# К вопросу о стандартах и инструментах верификации «критического» программного обеспечения

Под понятием «программное обеспечение, критичное с точки зрения безопасности» (Safety-Critical Software – для краткости будем называть его «критичное программное обеспечение») обычно понимают такое программное обеспечение, которое влияет на поведение систем, сбой которых может повлечь риск для человеческих жизней. Иногда этим же термином называют программы, разработанные в соответствии со специальными стандартами, принятыми для критически важных областей.

Главным препятствием, стоящим на пути применения «коробочных» программных продуктов в критических областях, является недостаточная информация о «родословной» изделий, т.е. о процессах разработки, тестирования, верификации и управления качеством и т.п. [1]. Эту проблему в значительной мере решает как лицензирование деятельности предприятий, так и сертификация соответствующей продукции на соответствие определенным требованиям, сформулированным, как правило, в виде стандартов или иных руководящих документов. К функциональной безопасности программного обеспечения имеет отношение ряд международных и национальных, общих и отраслевых стандартов, регламентирующих жизненный цикл технических систем. Основные из этих стандартов подробно рассмотрены в работе [2]. Мы же рассмотрим такой важный аспект разработки критичного программного обеспечения, как тестирование.

Тестирование является одним из видов верификации и направлено на определение того, соответствует ли программное обеспечение требованиям спецификации [3]. Каким же образом можно понять, следует ли доверять проведенному тестированию или нет? ГОСТ Р ИСО/МЭК 15408 (известный под названием «Общие критерии») в отношении тестирования вводит понятия «Покрытие» и «Глубина». Покрытие показывает полноту охвата тестами (т.е. покрытие тестами) функциональных возможностей объекта оценки, а глубина характеризует уровень детализации тестирования. При оценке корректности тестирования за рубежом часто используют классификацию и требования, установленные в стандарте DO-178B «Software Considerations in Airborne Systems and Equipment Certification» («Оценки программного обеспечения для сертификации бортовых авиационных систем и оборудования»), известный в Европе как ED-12B.

DO-178B исходит из того, что при эксплуатации системы существуют потенциальные угрозы безопасности, возникающие при сбое программного обеспечения (например, останов двигателей самолета в полете), и возможных последствий такого сбоя. Сертификация программного обеспечения на соответствие DO-178B, по сути дела, заключается в проверках, как реализованы средства предотвращения угроз. Очевидно, что, с одной стороны, не всегда ошибка в программе приводит к сбою системы, с другой стороны, сбой системы может произойти по причине, независящей от программы. Поэтому сертификация программного обеспечения отдельно от системы, в состав которой она входит, не имеет смысла (например, некоторые аспекты влияния на операционные системы аппаратуры ЭВМ рассмотрены в работе [4]).

Исходя из возможных последствий сбоя, DO-178B определяет уровни опасности и соответствующие им уровни сертификации программного обеспечения (см. таблицу 1).

Таблица 1. Уровни сертификации программного обеспечения.

| Уровень | Влияние на безопасность |
|---|---|
| A | Программное обеспечение, сбой которого может привести к возникновению или способствовать возникновению катастрофического сбоя самолета. |
| B | Программное обеспечение, сбой которого может создать условия или способствовать созданию условий для опасного сбоя |
| C | Программное обеспечение, сбой которого может создать условия или способствовать созданию условий для крупного сбоя |
| D | Программное обеспечение, сбой которого может создать условия или способствовать созданию условий для малозначительного сбоя |
| E | Программное обеспечение, сбой которого не может влиять на самолет или на нагрузку пилота |

DO-178B требует, чтобы каждая строка кода была выполнена в ходе тестирования. Запрещается включение в состав приложения какого-либо дополнительного кода, не прошедшего тестирование. Для каждого уровня сертификации заданы свои требования к покрытию тестами проверяемого кода [5]:

1) Покрытие операторов (Statement Coverage – SC) означает, что в ходе тестирования каждый оператор программы был вызван или использован не менее одного раза. Когда говорят о покрытии кода – «Code coverage» – обычно имеют ввиду именно SC.

