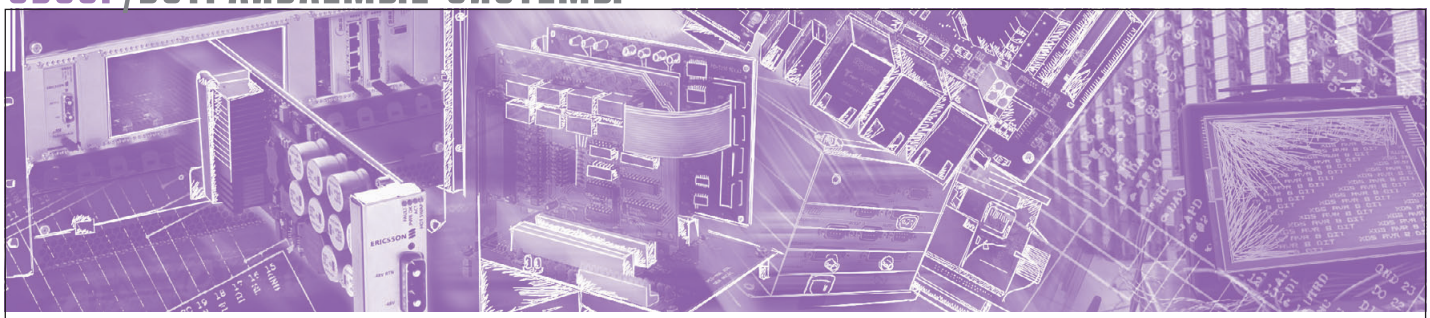

Пол Хендерсон, Джеймс Греннинг

Применение итерационных методик для тестирования встраиваемого ПО

В статье рассказывается о преимуществах итерационных методик разработки ПО, в частности, Agile, и о том, почему они востребованы на современном рынке. В качестве примера рассматривается модель тест-ориентированной разработки и сопутствующая технология непрерывной интеграции, а также приводится вариант их адаптации для индустрии встраиваемого ПО.

Введение

Множество ИТ-компаний на сегодняшний день уже внедрили итерационные методы разработки ПО и активно пользуются их преимуществами. Укороченный цикл проектирование–разработка–тестирование, свойственный итерационным методикам (Agile, scrum, extreme programming и т.п.), способствует более эффективному реагированию на изменения в требованиях, а также позволяет разработчикам оперативнее получать обратную связь по вопросам качества. Изменения в рабочем процессе разработчиков и тестировщиков, привносимые Agile и прочими итерационными методиками, дают менеджерам более прозрачную картину текущего состояния дел, что, в свою очередь, ведёт к более реалистичным рабочим планам и более предсказуемым графикам выпуска продукта. А поскольку основной упор в Agile делается на качество продукта, то, кроме сокращения сроков разработки и производственных издержек, его применение позволяет добиться также и снижения числа дефектов.

Однако при всех положительных качествах итерационных методик в индустрии встраиваемых приложений их внедрение явно запаздывает. Модульный дизайн, сложные технологии, кросс-разработка, необходимость тестирования на реальной системе и прочие особенности встраиваемого ПО – всё это создаёт на пути внедрения итерационных методик ощутимые трудности.

В настоящей статье рассказывается о преимуществах итерационных методик разработки ПО и о том, почему они востребованы на современном рынке. В качестве примера рассматривается модель тест-ориентированной разработки (Test Driven Development – TDD) и сопутствующая технология непрерывной интеграции (Continuous Integration – CI), а также приводится вариант их адаптации для индустрии встраиваемого ПО. Описываются типовые трудности, связанные с переходом к модели TDD, и стратегии, инструменты и техники, помогающие их преодолеть.

В более широком смысле статья содержит ряд полезных рекомендаций для коллективов о том, как можно применить имеющиеся наработки для адаптации к своему рабочему процессу новых методик, признанных многими коллегами в смежных областях как высокоэффективный инструмент повышения качества продукта и стабилизации рабочего графика.

Почему Agile?

