A Story of Software QA Automation

From My Perspective

(John Rains, 2022)

Most of the people who will read this have never heard of Silicon Controlled Rectifiers[1] and TTL integrated circuits[2], but here I was amid these new revelations in technology. I was working my way through college as a lab technician in semiconductor development when a prior colleague called to recruit me into a company doing advanced machine tool automation[3]. Upon graduation in 1971, I relocated from southeast England to Rochester, Minnesota[4] as on-site senior electronics engineer for the project's new home. My degree in Electrical and Electronics Engineering and my love of Mathematics and Physics gave me a solid foundation in generalization, and the ability to transition to various industries and subsequent diverse careers. These in one way or another, involved computers and some type of automation. In 2013, I took a contract with a FinTech company to provide their Software Quality Assurance. This company's product is systems for managing online lending where minutes of downtime equate to $millions in lost revenue. I was very familiar with quality principles, having previously implemented ISO 9000 and Six Sigma but the practices of this company were diametrically opposite to any extant quality practices; but there I was, the only QA person in a multi-located company.

First Principles

As companies go, there are good ones, and there are dysfunctional ones. In my experience, most, to some degree, are the latter. The Association for Talent Development lists definitions for four stages of dysfunction[5] and includes a dysfunctional behaviors checklist; it is well worth reading. By these measures, the company I had joined was probably at what ATD describes as *hovering between Stages I and II*, and I was facing the challenge of implementing change in this environment.

I found myself facing many issues, most notably:

- Poor communications
- Inexperienced middle management
- Lack of in-depth product knowledge[6]
- A missing sense of *team*
- Siloed departments with goals not aligned across departments

---

[1] https://en.wikipedia.org/wiki/Silicon_controlled_rectifier
[2] https://en.wikipedia.org/wiki/Transistor%E2%80%93transistor_logic
[3] https://www.practicalmachinist.com/forum/threads/moilins-system24-and-associated-developments.292822/
[4] https://en.wikipedia.org/wiki/IBM_Rochester
[5] https://www.td.org/insights/overcoming-workplace-dysfunction-and-restoring-functionality
[6] The source code for the core product had been purchased from another company and subsequently adapted into this niche market, along with a plethora of never-to-be-used legacy code.

- Fear of accountability and a reluctance to put things in writing
- Upper management driving software releases
- An apparent belief that QA is unnecessary (developers should get it right – right?).

To succeed, I had to do what seemed at the time, impossible – effect change in a change resistant environment. By nature, the concepts of QA should penetrate all areas of an organization, but when considered unnecessary, QA cannot garner respect and will never be given the authority to effect policy. I could taste failure before I'd even begun.

I often turn to my bookshelf to pull out William Bridges' *Managing Transitions*; in fact, I have lost count of the number of copies I have given away over the years. Every manager should own this book which was first published in 1992 and is now in its 25th-anniversary special edition (2007). I credit Bridges with creating the 4P process[7] for transition management, although if you Google *4P*, you'll drown in a plethora of Marketing related results. Many people are familiar with the 4WH axiom[8] but the most significant difference between 4P and 4WH is *part to play* - acknowledging the people who are impacted by the change.

As an aside, I read the book *Who Moved My Cheese* when it was first released. I hope I am not the only one who hates this book. My disdain is primarily due to the proposition that change happens so deal with it, which is the complete antithesis of the fourth P. One takeaway is that *Change* happens, but *Transition* is a managed process, not an accident.

Any plan is better than no plan at all, and I thought I should start by listing QA's responsibilities:

- Audits product requirements
- Forms the best plan to test the product features
- Eyeballs the product through a web browser by clicking through it
- Facilitates entering defects/bugs and the bug triage process
- Guides and manages user acceptance testing
- Is responsible for reporting metrics including
    - number of bugs found / fixed / outstanding
    - bugs per thousand lines of code
    - bugs per estimate points[9]
    - automated test coverage

This was by-the-book BS and probably the wrong starting point: Even I didn't believe in it. It did, however, help me to consider a better first step, and identify what was wrong. To make sense, I need to give some idea of the system environments and tools used in developing the product.

