Лиа Хоффман (Leah Hoffman)
(Перевод К. Фируна (constantine@firun. ru) статьи Hoffman L. In Search of Dependable Design // Communications of the ACM. 2008. Vol. 5 1 . № 07/08, pp. 14-16).

В поисках надёжного кода

Как разработчики программного и аппаратного обеспечения могут повысить надёжность своих систем?

В 1994 году в процессорах Intel Pentium I была обнаружена странная ошибка. Томас Р. Найсли (Thomas R. Nicely), в то время работавший в Линчбургском колледже штата Вирджиния, заметил, что процессор в некоторых случаях даёт неверные ответы в задачах на вычисление деления чисел с плавающей запятой. Вскоре и другие исследователи подтвердили ошибку и предоставили дополнительные примеры ошибочной работы процессора. И хотя изначально Intel пыталась спустить проблему на тормозах, но в конечном счёте под воздействием общественного мнения и широкой огласки проблемы в прессе, корпорации пришлось заменить все проблемные процессоры.

«Это была первая ошибка такого рода, которая стала главной темой вечерних новостей» – отмечает Эдмунд Кларк (Edmund Clarke) из Университета Карнеги-Мэллона (Carnegie Mellon University). Корпорации эта ошибка обошлась примерно в 500 миллионов долларов.

Практически 15 лет спустя, ошибка в Pentium I продолжает служить нравоучительным напоминанием, того, насколько дорога может быть ошибка, совершённая на этапе конструирования. Эта проблема не чужда и в создании программного обеспечения: система электронного документооборота, стоимостью в 170 миллионов долларов, была забракована ФБР в 2005 из-за бесчисленных сбоев, к тому же в конце 90-ых сбойная система учёта налоговых сборов Внутренней Налоговой Службы США потребовала миллиарды долларов для исправления большинства ошибок. И во времена, когда люди полагаются на компьютеры практически во всём – в автомобилях, сотовых телефонах, банкоматах и т. д. – цена ошибок при проектировании становится всё выше и выше. Хотя данные о финансовых потерях в результате программно-аппаратных ошибок очень сложно собрать, Национальный институт стандартов и технологий (NIST), подсчитал примерный объём потерь американского бюджета только лишь из-за сбойного программного обеспечения, которое вызывает потерю информации, уменьшившуюся производительность и увеличившиеся вложения на ремонт и поддержание программно-аппаратных средств. По их подсчётам, потери для американской экономики составляют примерно 59,5 миллиардов долларов ежегодно. Но дело

здесь не только и не столько в деньгах, сколько в человеческих жизнях. Сбойное программное обеспечение уже вызывало отключение дисплеев в кабине самолёта, выход из строя морских нефтяных платформ и сбои в системах наведения ракет.

«Из-за программных ошибок случилось лишь несколько катастроф. Но мы идём всё ближе и ближе к краю…» – говорит Дэниел Джексон (Daniel Jackson) из МИТ.

Эксперты сходятся во мнении, что выход из строя программного обеспечения происходит не из-за маленьких ошибок в коде, а из-за упущений во время проектирования. (Во многих случаях, ошибки в области безопасности, которые совершаются во время реализации, являются исключениями из этого правила.) Один класс ошибок появляется во время описания технического задания: проектирование программного обеспечения зачастую плохо объясняется или плохо понимается. Ещё один класс ошибок возникает из-за человеческого фактора, когда инженеры совершают недостаточно обоснованные предположения о той среде, в которой будут работать программно-аппаратные средства. Зачастую инженеры ошибаются в своих оценках, и сбои могут возникнуть в ситуации, которая не была предусмотрена на стадии проектирования.

Но ошибки могут возникать абсолютно в любое время. «Так как люди не совершенны и делают ошибки, то они могут их делать и на стадии проектирования», – предупреждает Джерард Хольцман (Gerard Holzmann) из лаборатории надёжного программного обеспечения NASA/JPL.

Хольцман – один из небольшой группы исследователей, которые занимаются разработкой программного обеспечения, приёмов и методов для повышения надежности проектирования. В настоящее время большинство программ отлаживают и затем улучшают путём тестирования. Тестирование может быть полезным для поиска маленьких ошибок, заявляют исследователи, но оно абсолютно беспомощно, когда требуется определить структурные ошибки. К тому же тесты, созданные для определённых сценариев, не всегда могут найти ошибки в поведении программ за пределами этих сценариев. По этой причине ведётся поиск дополнительных стратегий.

Один из многообещающих приёмов – это model checking. Идея этого метода заключается в создании алгоритма для проверки математической модели, в которой заложена логика работы программных и аппаратных средств. Несмотря на то, что реализация этого метода может быть очень длительной, это, тем самым, заставляет разработчиков излагать свои технические задания и требования в систематизированном, математическом виде, что минимизирует неясность. Ещё более важным является возможность model checkers автоматически дать контр-пример при нахождении ошибки, помогая разработчикам найти ошибку ещё до момента её написания в коде программы.

«Когда люди используют термин «надёжность», они имеют вероятностное представление, что ошибки находятся достаточно редко, в то время как люди, занимающиеся формальной верификацией могут с уверенностью сказать, что программа будет работать верно при всех заданных условиях», – объясняет Аллен Эмерсон (Allen Emerson) из Университета Техаса в Остине (the University of Texas at Austin). (В распознавании важности формальной верификации премия ACM имени Алана Тьюринга за 2007 год присуждена Эдмунду Кларку, Аллену Эмерсону и Джозефу Сифакису (Joseph Sifakis) за их первооткрывательскую работу в области model checking).