Самая важная задача, стоящая перед разработчиками, тестировщиками и менеджерами, – это разрабатывать ПО максимально высокого качества, соблюдая баланс между сроками выпуска, функциональностью и затратами. Однако опыт показывает, что традиционные линейные методологии разработки ПО, известные как «водопад» и «V-модель», зачастую приносят прямо противоположный результат. Проекты неизбежно сталкиваются с текучкой требований и изменениями рыночных условий; в результате в самый разгар работы приходится принимать незапланированные решения. Как следствие, результат зачастую не оправдывает ожиданий: количество дефектов велико, часть функций пришлось исключить, бюджет превышен, сроки сдачи сорваны, клиенты недовольны.

Почему же линейный подход, казалось бы, десятилетиями служивший верой и правдой, сейчас создаёт разработчикам столько проблем? Ответ прост: линейные методики не обеспечивают достаточной гибкости, чтобы удовлетворять современным условиям разработки ПО, таким как

- растущие объёмы кода;
- часто меняющиеся требования;
- сложности интеграции и отладки;
- новые сложные аппаратные архитектуры;
- сжатые сроки тестирования.

(Справедливости ради следует заметить, что автор здесь немного передёр-

гивает, – эти проблемы так или иначе существовали всегда. «Священные войны» между сторонниками и противниками Agile идут постоянно, при этом по законам жанра и те и другие часто забывают, что любая модель может быть использована как во благо, так и во вред. В частности, «водопадная» модель действительно утопична, но что касается V-модели, в её определении вовсе не заложены пресловутые ригидность и бюрократичность, часто приписываемые ей сторонниками Agile. С другой стороны, Agile-методологиям тоже свойственны крайности, например, за Agile часто выдают программирование на основе здравого смысла, без письменных спецификаций – в результате код получается стабильный, но к требованиям не имеющий никакого отношения. Важно помнить, что Agile-методики ориентированы в первую очередь на проекты, в которых присутствует существенная текучка требований, в ответственных приложениях её уровень всегда гораздо ниже, чем в коммерческих. Универсальных методик разработки ПО не существует, и при выборе методики всегда нужно учитывать особенности проекта и заказчика. – *Прим. пер.*)

Линейные методики подразумевают последовательность чётко определённых фаз проекта: определение требований, концептуальное проектирование, кодирование, интеграция и, наконец, тестирование. Для начала каждой последующей стадии предыдущая должна завершиться полностью (или хотя бы преимущественно). При этом предполагается, что к началу проекта разработчикам известно всё необходимое, и тестирование достаточно провести в самом конце. Однако это предположение является ложным. (Опять же, это относится только к «водопадной» модели. Суть V-модели состоит не в жёсткой последовательности фаз, а в привязке требований к архитектуре, коду и тестовым сценариям. Всё это может меняться во времени, но только в связке, с сохранением целостности. Гибрид итерационной методики с V-моделью, позволяющий увеличить гибкость последней, называется спиральной моделью, или моделью Боэма. – *Прим. пер.*)

Реальное положение дел таково, что на момент начала проекта окончательного понимания задачи обычно ни у кого нет; архитектура, API и функциональность по ходу проекта постоянно изменяются, и это неизбежно влияет как на разработку, так и на тестирование. Как следствие, если отложить тестирование на потом, последствия будут печальны. Так что же делать?

Будем умнее

Гибкости, необходимой современному процессу разработки ПО, можно добиться применением итерационно-инкрементных методов. В сочетании с непрерывной интеграцией и соответствующими методиками тестирования они дают коллективам разработчиков и тестировщиков эффективный инструментарий для преодоления многих проблем процесса разработки, характерных для текущей ситуации в индустрии.

Agile-методология – одна из наиболее успешных в данном классе. В её рамках тесты разрабатываются сразу же, а процесс включает в себя множество итераций с непрерывным получением обратной связи. Ей также свойственны непрерывная корректировка планов и частое и ранее тестирование, что упрощает реализацию изменений, возникающих по ходу проекта. Тесты автоматизируются и могут выполняться повторно после каждой модификации кода, предупреждая о нежелательных последствиях.

Требуя меньше временны́х затрат на административные задачи и совещания, Agile-методология помогает разработчикам и тестировщикам уделять больше времени своей основной работе. Она способствует более эффективному взаимодействию и позволяет принимать решения быстро и обоснованно. В результате работа не только движется в нужном направлении, но и становится интереснее.

Agile-проект начинается с разбиения большой задачи на более мелкие и поэтому более управляемые составные части (обычно это некие законченные подмножества функциональности). При этом «большая картинка» всё время остаётся на виду, но в своих рабочих планах участники проекта могут сосредоточиться на конкретных небольших фрагментах. Это существенно увеличивает шансы выполнения функциональных требований с надлежащим качеством и в намеченные сроки.