---

[7] Purpose, Picture, Plan, Part to Play
[8] What, Why, When, Who, How
[9] https://www.atlassian.com/agile/project-management/estimation

The Environment

The company is a Microsoft software house. When I came on board, the servers were cloud-based on AWS but were later relocated to a commercial data center with Hyper-V servers. User interfaces are browser based, developed in classic ASP and ASP.Net, and customer-facing websites are responsive and required to function correctly on IE, Chrome, Safari, and Firefox, both desktop and mobile.

The core management system (CMS) provides the interface for customer service representatives (CSR) who are distributed across several locations with the CSR workstations running Windows 95, and IE6, soon updated to XP and IE8[10].

Software Developers are also distributed across several time zones, and developer team collaboration was facilitated through Google Hangouts[11].

There were four functional environments: Prod, Stage, Dev, and QA. Prod and Stage shared the production databases, while Dev and QA shared their own, common databases. Databases employed were both SQL Server and MySQL.

Development methodology is claimed to be Agile but is so Waterfall one gets wet just looking at it. Issues and new features are tracked in JIRA. Source control uses Git[12] and code deployments were manual (later TeamCity). CI/CD was a far-off in the future concept at that time.

Almost every aspect of the product involves PII[13].

Problems and Solutions

Back to what was wrong with the process. These were the top ones I identified.

- A persistent cycle of software release/fail/rollback
- Lack of good quality data for Dev and QA.
- PII leaks

The persistent cycle of release/fail/rollback sat squarely on the shoulders of the IT manager who often succumbed to pressures to get new features *out the door*. So, by default, regression testing was being done in Production.

---

[10] The restriction to IE was the extensive legacy use of VB Script in the CSR facing web pages. One of the heroes I refer to later took on the task of rewriting to Java Script, and the restriction was eliminated, allowing the CSR workstations update to Windows 10, using any browser.

[11] Superseded by Microsoft Teams.

[12] Later Bitbucket to leverage Atlassian consistency.

[13] Personal Identifiable Information (PII): Any representation of information that permits the identity of an individual to whom the information applies to be reasonably inferred by either direct or indirect means.

I make a differentiation between primitive data (the data entered by the customer) and extended data (the data generated by the system when consuming the primitive data). Software development involves a great deal of trial and error, and in a data-driven environment it is normal for this process to corrupt the data being used. These corruptions need to be cleaned up, but the extensive use of database triggers made this very difficult. So, periodically, the Dev environment needed to be refreshed with cleaner data – from Prod.

In Dev, a new feature might demand extended data that simply didn't yet exist, so some type of *synthesis* was required. This cannot, in my opinion, be the case in QA where extended data should always be system generated.

The QA environment MUST be equivalent to the Production environment. If the Production environment contains known defects (a reality), then the QA environment MUST include similar defects and tests should be included to ensure these defects are handled correctly.

When a Dev needed to test changes, they would search the Dev/QA environment for an account in the required state and, not finding one, would search Prod, and upon finding some, and with the assistance of DBAs, the Prod data would be copied to Dev. This was often an imprecise process, mainly due again to database triggers, and it caused the PII leak issue.

The issue of PII leaks was the first issue I addressed as it represented a quick win. I can never fully agree with the practice of data manipulation for QA testing, but the bleeding needed to stop. Real customers could receive auto-generated emails from the Dev environment because data copied from Prod to Dev was not sanitized, and lawsuits ensued. The official solution was to turn off auto-emails, but these are a critical part of the system which now could not be tested. When testing the auto-email was absolutely necessary, it would be turned on but often would not be turned back off – more lawsuits.

The two-step solution was simple. I created a domain on my personal web host (I know, I know – it should have been done within the corporate systems, but this is a dysfunctional environment and getting anyone to take ownership was difficult). Next, I communicated with DBA management (because it was fundamentally DBAs that were exposing PII) that they needed to implement a policy to sanitize the associated email domain. Although not perfect, it stemmed the issue until a better method could be implemented[14], and it could be measured. I created a database job to periodically examine the Dev/QA database email domains and report all non-conformities. Ironically, the DBA manager was the major culprit.

