10 моментов уменьшения ошибочности ПО

Барри Боэм (**Barry Boehm**), Университет Южной Калифорнии
Виктор Базили (**Victor R. Basili**), Университет Мэриленда

Недавно грант Национального научного фонда дал нам возможность основать Центр эмпирически основанной разработки программного обеспечения (ЦЭОРПО). ЦЭОРПО стремится как можно больше преобразовать процесс разработки программного обеспечения из кустарной практики в инженерную практику посредством извлечения, организации и распространения эмпирических данных по разработке программного обеспечения и эволюционной феноменологии. Словосочетание «как можно больше» отражает тот факт, что разработка программного обеспечения должна оставаться человеко-интенсивным и постоянно меняющимся полем деятельности. Тем не менее, мы выявили, что исследователи создали объективные и количественные данные, отношения и предсказывающие модели, которые помогают разработчикам программного обеспечения избегать предсказываемых ловушек и увеличивают их способность предсказывать и управлять эффективными программными проектами.

Здесь мы описываем наработки в этой области, которые имели место с момента публикации «10 моментов промышленных метрик» в 1987 году (Боэм Б. *Программное обеспечение IEEE*. – 1987, сентябрь. – С. 84-85 [B. Boehm, *IEEE Software*, Sept. 1987, pp. 84-85]). Принимая во внимание, что ЦЭОРПО отдаёт высокий приоритет уменьшению ошибочности программного обеспечения, мы думаем, что пора обновить ту раннюю статью представлением нижеследующего списка 10 моментов уменьшения ошибочности программного обеспечения.

ОДИН

Обнаружение и исправление программной проблемы после доставки часто в 100 раз дороже, чем обнаружение и исправление её на этапах формулировки требований и проектирования.

Как заметил Боэм (Boehm) в 1987 году: «Понимание этого было главной движущей силой в фокусировке практики промышленного программного обеспечения на сквозном анализе требований и проектировании, на ранней проверке и утверждении и на первостепенном прототипировании и моделировании с целью избежать дорогих последующих исправлений».

Сложность программного обеспечения и ускоренные графики разработки делают трудным избежание ошибок. Эти 10 техник могут помочь уменьшить недостатки Вашего кода

В этом обновлённом списке мы добавили слово «часто», чтобы отразить дополнительное понимание этих наблюдений. С одной стороны, фактор увеличения стоимости для малых, некритических программных систем равен, скорее, 5:1, чем 100:1. Это отношение говорит от том, что мы можем разрабатывать такие системы более эффективно в менее формальном, длительном режиме прототипов, который по-прежнему делает упор на получение результатов больше рано, чем поздно.

С другой стороны, хорошая архитектурная практика может значительно уменьшить фактор увеличения стоимости даже для больших критических систем. Такая практика уменьшает стоимость большинства исправлений благодаря их заключению в малые,

хорошо изолированные модули. Хорошим примером служит состоящая из более чем миллиона строк программного кода система CCPDS-R, описанная в книге Уокера Ройса (Walker Royce) *«Управление программными проектами»* (*Software Project Management*) издательства «Аддисон–Уэсли» (Addison-Wesley), 1998 года.

ДВА

Текущие программные проекты тратят от 40% до 50% усилий на переделки, которых можно избежать.

Такие переделки состоят из усилий, затраченных на исправление программных трудностей, которые могли быть обнаружены раньше и исправлены менее дорого, или их можно было вообще избежать. Соответственно, очевидно, что некоторые усилия должны состоять из «неизбежных переделок», – это наблюдение приобрело правдоподобность, увеличивающуюся вместе с растущей реализацией систем, больше ориентированных на взаимодействие с пользователем, на основе *внезапных* процессов. В таких процессах требования возникают из прототипирования и других форм многопользовательского участия, что является отклонением от традиционных *уменьшающих* процессов, которые выставляют вперёд требования, после чего уменьшают их на практике через проектирование и кодирование. Внезапные процессы показывают, что изменения в определении системы, касающиеся повышения её рентабельности, не должны игнорироваться посредством классификации их как ошибок, которых можно избежать.