В командах, работающих по Agile-методологии, разработчики и тестировщики тесно сотрудничают с самого начала. Они совместно разрабатывают наборы тестов, которые бы независимо верифицировали каждый элемент функциональности. В отличие от традиционного взгляда на тестирование, такой подход выделяет его активную роль в процессах разработки и управления требованиями, сдвигая основной фокус с обнаружения дефектов постфактум на динамическую коррекцию требований и профилактику.

Командная работа и итерации

Фрагменты функциональности в рамках Agile-проекта обычно называют пользовательскими историями, или просто историями (оригинальные термины звучат как story/ user story. – *Прим. пер.*). Когда становятся понятны все истории, составляющие проект, члены команды могут отсортировать их, скажем, по оценке трудозатрат. Оценка обычно производится в безразмерных баллах (story points): 1 балл назначается наименее трудоёмким историям, а 5 – тем, трудоёмкость которых в 5 раз выше. План составляется на основе оценки «пропускной способности» команды, измеряемой в баллах на одну итерацию, и расстановки приоритетов (исходя из требований клиента).

Такое ранжирование позволяет разделить истории на обязательные к реализации (must-have), желательные (should-have) и т.д., и на основании этого уже принимать взвешенные решения о том, какие истории следует включать в итерацию, а какие нет. После нескольких итераций оценочная «пропускная способность» команды заменяется на реальную и позволяет судить о том, сколько работы может быть выполнено в последующих итерациях. Это обеспечивает процессу планирования необходимую обратную связь, делая планы более реалистичными и заранее предупреждая о возможных сдвигах графика.

История считается реализованной, только когда тестирование по ней успешно завершено. Базовый Agile-процесс подразумевает сперва написание функционального теста (story test) для каждой истории, над которой ведётся работа, – это позволяет чётче понимать, что конкретно история должна делать. Когда этот тест успешно пройден, история признаётся завершённой. Поскольку код – очень хрупкая субстанция, функциональные тесты автоматизируются; это сводит усилия по их повторному выполнению практически к нулю и позволяет повторять их регулярно после каждого изменения, обнаруживая дефекты незамедлительно и предотвращая их повторное появление. Такую практику часто называют разработкой на базе функциональных тестов

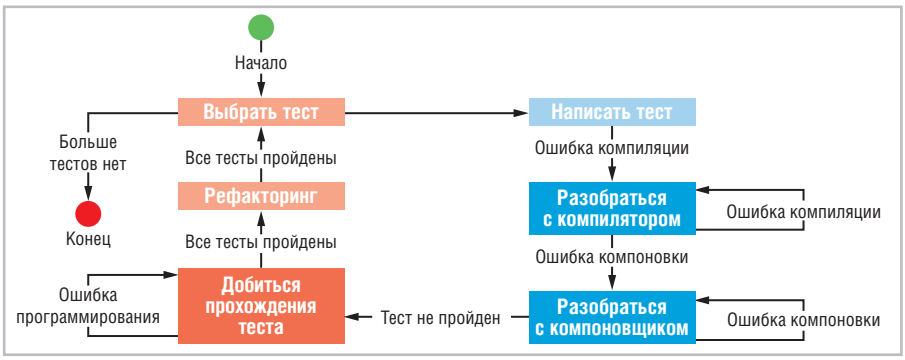

Рис. 1. Цикл обратной связи: код и модульные тесты

(acceptance test-driven development, или, в общем случае, просто test-driven development – TDD).

На уровне кода при этом используется разработка на базе модульных тестов (unit test-driven development). Это значит, что разработчики действуют в коротком цикле обратной связи, когда модульный тест пишется непосредственно перед разработкой программного модуля, который ему надлежит проверять (рис. 1). В процессе история может компилироваться некорректно или не проходить тест (хорошей практикой считается намеренное внедрение ошибок в код, чтобы убедиться, что тест способен их обнаружить); программист исправляет код до прохождения им теста, а затем обычно проводит рефакторинг (доработка кода, не затрагивающая его поведение, но улучшающая его логичность и прозрачность для понимания. – *Прим. пер.*). Затем цикл повторяется, и так до тех пор, пока не будут пройдены все модульные тесты, достаточные для того, чтобы пройти функциональный тест.