It's amazing how the Heisenberg uncertainty principle applies; if you try to measure something it changes. So, commutatively, if you want something to change, measure it. Any win is still a

[14] The method had not been changed as of my leaving four years later, but at least, the company now owns and hosts the domain.

win, and a win that saves the company money gets attention. Because of this solution, QA gained some credibility.

The matter of QA and Dev sharing databases should have been easy to resolve but it still took a fight. The existing databases had become so corrupted they were virtually unusable for either function. A perfect solution would be to create new data from scratch in new databases, but nobody had the knowledge or budget to do this. Instead, I negotiated for a distinct server environment for QA and started with a clone of Production that I sanitized as best I could. I felt it critical that the QA environment was owned by QA, and no one should mess with it's data.

Automated Continuous Regression Testing

I believe that Developers Unit test, and QA Regression tests. For regression testing, I needed a supply of new, distinct customers. There are now libraries for Python, PHP, Perl, and Ruby named Faker, and there is a dot Net cousin, Bogus. These weren't around at the time, so I created my own. I extracted massive amounts of data from existing production databases, disintegrated the data into discreet tables like Fname, Lname, Street, City, etc. then wrote SQL stored procedures to randomly recombine the disintegrated data into what I called Synthetics[15] [16]. These sprocs produced highly correlated data, i.e., the phone exchange, cell provider, ISP, IP, state, zip, and much more, were correlated. This correlation, or reference integrity[17], is essential to circumvent the basic functions of Risk assessment processing.

More recently, just for fun, I wrote a Windows Forms app that exercises Bogus; it is publicly available at https://github.com/JohnRains/Synthetic. While there is a suggestion that Bogus supports reference integrity, and creates correlated data, at the time of writing, it doesn't.

I created a desktop tool for broad use, to facilitate the consumption of Synthetics in the QA and Dev environments. It employs *click to copy* fields for manual entry into the customer facing web pages as well as XML data to POST, as third-party lenders require. This latter mechanism was most useful for populating large quantities of primitive data. Now, I could populate the QA and Dev environments with primitive data, on-demand. However, it did not address extended data.

QA was starting to gain traction, and the frequency of rollbacks relating to customer-facing changes had dropped to near zero, but it wasn't enough and didn't yet represent a valid shift-left[18] in the process. Getting the QA and Dev environments populated with high-quality extended data was crucial. Tempus fugit and we acquired a really good IT manager - a champion of true Agile methodology, unit testing, SOLID[19], and CI/CD[20]. This new manager offered, and I accepted a full-time position as QA Automation Engineer.

---

[15] A fantastic feature of MS-SQL is "Cross Apply" which I used liberally in the creation of synthetics
[16] Synthetics were frequently challenged and examined by Legal, and every time passed muster.
[17] A data quality characteristic, not to be confused with referential integrity
[18] https://smartbear.com/learn/automated-testing/shifting-left-in-testing
[19] https://en.wikipedia.org/wiki/SOLID
[20] Continuous Integration/Continuous Delivery and Deployment.

Once again, enter 4P with a *Purpose* of continuously, automatically, exercising the entire system, with the goals, for QA and Dev, of 1) perpetual regression testing, and 2) creation of clean extended data. In the *Picture*, the future state would be environments where high-quality data could be found in any desired state and defects in Dev would be discovered soon after the code was deployed. The plan for this involved the Selenium WebDriver framework[21]. I had been playing with various keystroke recording/playback tools, but Selenium WebDriver provides an interface to programmatically interact with a web browser. Once the plan was fully formed in my mind, I published a document named *How Automation: Part One*, extracts from which are given here.

General Concepts:

Context:
In this context, Automation refers to the process of programmatically emulating a customer, CSR, or process. Automation processes are defined in *tests* and the framework used to run the tests is NUnit3[22]. Interaction with browsers uses Selenium WebDriver.

Anatomy:
An Automation test is a set of code compiled into a dynamic link library (DLL) that can be executed from a Windows command using a command-line runner. The code is *decorated* with attributes: attributes separate sections of the code into subsections. An attribute in the code is not part of the actual executable code but is used to associate metadata with the code. Attributes declared in a DLL can be read[23] by test driver programs such as NUnit3 and this feature is used by the NUnit3 command line runner to identify the [TestFixture] and [Test] code sections of the DLL. The *test runner* interacts with the *test program*, guided by the schedule parameters, and the *test program* interacts with the browser using Selenium.