Уменьшение объёма переделок, которых можно избежать, может обеспечить значительное увеличение производительности при разработке программного обеспечения. В нашем поведенческом анализе того, как стоимость программного обеспечения влияет на затрачиваемые усилия, для модели «Кокомо II» (Боэм Б. И др. *Расчёт стоимости программного обеспечения на основе модели «Кокомо II»*. – Прентис Холл, 2000 [B. Boehm et al., *Software Cost Estimation with Cocomo II*, Prentice Hall, 2000]), мы обнаружили, что большая часть сбережённых усилий, порождённых улучшением зрелости процесса разработки программного обеспечения, программных архитектур и управления программными рисками, исходит из уменьшения объёма переделок, которых можно избежать.

ТРИ

Около 80% переделок, которых можно избежать, исходят от 20% ошибок.

Значение этих 80% может быть меньше для малых систем и больше для очень больших. Два главных источника переделок, которых можно избежать, включают в себя наскоро составленные требования и номинальные формы проектирования и разработки, при которых позднее приспособление неноминальных требований приводит к большой ломке архитектуры, проекта и кода. Отслеживающая система для отчётов по программным проблемам, которая записывает усилия, затрачиваемые на исправление каждой ошибки, позволяет Вам достаточно легко анализировать данные с целью определения и обработки дополнительных источников переделок.

ЧЕТЫРЕ

Около 80% ошибок исходят из 20% модулей, а около половины модулей свободны от ошибок.

Исследования различных сред на протяжении многих лет показали, что с поразительным постоянством от 60% до 90% ошибок возникают из 20% модулей, с медианой около 80%. С одинаковым постоянством почти все ошибки концентрируются в примерно половине произведённых модулей.

Соответственно, очевидно, что выявление характеристик склонных к ошибкам модулей в конкретной среде может быть оправданным. Разнообразие контекстно-зависимых факторов способствует склонности к ошибкам. Некоторые факторы обычно способствуют склонности к ошибкам вне зависимости от контекста, однако, включая уровень связывания данных, размер, сложность и объём изменений повторно используемого кода.

ПЯТЬ

Около 90% времени простоя исходит, в основном, от 10% ошибок.

Некоторые ошибки непропорционально влияют на время простоя системы и её надёжность. Например, анализ истории отказа программного обеспечения для девяти больших программных продуктов фирмы IBM показывает, что около 0,3% ошибок породили около 90% времени простоя. Таким образом, тестирование, основанное на рисках, включающее понимание работы системы и уделяющее особое внимание тестированию по высококритичным сценариям, ясно является экономически выгодным.

ШЕСТЬ

Тщательный пересмотр ловит 60% ошибок.

Учитывая то, что обнаружение и исправление большинства ошибок на ранних этапах цикла разработки проекта является экономически более выгодным, чем их более позднее обнаружение, мы ищем техники, которые обнаруживают ошибки как можно раньше. Многочисленные исследования подтверждают, что тщательный пересмотр кода является эффективной техникой, которая ловит от 31% до 93% ошибок, с медианой около 60%. Таким образом, упомянутое в статье 1987 года значение в 60% остаётся приемлемой оценкой.

Тщательный пересмотр, инструменты анализа и тестирование ловят различные классы ошибок в различных точках цикла разработки.

Факторы, влияющие на процент отлавливаемых ошибок, включают в себя число и тип выполняемых тщательных пересмотров, размер и сложность системы и частоту ошибок, лучше отлавливаемых посредством исполнения, таких как ошибки параллелизма и алгоритмов. Наши исследования представляют доказательства того, что тщательный пересмотр, инструменты анализа и тестирование ловят различные классы ошибок в различных точках цикла разработки. Нам требуются дополнительные эмпирические исследования, чтобы помочь выбрать наилучшую смешанную стратегию вкладов в уменьшение ошибочности.

СЕМЬ

Пересмотры с перспективой ловят на 35% больше ошибок, чем ненаправленные пересмотры.

Техника чтения, основанная на сценариях (Базили В.Р. Развивающиеся и упаковывающие технологии чтения. / *Системы и программное обеспечение*. – 1997, № 1. – С. 3-12 [V.R. Basili, "Evolving and Packaging Reading Technologies," *J. Systems and Software*, vol. 38, no. 1, 1997, pp. 3-12]) предлагает набор формальных процедур для обнаружения ошибок, основанных на изменении перспектив. Объединение нескольких перспектив в один просмотр предлагает широкое, но всё ещё сосредоточенное покрытие

пересматриваемого документа. Этот подход стремится породить сосредоточенные техники, направленные на особые цели обнаружения ошибок, путём использования существующей истории ошибок организации.

