**Три уровня автоматизации тестирования**

Условно, тестируемое приложение можно разбить на 3 уровня:

* **Unit Tests Layer**
* **Functional Tests Layer (Non-UI)**
* **GUI Tests Layer**

Для обеспечения лучшего качества продукта, рекомендуется автоматизировать все 3 уровня. Рассмотрим более детально стратегию автоматизации тестирования на основе трехуровневой модели:

**Уровень Unit Test layer**

Под автоматизированными тестами на этом уровне понимаются Unit Tests, написанные разработчиками. Также, никто не запрещает тестировщикам писать тесты, которые будут проверять код, конечно же, если квалификация тестировщиков позволяет это. Наличие подобных тестов на ранних стадиях проекта, а также постоянное их пополнение новыми тестами, проверяющими «баг фиксы», убережет проект от многих серьезных проблем.

**Уровень Functional Test Layer (non-ui)**

Как правило не всю бизнес логику приложения можно протестировать через GUI слой. Это может быть особенностью реализации, которая прячет бизнес логику от пользователей. Именно по этой причине по договоренности с разработчиками, для команды тестирования может быть реализован доступ напрямую к функциональному слою, дающий возможность тестировать непосредственно бизнес логику приложения, минуя пользовательский интерфейс.

**Уровень GUI Test Layer**

Под внешним слоем мы понимаем пользовательский интерфейс, средства автоматизации которого были описаны выше. На данном уровне есть возможно тестировать не только интерфейс пользователя, но и бизнес слой, выполняя операции вызывающую бизнес логику приложения. С моей точки зрения, такого рода сквозные тесты дают больший эффект нежели просто тестирование бизнес слоя, т.к. мы тестируем бизнес слой эмулируя действия конечного пользователя.

**Архитектура Автоматических Тестов**

Взяв за основу основные **поля** [**тест кейса**](http://alexeybulat.blogspot.com/2008/02/test-case-writing.html) - Precondition, Steps & Post Condition, переносим их и в тест скрипт.

Получаем правило, что каждый скрипт должен иметь:

* Precondition
* Steps
* Post Condition

Перечислим основные функции скрипта:

1. Precondition

* Инициализация приложения (например, открытие главной страницы, логин и переход в необходимую часть приложения и подведение системы к состоянию пригодному для тестирования)
* Инициализация тестовых данных

2. Steps

* Непосредственное проведение теста
* Занесение данных о результате теста, с обязательным сохранением причин провала и шагов, по которым тест прошел

3. Post Condition

* Корректное завершение работы приложения

Рекомендуется создать общую библиотеку по обработке ошибок и исключительных ситуаций. Пример:

* PreConditionException
* TestCaseException
* PostConditionException

В итоге, воспользовавшись вышеописанными рекомендациями, у вас есть общая структура каталогов, общая архитектура скриптов и сценариев, общая библиотека обработки ошибок и исключений.

**Стратегия использования автоматизированных тестов**

Для обеспечения лучшего качества продукта, я рекомендую следующий подход:

1. Написанием тестов должны заниматься «специально обученные люди» - Automation Testers. После написания, тесты передаются команде ручного тестирования, которая уже осуществляет их ежедневный запуск и анализ результатов. Тем самым автоматизированные тесты также проходят тестирование, и в результате увеличивается их надежность и жизнеспособность.
2. Написанные и отлаженные тесты также могут передаваться команде разработки, для отладки новых версий.
3. Команде разработки рекомендуется осуществлять ежедневную сборку, с прогоном всех написанных тестов на все «3 уровня». И только после того, как новая версия начинает удовлетворять критериям качества, осуществлять установку новой версии на QA платформу.

Написание и подход к автоматизации тестирования зависит от процесса разработки приложения. Взяв за основу [RUP (Rational Unified Process)](http://alexeybulat.blogspot.com/2008/01/rup-project-process.html), могу предложить следую схему, разбитую на фазы:

**Inception phase** – выбор инструмента автоматизации

**Elaboration phase** – написание тестов на основную архитектуру (в дальнейшем эти тесты будут использоваться для прима билда – Build Verification Tests)

**Construction phase** – более детальная автоматизация: критическая функциональность, проверка регрессий, end-to-end сценарии

**Transition phase** – подготовка тестов к передаче заказчику (если это требуется)

**Файловая структура тестовых скриптов и сценариев**

Во избежание проблем с использованием и сопровождением инструмента, тестовых скриптов и сценариев рекомендуется принять корпоративный стандарт для структуры каталогов (packages), стандартов к написанию кода (для java тестов – java code convention), а также общую архитектуру создания скриптов и сценариев.

Рекомендовано создать следующую файловую структуру:

Project/

* bin – папка для хранения запускных файлов
* lib – папка для хранения сторонних библиотек
* conf – папка, содержащая свойства и параметры всех тестов
* src – папка с исходным кодом тестов
* results – папка, куда будут записываться результаты прогона.

Причем аналогичная структура будет удобна при использовании любого выбранного инструмента, с некоторыми доработками.

Пример, организации файловой структуры, для тестов на HP (Mercury) WinRunner:

Project/

* API – аналог /lib
* Scripts – аналог /src
* GUI
* Tables
* Files

Папки GUI, Tables & Files – это разбитая на части папка /conf