Test Fixture:
In NUnit, the [Test Fixture] is used to establish the environment within which the tests will run. It contains its own attributes, [OneTimeSetup], which executes code that sets up the test environment for the test, and [OneTimeTearDown] which executes code to clean up the environment once the test had finished. [Test]s are defined within the bounds of the [TestFixture]. A [TestFixture] can contain multiple [Test]s but I choose to not do so, consistent with SOLID concepts.

Test:
The [Test] section defines the steps to execute the test.

---

[21] Browsers don't inherently support Selenium, requiring a proxy, such as Mozilla (Firefox) gecko driver.
[22] There are many unit testing frameworks; originally Junit for Java. xUnit.Net is one I would focus on today.
[23] This is known as Reflection. Reflection enables external access to attributes within the code of a DLL.

Test Grouping

For this, tests are defined in three groups, and subsequently three DLLs:

GeneralAutomation

This group contains tests and processes that do not involve a browser. A good example is ACH Returns. Automated Clearing House (ACH) is a network for processing bank transactions. These transactions are batched for transmission to an ACH processor and sometimes a transaction within the batch will fail[24] and that transaction will be returned in a returns file along with a return reason code. In the test environment, the ACH batch file is created but not actually sent to a third party. An ACH returns file can be synthesized in the General Automation group by executing a sproc that randomly selects some of the previous day's transactions, attaches a randomly selected return code, flat or biased[25], and builds a Returns file that can then be processed in the CMS tests.

CustomerAutomation

This group contains tests for the product as presented to the customer in a web browser. It also contains tests for APIs that the application provides, such as the XML POST mentioned earlier.

CMSAutomation

This group contains tests for the core management system and the web pages presented to CSRs and administrative functions. It also contains tests for APIs that the application provides.

Parameters

Tests are invoked in a Windows command line with parameters that are subsequently extracted by the test code.

Invoking a test

Once configured, a test can be invoked from the Visual Studio test runner or a command-line runner. When invoked from Visual Studio, the parameters are provided by defaults declared in the Parameters class. When invoked from the command line, the Parameters class extracts the parameters from the command line.

Command Line

Example 1: This is one of the shortest invocations.

```
C:\NUnit\Nunit3-Console.exe -work:C:\Users\jrains\Automation\TestResults\ -
out:TestResult.xml
--where "name =~ 'simpleTest'" --p environment=qa "C:\QA_Libraries3\GeneralAutomation.dll"
```

[24] e.g. NSF (insufficient funds)

[25] Biased would represent the distribution density of return codes in a production environment

Breaking it down

C:\NUnit\Nunit3-Console.exe invokes the NUnit3 command line runner.

-work:C:\Users\jrains\Automation\TestResults\ -out:TestResult.xml establishes the working directory and output file name for the results of the test (Passed. Failed, Inconclusive, etc.).

--where "name =~ 'simpleTest'" declares the name of the test to be executed.

--p declares the parameters for the test; in this case, environment=qa is the target environment for the test.

"C:\QA_Libraries3\GeneralAutomation.dll" declares the full path to the DLL that contains the named test.

Example 2:

C:\Nunit\NUnit3-Console.exe -work:C:\Users\jrains\Automation\TestResults\ -out:TestResult.xml --where "name =~ 'moreComplexTest'" --p merchant=12345;environment=qa;browser=Firefox;browserSizeX=1200;browserSizeY=900;eSig=yes;pageScreenshots=no "C:\QA_Libraries3\CustomerAutomation.dll"

In this example for the *moreComplexTest* test, several parameters set the merchant parameter to 12345, with the test running in the QA environment, using Firefox browser at window dimension of 1200 x 900, requesting that the webpage is also eSigned and that no screenshots are taken.

Scheduling Tests

All of this is technically great, but I'm sure the reader can already see the difficulty of composing such complex command lines. Enter the concept of a test schedule and a schedule builder.

Regression Scheduler

The first iteration of test schedulers consumed pre-validated command lines from a database table, but this presented an issue not knowing ahead of time the maximum number of parameters. Adding tests increases the row dimension but adding parameters requires increasing the column dimension, which is substantially more difficult to implement.