Model checking показал

поразительный успех в верификации аппаратных средств. К примеру Худонг Жао (Xudong Zhao), выпускник Кларка, показал, что model checking нашло бы ошибку в процессорах Pentium I и доказало бы верность её исправления корпорацией Intel. С тех пор корпорация Intel – одна из наиболее сильно использующих эту технологию.

Но так как даже маленькие программы могут иметь миллионы различных состояний (проблема известная в этой научной дисциплине как «комбинаторный взрыв»), есть предел размеру и сложности спроектированного программного обеспечения, которые может проверить model checking. Так что пока model checking не стал столь успешным для программного обеспечения, каким он стал для аппаратного. Верификация реагирующих систем – комбинации аппаратных и программных средств, взаимодействующих с экстремальным окружением – также остаётся проблематичной, в основном из-за сложности разработки верных математических моделей.

«Мы прошли большой путь за последние 28 лет, и теперь есть огромная разница в размере проблем, которые мы вынуждены решать по сравнению с 1980 годом», – говорит Хольцман – «но конечно мы более амбициозны и наши средства стали более точными и сложными, так что предстоит ещё много чего сделать.»

Другие технологии включают специализированные языки программирования, которые намного лучше подходят для создания надёжного и повторно используемого программного обеспечения. Например, Eiffel, разработанный Швейцарским Федеральным Институтом Технологий, под руководством Бертрана Мейера (Bertrand Meyer), получившего премию ACM в области программного обеспечения за 2006 год. Другой пример представляет Alloy, разработанный Дэниелом Джексоном и Группой Программного Проектирования МИТ (MIT Software Design Group), и доказавший свою работоспособность.

Вдобавок к новым языкам и методам, другие исследователи сфокусировались на методологиях и способах разработки программного обеспечения.

«Я не очень верю в формальный анализ», - говорит Гради Буч (Grady Booch) из IBM Research - «Проблемы имеют тенденцию возникать в том любопытном месте, где встречаются технологические и социальные аспекты». Например, после наблюдения за 50-ю разработчиками на протяжении 24 часов, Буч выяснил, что только 30 процентов их времени было потрачено на программирование – остальная часть времени была потрачена на разговоры с другими членами коллектива. Буч считает, что избегать непонимание в работе коллектива очень важно. Также Буч известен тем что разработал язык UML (совместно с Иваром Якобсоном (Ivar Jacobson) и Джеймсом Рамбо (James Rumbaugh). Язык UML представляет собой способ графического представления работы программы или аппаратной части в виде абстрактной модели, тем самым это помогает командам разработчиков лучше понимать друг друга, разрабатывать новые технологии, верифицировать новые конструкции и т.д. Не так давно, Гради продолжил свою работу над видением процесса разработки в онлайн справочнике «Handbook of Software Architecture», который включает в себя большую коллекцию преимущественно программного обеспечения и их характеристики и свойства, которые позволяют разработчикам использовать уже имеющийся опыт в своих проектах.

«Повторное использование проще при использовании более высоких уровней абстракции», - объясняет Буч. - «Так что мы повторно используем шаблоны, если нет необходимости использовать код снова».

Дэниел Джексон из МИТ является другим сторонником идей Буча. «Во-первых, мы должны убедиться в уровне надёжности, который нам необходим, а во-вторых мы должны подумать о том, что на самом деле никак не должно выходить из строя, о том, что важно, а что нет», – говорит Джексон.

На самом деле, разработчикам следует начинать не с типичных требований, описанных в бумагах в процедурном стиле, а им следует понять о чём вообще идёт речь, говорит Джексон. «Как можно построить надёжную систему, если вы не знаете что значит слово «надёжный»?» – спрашивает он.

И чем более приложение критично к ошибкам, тем более осторожным должен быть разработчик. «Если зависает ваш компьютер, это неудобно, но это не угрожает ничьей жизни», – говорит Хольцман. Среди его целей и его лаборатории, которая работает над программным обеспечением для космической программы, – найти простых, но в то же время эффективных методик программирования. Его предложения выглядят «драконовскими» (в критичных к безопасности приложениях они запрещают использовать goto, рекурсию и т.д.), но они призваны увеличить простоту, предотвратить программистские ошибки и стимулировать разработчиков к разработке более логичных архитектур. Более простые программы также легче верифицировать, с помощью того же *model checking*. После того как разработчики убеждаются в правильности такого подхода, они осознают что это хорошая плата за возросшую надёжность.

Если сосредоточиться на простоте, то можно получить большую прибыль, в особенности для сложных систем, которые будет очень сложно обновлять или заменять. Таким образом, поиск ошибок на начальном этапе разработки и верификация модели очень выгоден, в отличие от исправления ошибок на момент выпуска продукта. Этот урок уже усвоили многие компании, в том числе и Intel.

«Computer science – очень молодая наука», – объясняет Джозеф Сифакис (Joseph Sifakis), директор исследований в CNRS. «У нас нет теории, гарантирующей надёжность системы, которая может говорить нам как строить системы, надёжные с самого начала. У нас есть только несколько рецептов того, как приготовить хорошие программы и хорошее оборудование. Мы учимся на своих ошибках».