2) Покрытие ветвей (Decision Coverage – DC) означает, что в ходе тестирования каждая точка входа в программу и выхода из нее была использована не менее одного раза так, что каждое возможное значение логических условий принималось не менее одного раза. По сути дела, это означает, что в ходе тестирования каждое логическое условие имело и значение «истина», и значение «ложь».

3) Покрытие условий и ветвей (Modified Condition/Decision Coverage – MC/DC) означает, что в ходе тестирования каждая точка входа в программу и выхода из нее была использована не менее одного раза так, что каждое решение в программе принимало все возможные значения, и при этом было показано, какое влияние оказывает на решение каждое условие независимо от остальных условий. Для сложных логических операций необходимо разрабатывать таблицы истинности, что бы определить все возможные комбинации значений «истина» и «ложь».

В таблице 2 показано, какие из требований к покрытию кода тестами предъявляются на разных уровнях сертификации:

Таблица 2. Требования к покрытию кода.

| Уровень | Покрытие | Требование к покрытию |
|---|---|---|
| A | MC/DC | Уровень B + 100% MC/DC |
| B | DC | Уровень C + 100% DC |
| C | SC | Уровень D + 100% SC |
| D | | 100% покрытие требований |
| E | | Нет требований |

Наверное, трудно представить себе коммерческую разработку, к которой предъявляются требования по уровню E. Сертификация же программного обеспечения по уровню D имеет родственников в российской практике: приемо-сдаточные испытания и проверка соответствия реальных возможностей декларированным. Проверки по уровням A, B и C невозможны без специализированных инструментальных средств.

Разумеется, реализация требований DO-178B (впрочем, как и других нормативных документов) приводит к существенному увеличению стоимости продуктов, что связано со значительными затратами на разработку дополнительных тестов, проведения дополнительных процедур тестирования и на разработку дополнительной технической документации. В связи с этим на рынке программного обеспечения существуют инструментальные средства, помогающие автоматизировать процесс верификации программ. Такие инструменты могут поставляться как штатные компоненты в составе интегрированных сред разработки, так и в виде специализированных инструментов. В качестве примера первого типа можно указать перспективу (т.е. инструментальный профиль) «Code Coverage» интегрированной среды разработки QNX Momentics IDE, в качестве примера второго типа – среду верификации программ IPL Cantata++.

Momentics IDE поставляется компанией QNX Software Systems (http://www.qnx.com), разрабатывающей одноименное и, пожалуй, наиболее известное в России семейство операционных систем жесткого реального времени. Перспектива «Code Coverage», реализованная в этой IDE, на самом деле представляет собой графический интерфейс к специальному компоненту системы программирования GNU Compiler Collection (GCC) – gcov. При включенном режиме поддержки покрытия кода компилятор «инструментирует» генерируемый объектный код, т.е. добавляют в него инструкции, позволяющие собрать статистическую информацию о прохождении базовых блоков (т.е., грубо говоря, линейных фрагментов кода) при выполнении кода, а так же о ветвях выполнения. При этом генерируются дополнительные файлы данных, позволяющие прослеживать соответствие между базовыми блоками исполняемого кода и строками исходного кода [6]. Затем приложение выполняется обычным образом, в результате чего собирается статистика, сколько процентов кода выполнено во всем проекте, в каждом из файлов проекта, в каждой из функций, определенных в проекте. Кроме статистики можно по исходным кодам приложения посмотреть, какие из строк реально выполнились (рис 1).