The second iteration uses sparsely populated XML files, eliminating the database dimension issue.

Examples of the same two tests above are shown below in XML.

```
<Test>
    <jira>JiraTicketNumber</jira>
    <testName>simpleTest</testName>
    <environment>qa</environment>
    <testLib>C:\QA_Libraries3\GeneralAutomation.dll</testLib>
</Test>

<Test>
    <jira> JiraTicketNumber </jira>
    <testName> moreComplexTest </testName>
    <merchant>12345</merchant>
    <browser>Firefox</browser>
    <browserSizeX>1200</browserSizeX>
    <browserSizeY>900</browserSizeY>
    <environment>QA</environment>
    <pageScreenshots>No</pageScreenshots>
    <testLib>C:\QA_Libraries3\CustomerAutomation</testLib>
</Test>
```

Tests are grouped into a Test Schedule with the following format.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Tests>
    <Test>
        {test 1 declarations}
    </Test>
    <Test>
        {test2 declarations}
    </Test>
    <Test>
        {test3 declarations}
    </Test>
    …
</Tests>
```

When invoked, the Scheduler executes each test in turn, extracts the results (pass, fail, inconclusive), and reports the results. A schedule can be built that represents a full regression test of the system to verify, on-demand, that the system continues to function correctly. It is worth noting for boundary testing that test parameters intentionally outside of the boundaries are expected to return fail. In this case, an expected *fail* that returns a *fail* is reported as a *pass,* and an expected *fail* that returns a *pass* is reported as a *fail.*

There is no practical limit to the number of tests or test parameters that can be included in a test schedule.

Schedule Builder

A major component in creating this QA Automation is the ability for anyone to create a test schedule, without involving a software developer. Enter XML documentation comments[26].

C# allows the developer to create documentation for their code by writing special comment fields indicated by triple slashes. At build time, the compiler extracts the XML comments into a file named for the Visual Studio solution; in these cases, GeneralAutomation.xml, CustomerWebsiteAutomation.xml, CMSWebsiteAutomation.xml. These comments allow the test developer to communicate out of the test code using pseudo-language. The reader can readily see the pseudo-language structure in the example below.

Stakeholders such as the Business Analyst, QA Tester, Marketing, and more, should be involved in determining the dimensions of the test being developed so that the QA Automation Engineer can code the XML Comments appropriately.