Чтобы процесс TDD работал корректно, необходимо, чтобы каждый модульный тест соответствовал 5 критериям, описываемым мнемонической аббревиатурой FIRST (каламбур: в оригинале “must be FIRST”, то есть «должны быть сначала». – *Прим. пер.*), введённой экспертами в Agile-методологии Бреттом Шухертом (Brett Schuchert) и Тимом Оттингером (Tim Ottinger):

- Fast (быстрый) – скорость выполнения тестов должна составлять сотни тысяч тестов в секунду;
- Isolated (изолирующий) – тесты должны однозначно локализовывать ошибки;
- Repeatable (повторяемый) – тесты должны давать одинаковый результат при повторении их в любое время и в произвольном порядке;
- Self-validating (однозначный) – результат теста не должен требовать никакого анализа и не должен допускать трактовок – только пройден/не пройден;
- Timely (своевременный) – тесты должны разрабатываться непосредственно перед написанием проверяемого кода.

Лучше предсказуемость, быстрее поставка

Agile-разработка – непрерывный процесс с постоянным взаимодействием участников и измеримым прогрессом. Разработчики и тестировщики непрерывно работают вместе, ранжируя истории, демонстрируя друг другу результаты, измеряя степень завершённости задач, интегрируя готовые компоненты и т.п. Такой подход оставляет гораздо меньше тестирования и стресса на конец проекта, когда накал страстей максимален.

Традиционный линейный подход к разработке ПО даёт ощущение поступательного движения, но на самом деле это иллюзия (рис. 2), так как основной хаос кроется как раз в завершающей фазе интеграции и тестирования. Несмотря на то что TDD может показаться более извилистым путём к финишной прямой, на самом деле он более безопасный и предсказуемый и менее затратный.

С позиции команды преимущества Agile и TDD включают в себя:

- концентрацию на том, что должен делать каждый элемент функциональности и какие сценарии отказов необходимо проверять;
- профилактику дефектов, поскольку элементы функциональности тестируются по мере их разработки; автоматизация тестирования позволяет мгновенно обнаруживать побочные эффекты и исправлять ошибки по горячим следам;
- возможность избежать дорогостоящих сюрпризов на заключительных фазах проекта, когда проблемы внезапно разрастаются как снежный ком и проект резко теряет предсказуемость.

С точки зрения руководства компании, Agile и TDD обеспечивают:

- улучшение качества программного продукта;
- улучшение предсказуемости графиков выпуска;
- ускорение процесса разработки при одновременном снижении затрат и рисков;
- увеличение степени соответствия требованиям заказчика с первой попытки;
- увеличение удовлетворённости и лояльности клиентов;
- увеличение степени удовлетворённости работой, в отличие от классических «маршей смерти» (термин не случаен – см. книгу Эдварда Йордона «Путь камикадзе: как разработчику программного обеспечения выжить в безнадёжном проекте». – *Прим. пер.*).

Особенности встраиваемых приложений

От разработчиков встраиваемого ПО часто можно услышать, что TDD производит впечатление интересного подхода, но в индустрии встраиваемых систем всё немножко сложнее.

В классической ИТ-индустрии разработчики и тестировщики могут спокойно предполагать, что разрабатываемые ими продукты будут работать на стандартной стабильной аппаратуре и взаимодействовать со стандартными отлаженными программными компонентами. Во встраиваемых приложениях, напротив, практически все уровни ПО либо разрабатываются заново, либо подвергаются модификации, причём всё это может происходить параллельно.

Более того, встраиваемое ПО должно тестироваться на реальной целевой ап-

Рис. 2. Предотвращение «сюрпризов» на финальных стадиях

паратуре, а она может быть «сырой» и ограниченно доступной. Таким образом, даже если истории легко тестируются обособленно, их нельзя считать завершёнными, пока они не пройдут тесты в «боевых» условиях, то есть в окружении реального целевого ПО и на реальном оборудовании. Раннее тестирование, предусмотренное Agile-методологией, помогает избавиться от части проблем заранее, сокращая объёмы более трудоёмкого тестирования и отладки на реальном оборудовании.