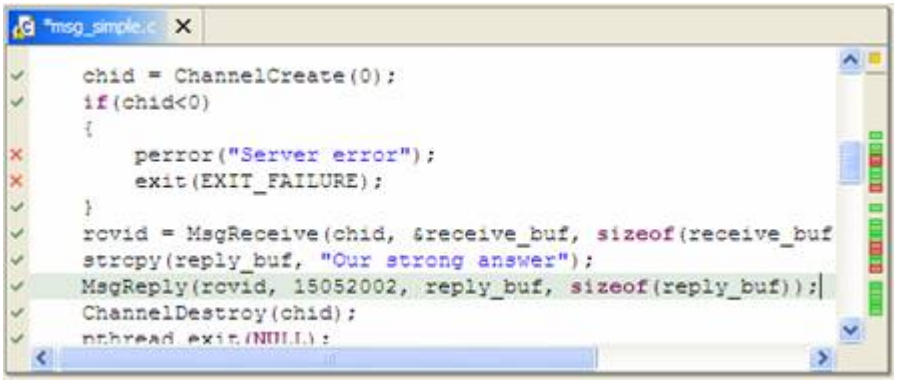


Рис. 1. Исходный код функции с маркировкой о прохождении строк в процессе выполнения программы.

К сожалению, в настоящий момент (мы говорим о QNX Momentics IDE версии 6.3.0 Service Pack 1) еще не реализована визуализация ветвей выполнения кода, хотя необходимая информация генерируется GCC [7] – эта возможность обеспечила бы DC-покрытие. Однако эта IDE и платформа Eclipse (http://www.eclipse.org), на которой она основана, развиваются весьма быстрыми темпами.

Cantata++, наряду с AdaTEST, один из наиболее известных коммерческих инструментов, специально предназначенных для анализа программного обеспечения критических систем, разрабатываемый британской компанией IPL Information Processing Ltd (http://www.ipl.com). Cantata++ тесно интегрирована со средами разработки Microsoft Visual Studio, Wind River Tornado, а так же рядом других [8], что позволяет использовать этот инструмент разработчикам достаточно широкого спектра систем. Например, для использования Cantata++ совместно с вышеупомянутой QNX Momentics IDE, необходимые компоненты добавляют в панель инструментов с помощью мастера подключения внешних средств (рис. 2).

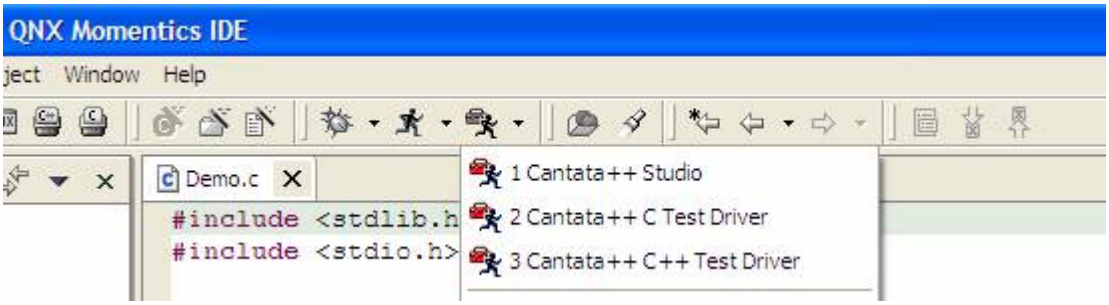


Рис.2. Доступ к компонентам Cantata++ из панели инструментов QNX Momentics IDE.

Для упрощения инструментирования приложений используются соответствующие мастера. Например, мастер «Code Coverage Wizard» позволяет задавать набор правил, позволяющий проводить анализ покрытия кода тестами, удовлетворяющий любому уровню сертификации по DO-178B (см. рис. 3).