Техники чтения, основанная на сценариях, применяются при формулировке требований, объектно-ориентированном проектировании и просмотрах пользовательского интерфейса. Улучшения в норме выявления ошибок изменяется от 15% до 50%. Далее, сосредоточенные техники чтения способствуют обучению неопытного персонала, улучшают связь по процессу и благоприятствуют продолжительным улучшениям.

ВОСЕМЬ

Практика личной дисциплинированности может уменьшить нормы внесения ошибок на величину до 75%.

На практике представлены несколько процессов личной дисциплинированности. Среди них процесс разработки программного обеспечения «чистая комната» Харлана Милла (Harlan Mill) и личный процесс разработки программного обеспечения (ЛПРПО) Уоттса Хамфри (Watts Humphrey).

Данные по использованию технологии «чистая комната» в НАСА показывают от 25% до 75% уменьшения нормы ошибочности во время тестирования. Использование «чистой комнаты» также показало уменьшение объёма усилий, затрачиваемых на переделки, так, что только 5% исправлений потребовали более одного часа, тогда как стандартный процесс приводил к тому, что более 60% исправлений отнимали столько же времени.

Сильное сосредоточение технологии ЛПРПО на корневом анализе программных ошибок и перегрузок, порождаемых индивидуумом, и на разработке личных листов проверки и личной практики для избежания будущего повторения, значительно уменьшило нормы личной ошибочности. Разработчики часто наслаждаются уменьшением ошибочности в соотношении 10:1 между упражнениями № 1 и № 10 учебного курса ЛПРПО.

Эффекты на уровне проекта более рассредоточены. Они зависят от таких факторов, как уровень зрелости программного обеспечения в организации и готовность персонала и организации работать в рамках высокоструктурированной программной культуры. Когда Вы объединяете ЛПРПО с полностью совместимым командным процессом разработки программного обеспечения (КПРПО), нормы уменьшения ошибочности могут взлететь 10-кратно, а то и больше, для организации, которая работает на умеренном уровне зрелости. Результаты имеют склонность быть менее впечатляющими, если организация уже использует полностью зрелые процессы.

Специальный выпуск «Кроссток» (*CrossTalk*) за июнь 2000 года под названием «Сбережение времени использованием ЛПРПО и КПРПО» предлагает хороший набор соответствующих обсуждений, включая опыт, показывающий, что добавление ЛПРПО и КПРПО в организацию уровня СММ 5 уменьшило число ошибок по тесту на допустимость на величину, в общей сложности, около 50%, и уменьшило число высокоприоритетных ошибок на величину около 75%.

ДЕВЯТЬ

При прочих равных условиях, разработка высоконадёжных программных продуктов стоит на 50% больше из расчёта на одну инструкцию исходного текста, чем разработка низконадёжных программных продуктов. Как бы там ни было, вложение средств более чем целесообразно, если проект включает значимые операции и затраты на эксплуатацию.

Анализ точечных данных 161 проекта для модели «Кокомо II» обнаружил добавочную стоимость в размере 53% для её фактора требуемой надёжности, при

нормализации эффектов 22 других факторов. Значит ли это, что ключевая книга Филипа Кросби (Philip Crosby) под названием *«Качество бесплатно»* (издательство «Ментор» (Mentor), 1980 год) неправа? Возможно, для некоторого низкокритичного программного обеспечения, с коротким жизненным циклом, но не для самых важных случаев.

Во-первых, в модели «Кокомо II» низконадёжное программное обеспечение стоит примерно на 50% больше из расчёта на одну инструкцию исходного кода при обслуживании, чем при разработке, тогда как высоконадёжное программное обеспечение стоит примерно на 15% меньше при обслуживании, чем при разработке. Для типичной стоимости жизненного цикла с её 30% разработки и 70% эксплуатации низконадёжное программное обеспечение становится таким же по стоимости из расчёта на одну инструкцию, как и высоконадёжное программное обеспечение – опять, при прочих равных условиях.

Во-вторых, в подобной «Кокомо II» модели качества высоконадёжное программное обеспечение избегает в 4 раза больше ошибок, чем средненадёжное программное обеспечение, которое, в свою очередь, избегает в 4 раза больше ошибок, чем низконадёжное программное обеспечение. Например, рассмотрим средненадёжную систему вроде коммерческой биллинговой системы, в которой операционная стоимость программных ошибок – из-за потерянного рабочего времени, потерянных объёмов продаж, стоимости дополнительного обслуживания клиентов, судебных издержек, убытков повторного бизнеса и так далее, – приблизительно равняется стоимости разработки и эксплуатации программного обеспечения. Для такой системы норма ошибочности, увеличенная вследствие использования низконадёжного программного обеспечения, сделала бы стоимость владения ею приблизительно в 3 раза большей, чем стоимость владения высоконадёжным программным обеспечением.