Условия окупаемости TDD-методики

Чтобы адаптировать подход TDD к индустрии встраиваемых приложений, командам разработчиков необходимо локализовать узкие места, которые могут свести на нет преимущества использования TDD, и найти способы их избежать. Также требуется внедрение необходимой автоматизации, чтобы сделать итерации как можно более короткими и менее утомительными. Основные рекомендации по внедрению TDD в индустрии встраиваемых приложений можно свести к следующему.

- **Формализуйте процесс модульного тестирования.** Нельзя отдавать его на волю случая. Используйте инструменты автоматизированного тестирования, либо свободно распространяемые, либо более сложные коммерческие, наподобие IPL Cantata++ для Wind River Workbench, чтобы сделать процесс быстрым и повторяемым. Такие инструменты предоставляют готовый каркас для построения тестов, включая «обёртки» (wrappers – код, вызывающий тестируемый модуль с нужными параметрами и перехватывающий возвращаемый результат. – *Прим. пер.*), «заглушки» (stubs – код, используемый для имитации ещё не разработанных программных модулей; принимает такие же входные параметры, возвращает результаты корректного типа, но не производит никаких реальных действий, используется для формирования необходимого окружения для тестируемого модуля. – *Прим. пер.*), таблицы входных и ожидаемых выходных значений и прочие компоненты для формализации тестов. Каждый из этих компонентов можно многократно повторять и оттачивать в процессе рефакторинга, а затем сохранять готовые модульные тесты для повторного использования в автоматическом режиме вместе со всеми остальными: функциональными, системными и интеграционными тестами.
- **Используйте по максимуму инструментальные компьютеры.** Учитывая, что реальное целевое оборудование зачастую бывает в дефиците, особенно на начальных стадиях проекта, большинство команд использует для тестирования свои рабочие станции. Это выигрышная стратегия, но она таит в себе риски, так как тесты могут компилироваться и выполняться на инструментальной и целевой системах по-разному. Окончательное тестирование в любом случае должно выполняться на целевой системе, но промежуточные шаги помогут избежать проблем на стадии интеграции. Сначала удостоверьтесь, что ваш код является многоплатформенным, чтобы быть уверенным, что он будет работать и на инструментальной машине, и на целевой. Чтобы добиться этого, необходимо знать особенности компиляторов, библиотек, оборудования и ОС для обеих сред. Также переносимости кода способствует разделение системно-зависимого и системно-независимого кода по разным модулям – это поможет как с точки зрения хорошего стиля программирования, так и с позиции TDD. Переносимость кода обеспечивает бóльшую устойчивость технического решения и облегчает последующий перенос на целевую архитектуру.
- **Проводите частое тестирование в среде, совместимой с целевой.** Добившись работоспособности кода на инструментальной системе, необходимо проверить его работоспособность и на целевой системе тоже. Это можно делать прямо на инструментальном компьютере с использованием симуляторов или эмуляторов, таких как Wind River Simics, VxWorks VxSim или Linux QEMU. В идеале кросс-платформенные истории должны заработать на них сразу же, но повторное выполнение формализованных модульных тестов часто позволяет находить новые ошибки сразу же, пока ещё не прошло много времени. Если код не работает, данные среды предоставляют исчерпывающую поддержку диагностических и отладочных инструментов, чтобы локализовать и исправить ошибки. Затем по мере доступности целевого оборудования тесты желательно повторить ещё раз – уже в реальной среде (рис. 3).
- **Используйте процесс непрерывной интеграции.** Поскольку в проектах по разработке встраиваемого ПО несколько групп зачастую работают параллельно над программными компонентами разного уровня, важно начать тестировать эти компоненты вместе как можно раньше и делать это как можно чаще. Вместо того чтобы откладывать интеграцию до конца цикла, необходимо проводить её непрерывно. Непрерывная интеграция (Continuous Integration – CI) является сопутствующей практикой TDD (рис. 4). Она подразумевает, что участники проекта регулярно интегрируют внесённые изменения и сохраняют их в системе управления версиями, после чего уже на интегрированном наборе модулей выполняется регрессивное тестирование. Обычно в Agile-командах серверы непрерывной интеграции настроены на автоматическую перекомпиляцию (rebuild) проекта по каждому факту внесения изменений (check-in), таким образом весь коллектив старается постоянно поддерживать систему работающей и поступательно развивающейся. Задержки интеграции влекут за собой увеличение числа проблем по мере того как подсистемы модифицируются и развиваются; регулярность позволяет избежать авральной интеграции в конце проекта, когда разработчики внезапно понимают, насколько они реально далеки от создания работающей системы. CI можно сперва реализо-