```
[TestFixture]
 public class CreateApplicationTest : SimpleTestBase
 {
    /* Data for the Schedule Builder */
    /// <summary>
    /// <para>The first line of the summary gives a brief description of the test.</para>
    /// <para>The following lines of the summary convey information about the available parameters.</para>
    /// <para>Within the summary, a carriage return can be inserted using &#xD;</para>
    /// <para>More relevant information</para>
    /// </summary>
    /// <returns>Pass or Fail</returns>
    /// <value>environment select:qa,dev,uat required</value>
    /// <value>merchant select:12345, 67890 required</value>
    /// <value>browser select:Firefox,IE default:FireFox</value>
    /// <value>browserSizeX select:value default:1200</value>
    /// <value>browserSizeY select:value default:800</value>
    /// <value>annualIncome select:value default:</value>
    /// <value>bankrupt select:yes,no default:no</value>
    /// <value>bttcDismiss select:yes,no default:no</value>
    /// <value>bttcHour Select:value default:</value>
    /// <value>bttcMinute Select:00,15,30,45,no default:no</value>
    /// <value>bttcAmPm Select:Am,Pm,no default:no</value>
    /// <value>cellPhone select:value default:</value>
    /// <value>eSig select:yes,no default:yes</value>
    /// <value>Amt select:value default:</value>
    /// <value>nextDate select:value default:</value>
    /// <value>paperCheck select:yes,no default:no</value>
    /// <value>periodicity select:w,b,s,m,rand default:rand</value>
    /// <value>query select:cj,ln,no default:no</value>
```

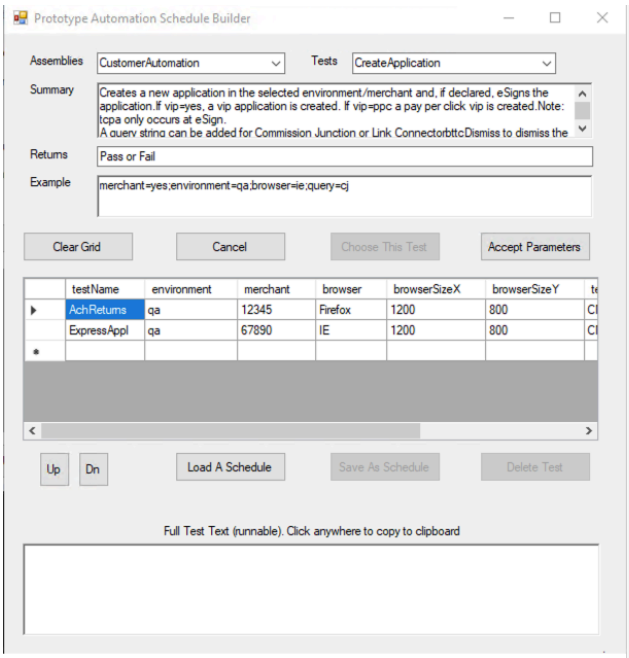
[26] https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/xmldoc/.

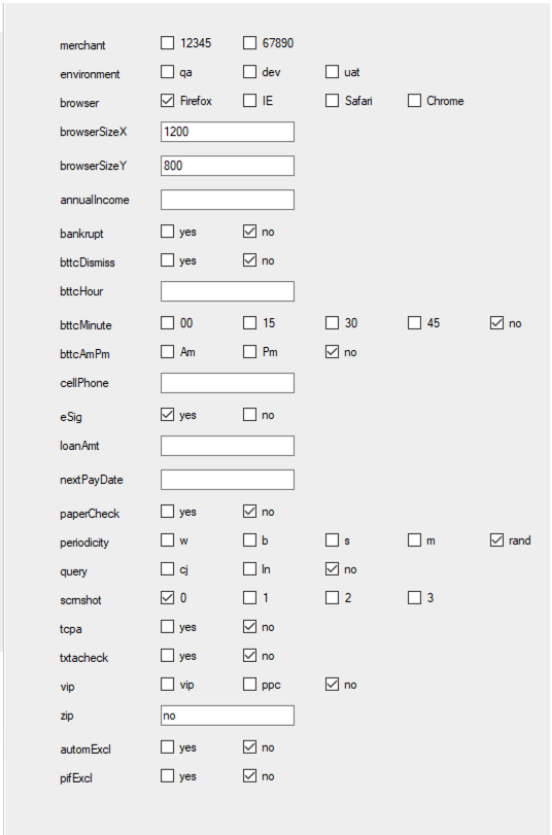
```
/// <value>scrnshot select:0,1,2,3 default:0</value>
/// <value>tcpa select:yes,no default:no</value>
/// <value>xpectedResult select:pass,fail,inconclusive default:pass</value>
/// <value>automExcl select:yes,no default:no</value>
/// <example>merchant=yes;environment=qa;browser=ie;query=cj</example>
```

To allow the Schedule Builder to be aware of the tests available, the test groups are manually added to the Assemblies.xml file.

```
<Assemblies>
  <Assembly>C:\QA_Libraries3\CustomerAutomation.xml</Assembly>
  <Assembly>C:\QA_Libraries3\CMSAutomation.xml</Assembly>
  <Assembly>C:\QA_Libraries3\GeneralAutomation.xml</Assembly>
</Assemblies>
```

In the Schedule Builder, when a test group is selected from the Assemblies drop-down, the tests available within that group are listed in the Tests drop-down. Choosing a test pops up a modal window where test parameters and declared options can be set according to the pseudo-language coded for that test. The test can then be added to the test schedule which is displayed in a DataGridView, and can be saved as a named schedule.





Tests can be re-ordered in the DataGridView and previously created schedules can be loaded and edited.

You might notice that the Schedule Builder is labelled Prototype. It was built as a proof of concept with the intent of creating a web based Builder available for anyone to use – a goal I did not achieve.

QAA Control

QAA Control is the controller used to run the regression test schedules. It introduces the concept of phases; a start of day phase (SOD), a Churn phase, and an end of day phase (EOD) mimicking the way the production system flows. Each phase has an associated schedule. The controller only allows SOD and EOD to run once each day, and there is no throttling of the tests, whereas Churn repeatedly runs its' schedule, executing each test within a time window, until the scheduled EOD start. These phases are defined in a control XML file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<control>
        <sodStart>7:30</sodStart>
        <churnStart>9:00</churnStart>
        <eodStart>16:30</eodStart>
        <endSchedule>17:00</endSchedule>
<!--    testWindow is the number of  5 seconds ticks that define the time window in which a test runs. A
        value of 30 is 2.5 minutes -->
        <testWindow>18</testWindow>
<!--  processHide controls whether the cmd window is shown or hidden -->
        <processHide>true</processHide>
</control>
```

