Six Principles of Design

The term Principles of Design is often used in Art and Photography. Similarly, there are also design principles that can be found in any successful automation project. Six Principles of Design are based on a standard of quality characteristics, of which these six presents the core of Qentinel Paceword automation Principles of Design beautifully.

1. Maintainability
2. Usability
3. Reusability
4. Efficiency
5. Security
6. Reliability

Maintainability

Maintenance of the test cases is often ranked as the number one issue in test automation. It is something that any test automation project must care about. It is not that uncommon to see test automation engineers spending all their time on test case maintenance instead of designing new test cases. Once you let the time needed for maintenance grow out of proportion, there's no way back, except redoing everything. Continuous feature updates in the software under test, of course, are the root cause for the maintenance need, but a fragile testware makes the maintenance effort skyrocket. Maintenance challenges exist in test execution tools as well as in test interface libraries and design techniques used which all are separate entities.

```
Qentinel - Contact Us
    Appstate            Qentinel
    ClickText           About us
    ClickText           Contact
    TypeText            First name          Jane
    TypeText            Last name           Doe
    TypeText            Email*              jane.doe@yahoo.com
    TypeText            Message             I need help in test automation
    ClickText           Send
    VerifyText          Thank you! We'll be in contact with you soon.
```

Qentinel Pace defines only a couple of guidelines for designing test cases and keywords, or Pacewords as they are called:

- Keywords are short, 8-12 characters are usually enough for a descriptive name and every extra character is considered a waste
- Each keyword carries out one atomic operation, such as filling an input field, rather than being an application-specific operation or routine
- There is a specific keyword, Appstate, for initializing the system under test in the desired state
- Every keyword has built-in verification

Such atomic keywords make a test automation solution easier to maintain, because:

- Everyone can understand the test case flow, which is important in reviews or PRs
- The same test case can be executed on different platforms if the test case flow is the same
- Keywords are not project-specific, but the same across different projects
- Test cases look nice and structured which make it a joy to use and maintain

There is a downside, too: test cases may become a bit long but this is more than compensated by the linear, easy-to-follow structure.

When comparing PaceWords to other scripting techniques, it is like Python in programming; simplicity in syntax, reliability, and versatility. In case there are doubts regarding Paceword approach maintenance, please, read the case study "How to automate 18 109 test cases and keep maintenance effort under 15%".

Usability

Usability is a key quality characteristic of any modern application. Usability starts from learnability, i.e. how long it takes to learn automating a test case and keywords to be used in this context. The simpler the technique used, the faster it is to learn not only by one test automation engineer but a team of automation engineers. Understandability is the core of the use of the Paceword technique, everybody who uses the application under test can also understand the test automation script even at a step-by-step level. This is a huge difference compared to other test automation techniques, such as Behavior Driven Development (BDD). To their merit, BDD test cases are very easy to understand at a high level, but very different when inspecting at step level.

Let's have another look at the same picture as before, but this time asking a question: Can I understand every step in the test case below?

```
Qentinel - Contact Us
    Appstate            Qentinel
    ClickText           About us
    ClickText           Contact
    TypeText            First name          Jane
    TypeText            Last name           Doe
    TypeText            Email*              jane.doe@yahoo.com
    TypeText            Message             I need help in test automation
    ClickText           Send
    VerifyText          Thank you! We'll be in contact with you soon.
```

It is easy to follow the flow of Pacewords because the technique mimics the actions and flow of thought of a human being. Someone may wonder if that can be so simple. Yes, it can, because Pacewords take care of the complex logic of xpaths, time-outs, and other peculiarities.

Reusability

Most test automation solutions are built for one project or one platform only. PaceWord technique, on the other hand, is reusable across different projects and platforms. Different target platforms, such as web or mobile, need of course a specific implementation of the test interface library, but the Pacewords are the same across different platforms. ClickText, TypeText, etc., mimic what a human would do when using a graphical user interface and testing any system under test. The beauty of the concept is that the way a tester interacts with the automation solution is the same regardless of how the test interface library interacts with the system under test.