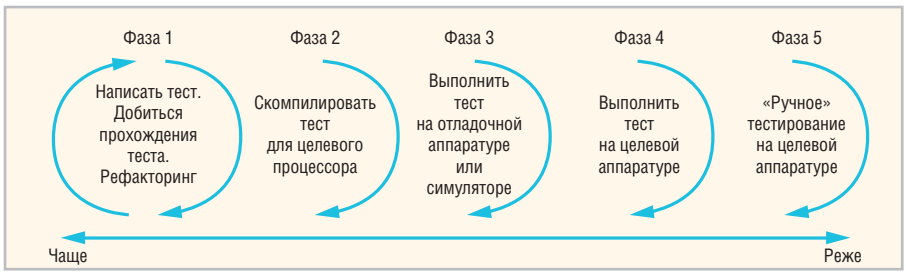

Рис. 3. Повторное выполнение регрессивных тестов в разных средах

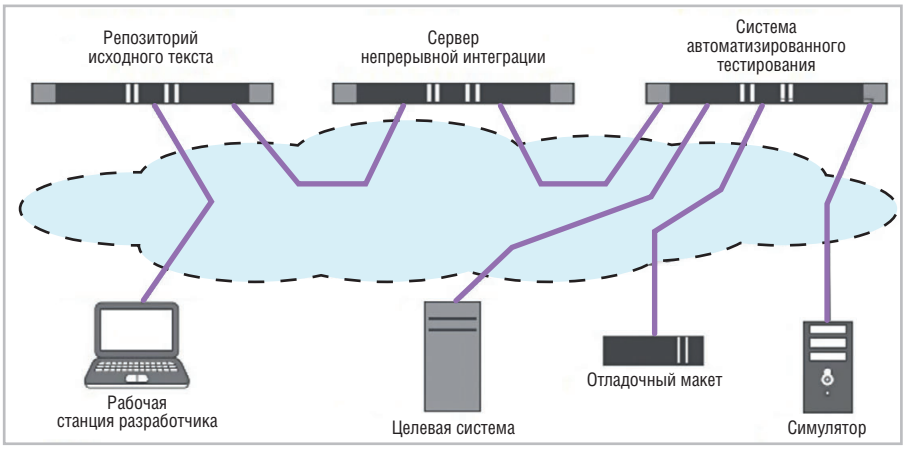

Рис. 4. Непрерывная интеграция

вать на программном уровне с использованием симуляторов, работающих в инструментальной среде, а затем, когда станет доступно целевое оборудование, перенести в целевую среду или продолжить работу в обеих средах параллельно.

- **Поддерживайте проект в постоянной готовности к тестированию.** Истинный Agile-процесс требует частого тестирования. Обычно первым в CI-процессе является тест на то, успешно ли прошла компиляция проекта. После этого уже можно последовательно выполнить все необходимые модульные, функциональные и интеграционные тесты, разработанные командой тестировщиков. Продолжительность циклов тестирования будет разной в зависимости от типов используемых тестов и их положения в жизненном цикле проекта: на ранних этапах они более часты, но более коротки; на более поздних и более сложных интеграционных стадиях – длиннее и реже. Однако, вне зависимости от длительности цикла тестирования, чтобы в полной мере пользоваться плодами TDD и CI, система автоматизированного построения проекта и регрессивного тестирования абсолютно необходима. Вы не сможете воспользоваться этими методиками, если будете полагаться на ручной труд, – Agile-методология требует автоматизации.

Тестирование встраиваемых систем с помощью Wind River Test Management

Чтобы безболезненно перейти к использованию Agile-методологии, используемый инструментарий автоматизированного тестирования должен быть способен хранить и повторно выполнять тесты по мере рефакторинга и валидации кода в различных средах. Он также должен предоставлять информацию, помогающую оценить, насколько выпускаемый программный продукт соответствует установленным критериям качества, и если выявляются проблемы, то вовремя предпринять необходимые действия.

Wind River Test Management – инновационная система автоматизированного тестирования качества ПО, разработанная специально для встраиваемых приложений. Она позволяет автоматизировать значительное количество операций по модульному, регрессивному, системному и интеграционному тестированию, освобождая больше времени для основных задач разработки ПО.