QAA Control is launched each day by the Windows Task Scheduler and self terminates at the time declared in <endSchedule>.

Each environment, QA, Dev, and UAT has its own instance of QAA Control and the supporting XML files can be independently tailored. I introduced UAT so that user acceptance testing can be performed without impacting any of the other environments.

Wrapping Up

QA Automation was satisfactorily running in the QA, DEV, and UAT environments, creating data and revealing defects to the developers early in the process – a truly shift left benefit. People would look through the windows of my little mission control center and watch, fascinated by QA Automation in operation. Software QA, and the level of quality of the company's software deliveries, had come a long way since those first days. I feel proud of what I achieved and thoroughly enjoyed doing it. Working with many diverse heroes in the industry was the greatest retirement gift I could ever have.

Final Thoughts

Henry Ford is attributed with the saying *Quality means doing it right when no one is looking.*

Throughout this document, I have referred to the quality of the data and the ease of corrupting the data. Another goal I had set for myself was to achieve what I had earlier said, a new, clean database set. I knew this would be a huge effort, but I also knew it would be worth doing. I did not achieve this goal, but I urge others to maintain and respect the data.

I must give a shout-out to the heroes. A well-functioning company has leaders and team players for its success. A dysfunctional company relies on heroes; take away the heroes and the slightly dysfunctional company will sink into stages III and IV. Generally, heroes are recognized by their peers, not themselves. It is worth aspiring to be a hero.

Communication, communication, communication. One can always talk too much but never communicate too much, and secrets are poison to a well-functioning company. The best company I ever worked for (as Manager of Corporate Applications Development) had a genuine open-door policy. It established a practice that managers should set aside one hour each month to meet 1:1 with their people from second level down, separate from the longer and more frequent first level down meeting. That hour was the employee's time to talk about anything they wanted, work-related or not, and ask any questions even ones that elicit, *I don't know*. And before you think this can only be done in a small company, this company had over 6,500 employees on the corporate campus alone ten times that in toto.

Anyone interested in knowing more about my thoughts on Software QA Automation can contact me by email: john@starfish.vi.

JSR

Who am I

- STC Ltd, UK (division of ITT)
  Semiconductor Development (college student)
- Molins UK, London, England
  Electronics Engineer, System 24
- Woolwich Polytechnic University EE
- Molins UK, Rochester, Minnesota
  Lead Systems Engineer
- Molins Richmond, Richmond, Virginia
  Manager, Electrical Engineering
- AMF Bowling Automatic Scoring Division, Mechanicsville, Virginia
  Director of Engineering
- nView Corporation, Newport News, Virginia
  Project Manager
- Circuit City Stores, Richmond Virginia
  Manager, Corporate Applications Development
- ComTekX (Computer Technology Experts)
  Owner
- Hope House Ministries
  Pastor
- Starfish.VI
  Owner, software developer, photographer
- Cane Bay Partners VI, LLLP
  Software QA Automation Engineer
- Musician
  The 4 J's, The Waterproof Sparrows, The Dusty Blues Band, independent song writer/performer
- Instruments
  Too many guitars to mention; Home recording studio using Alesis Q49/Q25 MIDI controllers, Focusrite Audio Interface, Logic Pro X
- Photographer and heritage historian for St Croix, VI (https://Heritage.vi)