логарифмическая, Littlewood-Verrall) и реализованы программно.
- Описано и реализовано программное средство сравнения моделей и выбора наиболее подходящей.
- Получена и обработана статистика по нескольким масштабным проектам с открытым исходным кодом (Eclipse, Mozilla, Kernel).
- Показана общая эффективность моделей оценки надежности применительно к выбранным проектам.
- Выбран наиболее подходящий промежуток времени для разбиения данных — один месяц.

Сформулированы следующие гипотезы о применении моделей оценки надежности к проектам с открытым исходным кодом:

- Более высокая точность достигается при максимальном возможном разбиении проекта на подпроекты и применении моделей к ним.
- Приемлемая точность и устойчивость достигается через 10-15 периодов.
- Наиболее подходящей для проектов с открытым исходным кодом является S-изогнутая пуассоновская модель.

В качестве дальнейшего развития данной темы 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.

9. Список литературы

На английском:

[1] R.Al-Ekram Software Reliability Growth Modeling and Prediction. Waterloo: Dept. of Electrical and Computer Engineering, University of Waterloo, 2002
[2] S.Alam Software Reliability Using Markov Chain Usage Model. Dhaka, Bangladesh: Depatment of Computer Science and engineering Bangladesh University of Engineering and Technology, 2005
[3] W.Farr Software Reliability Modelling Survey. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996
[4] C.Huang, M.Lyu A Unified Scheme of Some Nonhomogenous Poisson Process Models for 6. Software Reliability Estimation // IEEE transactions on software engineering. 2003. 29. №3.
[5] P.K.Kapur Software reliability growth and innovation diffusion models: an interface. Delhi: University of Delhi, 2004
[6] M.Limnios, N.Nikulin Recent Advances in Reliability Theory. Birkhäuser, 2000
[7] B.Littlewood How to measure software reliability, and how not to // Proceedings of the 3rd international conference on Software engineering. Atlanta, Georgia, United States, 1978. p 37 — 45.
[8] M.Lyu Software Reliability Engineering: A Roadmap // 2007 Future of Software Engineering. Washington, DC, USA, 2007. p. 153 — 170.
[9] J.D.Musa, K.Okumoto A logarithmic Poisson Execution time model for software reliability measurement // Proceedings of the 7th international conference on Software engineering. Orlando, Florida, United States, 1984. p 230 — 238.
[10] M.Ohba, X.Chou Does imperfect debugging affect software reliability growth // Proceedings of the 11th international conference on Software engineering. Pittsburgh, Pennsylvania, United States, 1989. p. 237 — 244.
[11] J. Shanthikumar A general software reliability model for performance prediction. Technical Report, Dept. of Ind. Eng. & Opns. Res., Syracuse University, 1980. p. 18
[12] C.Stringfellow An Empirical Method for Selecting Software Reliability Growth Models // Empirical Software Engineering 2002. 7. №4. p. 319 — 343.
[13] A.Wood Software Reliability Growth Models. Technical Report, 1996
[14] M.Xie Software Reliability Modelling. Singapore: World Scientific, 1991
[15] S.Yamada, H.Ohtera Software Reliability Growth Models with Testing Effort //IEEE Transaction on Reliability. R-35(1), 1986, p. 19-23.

На русском:

[16] Ширяев А. Н. Вероятность. М.: МЦНМО, 2004.
[17] Антонов А.В., Степанянц А.С. Методы анализа надежности (безошибочности) программного обеспечения программно-технических средств // Труды II Международной конференции «Идентификация систем и задачи управления». Москва, 2003.

На японском:

[18] 得能貢一、山田茂　マルコフモデルに基づくソフトウェアの運用信頼性評価法と最適リリース問題への応用。　鳥取大学工学部社会開発システム工学科、２００６．
Tokunou Kouichi, Yamada Shigeru A report on the problems of Markov software reliability growth models usage and final release. Tottori University, Faculty of Engineering sciences, Development department, 2006

[19] 梅田浩志　信頼度成長モデルを用いての信頼度成長曲線の作図プログラム作成。鳥取環境大学環境情報学部情報システム学科、 2008
Umeda Hiroshi The development of a program drawing reliability growth curves and implementing reliability growth models. Tottori University of Natural Sciences, Department of Infromation Systems, 2008

[20] 田村慶信、山田茂、木村光宏　デスクトップ環境におけるソフトウエア信頼性評価法に関する一考察
Tamura Keishin, Yamada Shigeru, Kimura Shoukou A report of the application of Software Reliability Growth Models for Desktop Environments, 2008

Приложение А

Первая таблица показывает оценки количества ошибок на конец каждого месяца.
Вторая таблица показывает результаты расчетов по алгоритму, описанному в разделе 4. Для каждого месяца приведены данные о точности и стабильности соответствующей модели. Устойчивость указана в процентах к предыдущему периоду, точность — в процентах среднеквадратичное отклонение, то есть чем ближе соответствующее значение к 100, тем лучше показатели.