Wind River Test Management – современная масштабируемая система с веб-интерфейсом, позволяющая создавать, хранить, управлять, выполнять и анализировать результаты всех тестов, разработанных в рамках TDD-процесса. Она избавляет от трудоёмких накладных расходов, связанных с ранним и частым тестированием. Будучи построенной на открытых стандартах, Wind River Test Management легко интегрируется с существующими системами компиляции/сборки, управления качеством и инструментарием разработки, принятыми в организации.

Однако Wind River Test Management – больше, чем просто система автоматизированного тестирования. В ней реализована уникальная технология динамического инструментирования, позволяющая измерять степень покрытия кода, выполнять трассировку от тестов к коду, а также проводить профилирование и тестирование в режиме «белого ящика». Предоставляя детальную видимость происходящего в «начинке» работающей системы, Wind River Test Management снабжает разработчиков, тестировщиков и менеджеров точными оценками качества ПО в любой точке процесса. Наличие этой информации позволяет производить программный продукт лучшего качества в более короткие сроки и с меньшими затратами.

Готовая платформа для TDD

Wind River Test Management предоставляет современную среду для тестирования встраиваемого ПО, реализующую высокий уровень автоматизации и обратной связи, необходимый для организации продуктивного TDD-процесса. Рассмотрим наиболее важные аспекты того, как Wind River Test Management помогает командам эффективно реализовать и использовать TDD.

Хранение и повторное использование тестов

Тесты, разрабатываемые участниками проекта, представляют собой ценную интеллектуальную собственность, которую можно и нужно использовать в задачах по улучшению качества, причём использовать неоднократно.

Wind River Test Management упрощает задачу централизованного хранения и структурирования тестов по мере их создания. Впоследствии на основе имеющегося репозитория можно составлять и выполнять целевые наборы тестов, а также управлять тестовыми прогонами и делиться результатами тестирования с коллегами.

Автоматизация регрессивного тестирования

По мере реализации проекта встраиваемого ПО с использованием TDD-методики число завершённых историй и связанных с этим операций интеграции постоянно растёт, что, в свою очередь, увеличивает объём и сложность тестирования. Согласно признанным практикам, это тестирование необходимо проводить как на инструментальных компьютерах, так и в симулируемой и реальной целевой среде.

Wind River Test Management позволяет разработчикам и тестировщикам избавиться от трудоёмких ручных операций, автоматизируя выполнение тестовых последовательностей, – достаточно просто выбрать нужные сценарии. Масштабируемое распределённое ядро Wind River Test Management поддерживает как интерактивные, так и пакетные операции. Wind River Test Management также существенно сокращает время анализа результатов, поддерживая

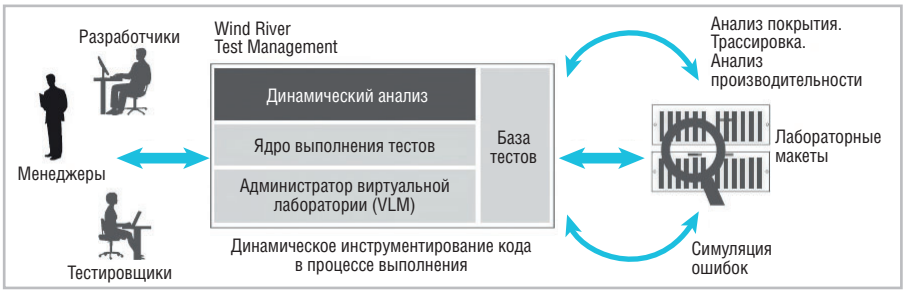

Рис. 5. Организация TDD-процесса с помощью Wind River Test Management

централизованное хранилище результатов тестирования и сводных отчётов.

Управление пулом целевых систем

В любой организации, занимающейся разработкой ПО, получить и сконфигурировать необходимые ресурсы для проведения тестирования всегда непросто. Если ожидать доступности ресурсов по несколько часов, а то и дней, то весь TDD-процесс пойдёт насмарку.