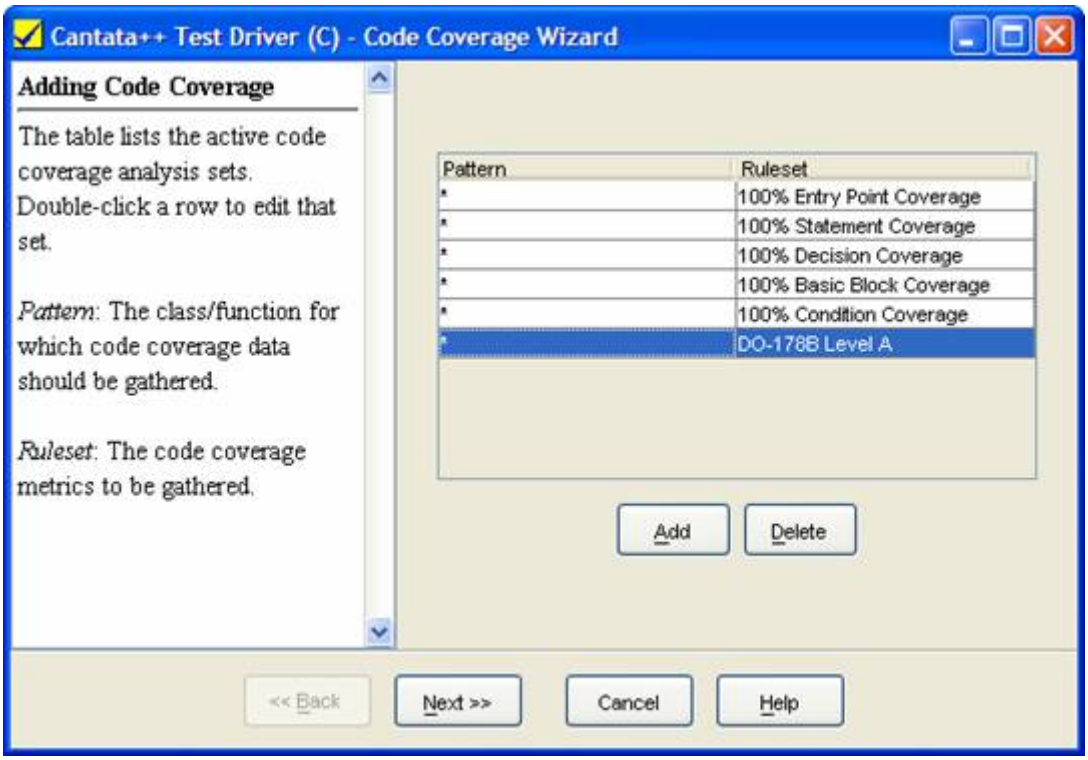


Рис. 3. Настройка инструментирования кода для анализа покрытия тестами.

После инструментирования соответствующих файлов проект пересобирают и проводят тестирование (его тоже можно автоматизировать с помощью Cantata++). Файлы с информацией, собранной в ходе выполнения приложения, анализируют в специальной среде Cantata++ Studio (Рис.4).

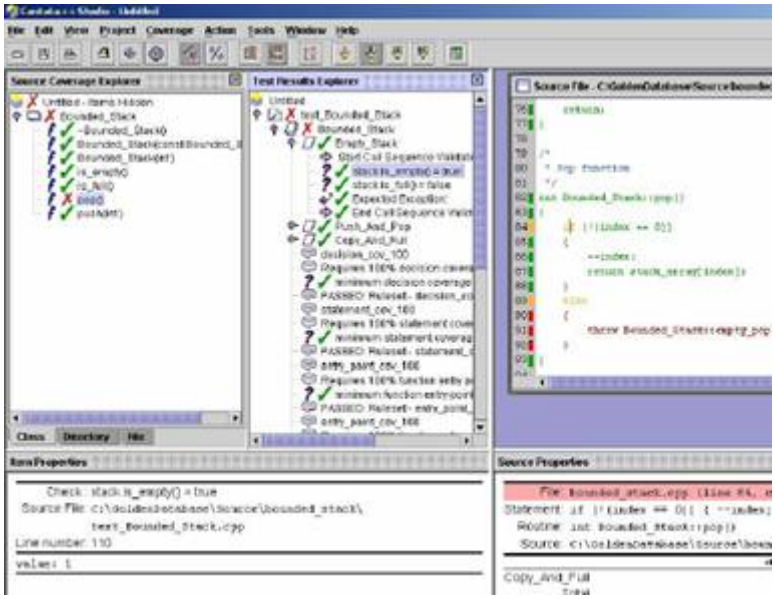

Рис. 4. Фрагмент среды Cantata++ Studio.