ДЕСЯТЬ

От 40% до 50% пользовательских программ содержат серьёзные ошибки.

Исследование 1987 года в этой области (Браун П.С., Гаулд Дж.Д. Экспериментальное исследование людей, создающих электронные таблицы. / АКМ транс. офис. инф. сист. – 1987, июнь. – С. 258-272 [P.S. Brown and J.D. Gould, "An Experimental Study of People Creating Spreadsheets," *ACM Trans. Office Info. Sys.*, July 1987, pp. 258-272]) выявило, что 44% из 27 электронных таблиц, созданных опытными пользователями, содержали серьёзные ошибки – в основном, в формулах. Притом, что сами авторы были уверены, что они создали правильные таблицы.

Создатели средств веб-программирования сталкиваются с серьёзной проблемой предоставления своих инструментов, наравне с ремнями и подушками безопасности, вместе со средствами безопасного вождения и правилами дорожного движения.

Последующие лабораторные эксперименты показали нормы ошибочных электронных таблиц в диапазоне от 35% до 90%. Анализ операционных электронных таблиц показал нормы ошибочности в диапазоне от 21% до 26%; более низкие нормы объясняются, вероятно, исправлениями, делаемыми уже в процессе операции.

Сейчас, а в будущем даже больше пользовательских программ распространятся от электронных таблиц до веб-сценарных языков программирования, способных отправлять агентов в киберпространство, чтобы сделать для Вас дела. Звания «начинающих волшебников» – пользователей-программистов также разрастутся быстро, давая многим, кто мало обучен или у кого мало опыта в том, как избегать или определять

высокорискованные ошибки, огромную силу создания высокорискованных ошибок. Одно исследование для книги о модели «Кокомо II» позволило приблизительно посчитать, что к 2005 году в США будет 55 миллионов пользователей-программистов. Если мы классифицируем разработчиков активных веб-страниц как пользователей-программистов, то это предсказание покажется правдоподобным.

Таким образом, создатели средств веб-программирования сталкиваются с серьёзной проблемой предоставления своих инструментов, наравне с ремнями и подушками безопасности, вместе со средствами безопасного вождения и правилами дорожного движения. Эта исследовательская проблема разработки программного обеспечения – одна из нескольких, обнаруженных в исследовании Национального научного фонда, «Получение интеллектуального контроля над разработкой программного обеспечения», которое мы подытожили в журнале *«Компьютер»* (номер за май 2000 года, с. 27-33).

Конечно, наш список может извлечь выгоду из совершенствования и дальнейших эмпирических исследований уменьшения ошибочности. Многие данные, объявленные нами, например, не имеют силы для взаимодействия между многими переменными, которые, если известны, могут дать ответы на вопросы вроде:

- Если я вкладываю средства в тщательный пересмотр, «чистую комнату» и «ЛПРПО», не плачу ли трижды за те ошибки, которые будут удалены?
- Каких объёмов тестирования эти вложения помогут мне избежать?

Мы надеемся привлечь разработчиков программного обеспечения к расширению списка 10 моментов уменьшения ошибочности программного обеспечения и других доступных в настоящий момент данных до постоянно развивающегося, открытого, доступного через веб-интерфейс справочника эмпирических результатов по стратегиям уменьшения ошибочности программного обеспечения. Мы также планируем начать подобные справочники по готовым коммерческим системам и другим возникающим программным областям. Мы приветствуем Ваше участие в этих усилиях и настоятельно советуем Вам посетить веб-сайт ЦЭОРПО по адресу http://www.cebase.org с целью получения дальнейшей информации. Вы также можете найти расширенную версию этой статьи по адресу http://www.cebase.org/defectreduction/top10.☼

***Барри Боэм** (**Barry Boehm**) является директором Центра разработки программного обеспечения Университета Южной Калифорнии. С ним можно связаться по адресу boehm@sunset.usc.edu.*

***Виктор Базили** (**Victor R. Basili**) является профессором Института продвинутых компьютерных исследований и отделения информатики и вычислительной техники в Университете Мэриленда. С ним можно связаться по адресу basili@cs.umd.edu.*