QA Automation Tool Requirements

The purpose of this document is to describe the QA requirements for selecting automation tools.

General Test Tool Requirements

This section describes general requirements that should be applied when selecting any QA automation tool.

Must Haves

- Budget
  - cost effective - what is QAs budget?
- Ease of Use
  - QA team can come up to speed quickly given current skill set
    - Test Script languages
      - preferred - Java
      - some experience - C#
      - Javascript?
    - IDEs
      - preferred - Eclipse
      - some experience - Visual Studio
- Test Tool(s) Roadmap
  - tools/libraries should be supported now and in the foreseeable future
- Customer Support
  - Phone or online ticket support
  - online help resources
  - user manual
  - [POH or an active user community. Many interesting tools are open source, so there's no help desk, but if there's an active community, then StackOverflow, etc. can be even better than a help desk.]
- Ability to integrate the tool with Azure DevOps CI/CD pipelines
- Reporting Mechanism
  - Convenient data capture and reporting (e.g. Screenshots, Video Recording, Text Logs)
    - at least on par with SilkTest TrueLog file but preferably more robust
  - report formats and coverage formats consumable by current reporting tools
    - Xray supports the following formats for importing test execution results to JIRA: JUnit XML output format, NUnit XML output format, xUnit XML output format, TestNG XML output format, Robot Framework XML output format, Xray JSON format, Cucumber JSON output format, Behave JSON output format
      - Import Execution Results - REST - Xray Cloud Documentation - Xray
    - Azure DevOps supports the following format for publishing test results: JUnit, NUnit 2, NUnit 3, Visual Studio Test (TRX), XUnit 2, CTest
      - PublishTestResults@2 - Publish Test Results v2 task
- Ability to generate statement coverage reports in a popular format for the code being exercised. (This may be provided by a mechanism other than the test runner itself, however coverage must be collectible.)
  - @Peter Harris *Is this the same as this?: Results reportable in popular standardized formats such that we can aggregate/analyze including over-time analysis. (Choosing a popular/standardized format will allow us to have other tools also report in those formats*

allowing aggregation across multiple areas of testing to use a singular tool for analysis) POH: almost. This one is a general statement. Mine is about coverage.

- Support connecting to different data sources
  - Database
  - Data files (CSV, Excel, etc…) for BYOD approach
- Integration with other Test Tools
  - JIRA
    - ability to update the status of a ticket
- Parallel, cross-device and cross-browser testing support
  - Platform agnostic: Web apps and mobile apps can be automated from the same platform
  - Parallel test execution: QA can execute our tests in multiple environments simultaneously
  - device support: mobile, web
  - web browser support: supports Chrome and Microsoft Edge (at a minimum)
  - mobile platform support: Android and IOS
- Ability to categorize/group tests easily for efficient run-time execution, reporting and analysis
  - Example: NUnit 3 offers multiple ways to categorize tests, such as:
    - the ability to filter tests by Class Name
    - ability to apply a Category attribute to each test

Nice To Haves

- Change Impact Analysis
  - Identification of tests to prioritize based on code changes and risk.

API Test Tool Requirements

Must Haves

- supports the following testing types: functional testing (Business Logic layer)
- supports the following levels of testing: smoke testing, integration testing, system testing
- supports Web Service REST APIs
  - HTTP methods: GET, PUT, POST, DELETE
  - parsing response format: JSON
- supports automating access to a protected API with ease
  - token generation
  - For example, postman has the concept of pre-request scripts that can be applied before a request is sent
- ability to write code before/after the API calls are made to set up/analyze data (for ex: postman allows javascript code)
- ability to group API requests or organize them into collections
- ability to send the same request multiple times asynchronously
  - this is useful for testing cases such as the one identified in QAE-9432
- Logging
  - ability to customize logging level (e.g. enable/disable certain types of logging)
    - For example, in some cases (e.g. GetMenu in OWS), the response can be quite large. This type of logging is useful while debugging a test but would otherwise clog up the logs. The ability to turn logging on/off for the response, for example, would be helpful.
  - types of data logged: URL invoked, request/response contents, expected and actual values being validated

Nice To Haves

- measuring and testing of API performance

- from Peter: security testing - Not full in-channel testing, but that security is applied to the api endpoints.

UI Test Tool Requirements

Must Haves

- supports the following testing types: functional testing - UI Testing and GUI Testing
- supports the following levels of testing: smoke testing, integration testing, system testing
  - *needed to compensate for when these levels of testing cannot be achieved via API testing*
- Test Scripting [POH: To me, this section feels to me like a reaction to our current UI testing tool rather than general requirements.]
  - Record & Playback tool with the ability to edit recorded scripts
    - has built in tool (e.g. Silk Test) or the ability to integrate (e.g. Katalon recorder is the Selenium IDE-compatible record & playback tool for browser automation testing)
  - Different Object Recognition methods
    - a variety of selection methods are useful because it will ensure that all objects can be recognized
      - examples include but are not limited to: automation id, name, class, XPath
  - Object Mapping support
    - option to map these objects in an object repository that should be easily updateable and managed
  - Support connecting to different data sources
    - Database
    - Data files (CSV, Excel, etc…) for BYOD approach
- Accessibility Tools
  - has built-in tools (e.g. Silk Test's Locator Spy) or the ability to integrate (e.g. Accessibility Insights) for object recognition

Nice To Haves

- Test Execution inherently occurs in a serial manner
  - This ensures that one command does not continue until the next has completed; thus avoiding the need for explicit timeouts that increase the time of a test, checking that an object is enabled before continuing, etc…
    - Example:
      - Open the URL
        - *and wait for the page load event to fire after all external resources have loaded*
      - Click on a button
        - *and wait for the element to reach an actionable state*
      - Validate the number of items displayed in a table
        - *and retry until the assertion passes or the command times out*
- Test Scripting
  - Recovery Handling
    - Recovery scenarios activate specific recovery operations when trigger events occur. The ability to create recovery scenarios and associate them with specific tests would be very useful.
      - Example: POS may have different warn values per order period as follows
        - Breakfast - allows multiple meal orders
        - Lunch - provides warning but allow multiple meal orders
        - Dinner - disallows multiple meal orders
        - A test that creates a "Now" meal order can be run at any point during a given day for any patient if there are recovery scenarios that are created separately, attached to the test and direct the test how to proceed in each of the above-mentioned scenarios.
- Self healing tests

- automatically identifies unexpected errors and recommends an alternative