Встроенный так называемый администратор виртуальной лаборатории (Virtual Lab Manager – VLM) помогает пользователям Wind River Test Management избежать этих проблем. С помощью VLM разработчики и тестировщики могут организовать использование всех необходимых тестовых ресурсов даже в самых больших организациях. VLM позволяет упорядочивать и резервировать устройства, платы, симуляторы, отладочные макеты, серверы и любое другое тестовое оборудование, а также получать к нему удалённый доступ, загружать ПО и т.п., а затем по окончании работы возвращать обратно в пул доступных ресурсов. VLM тесно интегрирован с модулем выполнения тестов, чтобы тестировщики могли выбрать для прогона необходимых сценариев оптимальные целевые системы, как физические, так и виртуальные. VLM помогает значительно упростить процесс управления ресурсами и, как следствие, повысить эффективность работы команды тестировщиков.

Фокусирование на коде, ещё не прошедшем тестирование

Учитывая высокую частоту итераций, принятую в рамках TDD, и высокую интенсивность внесения изменений в код, одной из наиболее важных задач для тестировщиков становится поспевание за новинками, регулярно появляющимися в свежих сборках. Большое количество деталей способно рассеять внимание и отвлечь усилия от основной задачи – тестирования именно того кода, в который были внесены изменения.

Wind River Test Management отслеживает привязку тестов к коду даже в процессе ручного тестирования, а также автоматически анализирует новые сборки на предмет того, какой код был добавлен, изменён или удалён. Эта информация впоследствии используется, чтобы определить, какие тесты требуют повторного выполнения после модификации кода или рефакторинга. Это позволяет команде сфокусироваться на том, что реально требует тестирования, а значит, увеличить продуктивность и повысить качество обратной связи.

Оценка качества тестирования

Учитывая свойственную встраиваемым системам сложность, часто бывает трудно определить, в достаточной ли мере было проведено тестирование. Причиной тому служит традиционный подход к тестированию, когда тестируемое устройство рассматривается с позиции «чёрного ящика» (black box), а видимость его внутреннего устройства отсутствует.

Способности Wind River Test Management к динамическому анализу позволяют тестировщикам перейти к концепции «белого ящика» (white box; термин, вероятно, был введён для красного словца – на самом деле более корректным здесь был бы термин «прозрачный ящик». – *Прим. пер.*), наблюдая за поведением «внутренностей» устройства в процессе тестирования (рис. 5). Способность видеть тестируемое устройство «насквозь» позволяет дополнительно исследовать временны́е характеристики выполнения кода, симулировать ошибки в реальном времени, наблюдать реакцию системы и т.п., иными словами, выполнять гораздо более детальную и глубинную диагностику.

Взаимодействие и координация

Чтобы подход TDD принёс ожидаемые плоды, необходимо, чтобы разработчики, тестировщики и менеджеры работали более тесно и слаженно, чем в рамках привычных методологий.

Wind River Test Management обладает масштабируемой распределённой архитектурой, которая обеспечивает командам разработчиков и тестировщиков единый интерфейс. Взаимодействие осуществляется посредством веб-приложений, а также через инструменты разработчика, как реализованные в виде плагинов Eclipse, так и командно-строковые. Вне зависимости от физического расположения и часового пояса пользователи Wind River Test Management будут иметь доступ к системе и смогут делать свою работу, делиться результатами с коллегами и способствовать стабильному продвижению проекта в правильном направлении.

Заключение

Рынок встраиваемых систем сильно изменился, и вместе с ним изменились ожидания клиентов. Клиентам необходимы инновационные и качественные продукты, причём вовремя и по адекватной цене; при этом они оставляют за собой право корректировать требования «по ходу пьесы», в том числе, когда проект уже в самом разгаре.

Agile-методики, включая TDD и CI, помогают разработчикам добиться повышения качества продукта и улучшения продуктивности. Чтобы использовать эти методики, нужны средства автоматизации тестирования, снабжающие разработчиков необходимой обратной связью. Современные программные системы наподобие Wind River Test Management, оптимизированные для тестирования встраиваемых приложений и способные анализировать их поведение в динамике, выходят за рамки традиционных подходов, помогая разработчикам и тестировщикам справляться с потоком изменений и концентрироваться на том, что действительно требует внимания, сохраняя конкурентоспособность в динамично меняющейся среде. ●

П. Хендерсон – вице-президент по маркетингу, направление систем тестирования ПО, Wind River
Д. Греннинг – основатель Renaissance Software Consulting
Перевод Николая Горбунова, сотрудника фирмы ПРОСОФТ
Телефон: (495) 234-0636
E-mail: info@prosoft.ru