Месяц	Реальные данные (data)	GO	JM	Logarithmic
1	1664	-	-	-
2	2226	4111	2966	2067
3	2619	3494	2250	2446
4	3207	3535	2654	2780
5	3929	-	3333	3356
6	4714	-	4173	4081
7	5697	-	5081	4877
8	6857	-	6230	5914
9	7678	-	7323	7483
10	7969	-	8134	8322
11	8235	8508	8500	8462
12	8662	8651	8669	8650
13	9240	9062	9082	9074
14	9899	9591	9639	9631
15	10434	10303	10324	10318
16	10979	10814	10850	10837
17	12094	11339	11396	11396
18	12813	12550	12507	12601
19	13186	13257	13193	13314
20	13587	13606	13613	13668
21	13993	13987	14045	14007
22	14528	14374	14431	14399
23	14901	14895	14965	14926
24	15329	15251	15315	15287
25	15940	15665	15730	15705
26	16481	16266	16354	16354
27	16937	16796	16898	16888
28	17283	17240	17347	17336
29	17912	17573	17674	17672
30	18618	18193	18318	18298
31	19275	19039	19033	19002

32	20108	19690	19648	19657

Таблица 2: Проект JDT. Количество ошибок по месяцам - оценки и результаты

Месяц	Goel-Okumoto		Jelinski-Moranda		Logarithmic	
	Точность	Усточивость	Точность	Усточивость	Точность	Усточивость
1	278	100	819	100	94	100
2	75	53	1057	68	8091	107
3	46	74	457	118	7460	108
4	-	-	183	130	6173	122
5	-	-	120	136	1613	126
6	-	-	101	141	2875	129
7	-	-	94	163	765	157
8	-	-	102	127	105	641
9	-	-	103	112	103	126
10	96	-	96	103	112	59
11	101	80	98	84	117	82
12	102	101	98	101	117	101
13	105	102	97	107	114	109
14	98	120	94	115	106	125
15	101	102	93	110	103	117
16	101	100	94	97	109	87
17	95	134	94	131	102	143
18	103	104	96	108	100	137
19	102	99	93	104	97	99
20	101	100	92	100	107	76
21	103	101	92	105	108	101
22	104	100	92	97	108	100
23	104	101	92	101	108	100
24	108	102	91	109	98	130
25	110	101	91	105	100	100
26	112	101	91	101	101	100
27	112	100	92	98	100	100
28	116	102	91	109	103	101
29	91	152	90	111	106	102

Таблица 3: Проект JDT - точность и усточивость моделей

Период	Данные	GO	JM	S-Shaped	Logarithmic
1	26044				
2	46884	37075	38998	29604	40927
3	76440	56401	60412	47147	60468
4	101660	-	89633	76456	79298
5	124086	-	114739	-	104502
6	141994	133700	137117	-	126843
7	155852	149643	155305		144604
8		161820	169796		158288

Таблица 4: Проект Platform - оценки количества ошибок за год

Период	Данные (data)	GO	JM	S-Shaped	Logarithmic
1	331				
2	458	817	592	7319	411
3	548	724	464	2773	504
4	675	556	557	1902	582
5	840	-	705	1684	708
6	1057	-	901	1678	874
7	1260	-	1166	1800	1099
8	1557	-	1375	1913	1362
9	1763	-	1647	2152	1588
10	1837	1944	1849	2272	1793
11	1891	-	1940	2242	1975
12	1983	2005	2007	2212	2009
13	2108	2072	2085	2242	2087
14	2190	-	2208	2322	2208
15	2305	-	2280	2363	2285
16	2399	2392	2393	2445	2396
17	2548	-	2482	2512	2486
18	2770	-	2638	2642	2639
19 (560)	2823	-	2871	2850	2870

Таблица 5: Проект JDT - Статистика по модулю APT

Месяц	Реальные данные (data)	GO	JM	S-shaped	Logarithmic
1	181				
2	193	-	309	2412	213
3	206	-	290	972	210
4	230	-	290	661	218
5	264	231	303	528	239
6	303	266	266	497	272
7	347	-	309	498	311
8	486	358	358	508	355
9	528	535	518	655	496
10	543	569	561	669	538
11	561	574	577	653	553
12	582	585	587	648	570
13	616	603	605	652	591
14	690	640	639	675	625
15	739	718	721	740	699
16	798	766	772	781	766
17	907	-	826	833	831
18	975	950	920	937	914
19	990	-	984	1001	982
20	1008	1031	1005	1011	997
21	1049	1047	1029	1024	1014
22	1082	1087	1071	1062	1056
23	1105	1109	1105	1092	1115
24	1130	1131	1132	1113	1137

Таблица 6: Проект JDT - ошибки с уровнем критичности "major"

Неделя	Реальные данные (data)	GO	JM	Logarithmic
1	2328			
2	2388	4643	2340	2967
3	2456	3572	3538	2667
4	2545	3265	3458	2640
5	2663	3172	3456	2683
6	2784	3187	3508	2775
7	2904	3239	3580	2879
8	2987	3309	2905	2988
9	3104	3351	2989	3060
10	3216	3439	3107	3171
11	3260	3527	3221	3277
12	3277	3263	3264	3316
13	3366	3543	3279	3328
14	3515	3369	3369	3414
15	3761	3521	3521	3561
16	4120	3775	3777	3806
17	4362	4153	4157	4167
18	4559	4408	4411	4409
19	4800	4609	4615	4606
20	4936	4861	4867	4847
21	5149	5000	5003	4982
22	5346	5219	5224	5194
23	5629	5422	5426	5391
24	5906	5715	5724	5676

Таблица 7: Проект Platform - статистика по неделям