Qentinel Pace supports various platforms, and anyone can extend it for more:

- Web application testing and Robotic Process Automation (RPA) – QWeb
  - Different browsers are supported
- Mobile application testing and RPA - QMobile
  - iOS and Android platforms are supported
  - Android Automotive is a special variant of the Android mobile
- Computer vision and optical character recognition - QVision
  - Native application testing and RPA
- Physical robots - iRobo
  - Control a real robot just the same way you would control a software robot

Test automation engineers love to implement rich project-specific keyword libraries. It may sound like a good idea but, in reality, it is likely to destroy your productivity. It is a bit like having to invent a new language every time you join a new team. A rule of thumb is that 20% of the keywords should cover 80% of all the steps used in a project, reference to Pareto principle (80-20 rule).

Efficiency

Efficiency is a term that is used on many occasions. Effectiveness and efficiency together mean "doing the right things right", i.e. Strategy and Operations. The efficiency of test automation design is linked to performance and resource utilization which is important in both keyword design and test case design. Some examples of performance, or timing, of:

- How long it takes to design a test case
- How long it takes to fix an issue in a test case (maintenance)
- How long it takes to run a test suite
- How long it takes to execute a keyword

The keyword technique in use has a great impact on test case design and maintenance time. Keyword internal design on the other hand has the biggest impact on the keyword execution time. While internal efficiency of the keywords matters its impact is much less significant than the impact on the efficiency of test case design: internal efficiency consumes computing resources while external efficiency consumes human resources.

Internal efficiency should not be ignored, though. Minor differences may grow big with repetition when the keyword is executed one million times minor things accumulate:

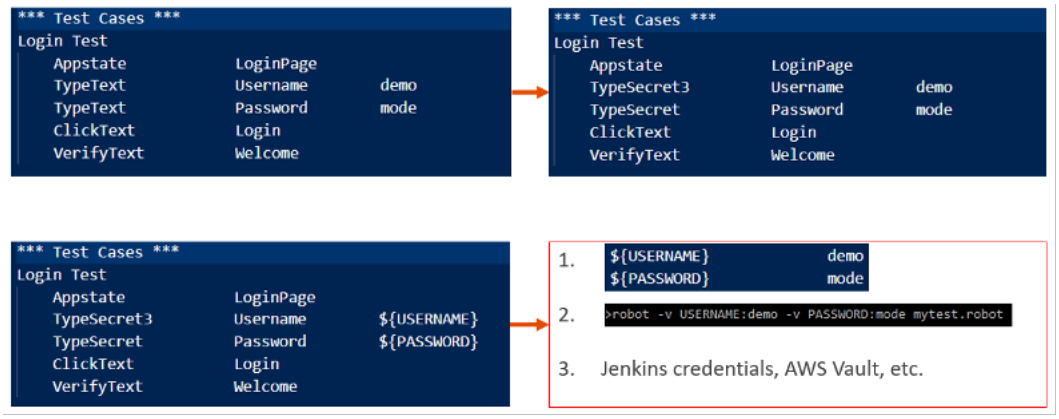
- 10ms sleep in a keyword executed 1 million times is more than 2 hours 45 minutes wasted time
- 2MB screenshot image is saved 1 million times is more than 2TB disk drive space

Million is actually a small number in test automation. For example, in the "How to automate 18 109 test cases and keep maintenance effort under 15%" study the most common keywords are executed over 1 million times every week.
Test Automation Efficiency is Art itself and about optimizing keyword design in conjunction with the right test case design by minimizing the time and resources used.

Security

Security is the key quality characteristic of any Software as a Service product. Unfortunately, one of the easiest ways to penetrate into a system is to study its test scripts. Security is even more important in the RPA field where real data and real environments are involved. A user must be able to trust that data is securely stored and can be securely used.

Let's go through a simple test case with five different types of security options when typing username and password:



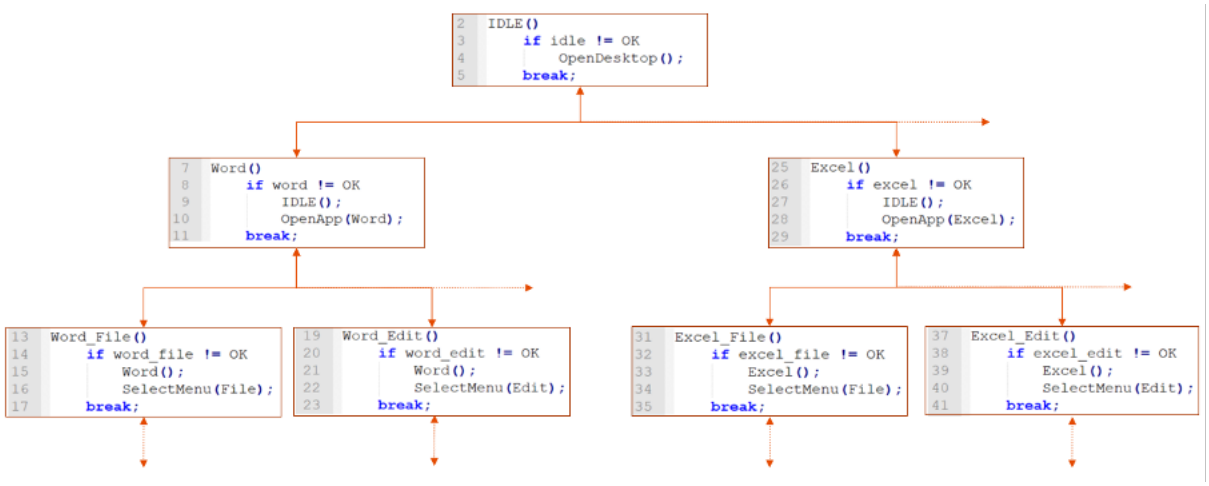
1. Top-Left: Username and password are visible on both test automation script and test results files. No data security at all.
2. Top-Right: Username and password are visible in the test automation script, but not in test results files. TypeSecret Paceword does not log parameters to the test results files, and TypeSecret3 only three first characters, i.e. in this case “dem********”
3. Bottom-Left: When using TypeSecret Pacewords and Robot Framework variables, there are three options:
    - Variables are visible if they are defined in test scripts or resource files
    - A user can use Robot FW command-line variable option when knowing the data
    - When using Qentinel Pace, secure data is stored in AWS Vault. Similarly, when using the Jenkins CI tool, the secure data can be saved under credentials plug-in

Every project is different, and the level of security depends on the requirements. Test automation projects may not have that strict security requirement, but RPA projects are running in production, and security requirements must be fulfilled. There are features in Qentinel Pace that take security-related SaaS requirements into account, more information is available on qentinel.com.

Reliability

Reliability is built into every Qentinel Paceword so that they work every time with minimal effort. This is about being a mature test automation solution. Debugging step by step a test case that is working randomly, can be a time-consuming nightmare – but a very common one. It is often caused by different delays in the application under test. The more sleeps/ waits there are in the test case scripts, the more immature the test automation solution is. The use of sleeps/waits does not belong to Paceword technique at all as they are built-in proving a fault-tolerant solution.

Recoverability after a failure can be managed using different techniques. One common technique is to reboot the system, or simply close and open the application after every test case. This, of course, has a big impact on the test suite execution time. There is a Paceword that manages this complex issue smartly. It is called “Appstate” which, by default, is the first step in every test case. See the pseudo-code below and inspect how “Appstate Word File” would execute given that the workstation can be in any running state.



The core of Appstate is “don’t do anything if the system state is right”, which speeds up the execution time significantly when this is taken into account in the test suite design. Appstate does not have a significant role in web application test automation, other than cleaner test automation scripts, but it has a more important role in mobile app testing and particularly in computer vision-based testing where there are no shortcuts but very complex navigations and exception handling. In short, Appstate is a navigation system that executes the pre-condition steps of a test case.

Paceword technique design guidelines

1. Use clear and descriptive test case names, e.g. Login – wrong password (Robot Framework)
2. You may use [Documentation] to explain the purpose of the test case (Robot Framework)

3. You may Use [Tags] for running selected test case groups (Robot Framework)
4. Appstate must be the first step in a test case; navigation and pre-condition of test case
5. Design test cases as readable as possible, so that anyone can review them
6. Do not use Sleep keyword instead use VerifyText with a timeout, if needed
7. Avoid the use of project-specific keywords, no more than 100 keywords total per project
8. If project-specific keywords needed, preferably use Python instead of Robot FW scripting
9. If a long test case, divide it into 10-line sections, or logical sections, and use the Log keyword to describe the next section so that a reviewer can follow the test case or task