Разумеется, никакая «студия» не сможет заменить квалифицированного инженера. Однако к чести российских компаний надо сказать, что они все серьезнее относятся к использованию передового мирового опыта при разработках программного обеспечения не только для зарубежных заказчиков, но и для внутреннего рынка. Особенно в столь важной сфере, как разработка критических систем. Осмысляются и внедряются не только широко известные стандарты, относящиеся к качеству разработки программного обеспечения вообще (ISO 9001, SW CMMI), но и специализированные стандарты, посвященные различным аспектам безопасности (в первую очередь, пожалуй, IEC 61508, ISO 15408, ISO 12207). Поэтому возможно, что в нашей стране найдут применение и наукоемкие технологии, широко используемые за рубежом. К таким технологиям относится статический анализ программного кода (т.е. анализ кода без реального выполнения), без поддержки которого немыслим инструментарий анализа программ. Рассмотренная нами Cantata++ для анализа объектно-ориентированного кода поддерживает следующие группы метрик:

- MOOSE – метрики Чидамбера и Кемерера (Chidamber and Kemerer);
- MOOD – метрики Фернандо Брито (Fernando Brito e Abreu);
- QMOOD – метрики Бансии и Дэвиса (Bansiya and Davis);
- Объетно-ориентированные метрики МакКэйба (McCabe);
- Метрики объектно-ориентированных зависимостей Роберта Мартина (Robert Martin);
- Software Science метрики Халстида (Halstead);
- Цикломатические мертики МакКэйба, Майерса и Хансена (McCabe, Myers and Hansen);
- Метрики классовой энтропии Бансии.

Полученную статистическую информацию можно с помощью специального шаблона импортировать в Microsoft Excel для визуального изучения.

Вообще, верификация критического программного обеспечения и инструменты, предназначенные для этой цели, составляют достаточно обширную и интересную область инженерных и научных знаний. Надеюсь, этот очень краткий экскурс поможет как менеджерам, так и инженерам увидеть новые пути и возможности для решения стоящих перед ними повседневных задач.

Источники информации:

[1] Dave Emery. Safety Critical Software. Доклад на конференции «Real-time and Embedded Systems Forum», 2001. (Презентация доклада: http://www.opengroup.org/rtforum/jul2001/slides/emery-safety.pdf).

[2] В.В Липаев. Технологические процессы и стандарты обеспечения функциональной безопасности в жизненном цикле программных средств. http://www.jetinfo.ru/2004/3/1/article1.3.2004.html

[3] IPL. An Introduction to Software Testing. http://www.ipl.com/pdf/p0820.pdf (перевод на русский язык – http://www.kpda.ru/~zyl/articles/testing_intro.pdf)

[4] George Schreck. Development of real-time COTS systems: perils and pitfalls. http://www.thalescomputers.com/docs/perils&pitfalls-final-600.pdf (перевод на русский язык – http://www.kpda.ru/~zyl/articles/COTS_perils.htm)

[5] Chip Downing. A Primer on Safety Certification. http://validatedsoftware.com/docs/Cert_Primer.pdf

[6] Using the GNU Compiler Collection (GCC). http://gcc.gnu.org/onlinedocs/gcc-3.3.5/gcc/

[7] Integrated Development Environment: User's Guide (электронная документация, входящая в состав QNX Momentics Professional Edition)

[8] Cantata++ 4.0 Platform Availability Guide. http://www.ipl.com/pdf/p0005.pdf