Erika Lepistö

Best Practices in Test Automation

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

28 October 2022

Abstract

Author:	Erika Lepistö
Title:	Best Practices in Test Automation
Number of Pages:	44 pages
Date:	28 October 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Health Technology
Supervisors:	Mika Lindh, Service Owner for Test Automation Päivi Haho, Principal Lecturer

Test Automation is a subsection of testing a part of software development itself. In test automation, the testing is done by programming an automated script that does the testing. For this reason, the field is a lot closer to traditional software development than testing. However, the people who get into test automation have usually not gone through traditional studies to become programmers. This has led to the quality of the code to suffer. To fix this a Best Practices documentation was created. The document goes through the main methods on a paradigm level, with real-life examples. The methods will in the long run result in better quality output from test automation developers. The document goes through the most important subsections: Project management, Test Architecture, Library Development and Version Control.

The purpose of the study was to approach the guidelines in a softer way, without creating a checklist of mandatory actions. The subsections were selected in accordance with what the team found to be the most important for the project health and most in need of guidelines. The intention was not to create a technical documentation that would go through the development process step-by-step. Instead the guidelines are supposed to guide people in their thought process on how they could develop themselves and improve their work.

Keywords: Quality assurance, test automation, best practices, rpa, robot framework, selenium, test architecture, version control, documentation, library development, test development, workflow

Tiivistelmä

Tekijä:	Erika Lepistö
Otsikko:	Parhaat käytänteet testiautomaatiossa
Sivumäärä:	44 sivua
Aika:	28.10.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Terveysteknologia
Ohjaajat:	Palvelu omistaja testiautomaatio Mika Lindh Yliopettaja Päivi Haho

Testiautomaatio on ohjelmistokehityksessä testauksen alamuoto, jossa testaus toteutetaan ohjelmoiduilla automaattisilla testeillä. Tästä syystä testiautomaatio on alana lähempänä ohjelmistokehitystä kuin perinteistä testausta. Alalle päätyy kuitenkin paljon työntekijöitä ilman varsinaista ohjelmistokehityksen koulutusta, mikä on johtanut tuotetun koodin tason laskemiseen. Tähän ongelmaan ratkaisuksi kehitettiin Parhaiden käytänteiden dokumentti. Tämä dokumentti käy ajatustasolla ja oikean maailman esimerkein läpi työtavat, jotka johtavat laadukkaampaan kehitykseen ja pitkällä tähtäimellä alan laadun parantumiseen. Dokumentti käy läpi testiautomaation tärkeimmät osa-alueet: projektin hallinta, testiarkkitehtuuri, kirjastokehitys sekä versionhallinta.
Työn tarkoituksena oli lähestyä ohjeiden tekemistä pehmeämmällä tavalla, eikä luoda listaa asioista, jotka on pakko tehdä projektista riippumatta. Osa-alueiden sisällöksi valittiin työn onnistumisen kannalta tärkeimmät osa-alueet, joissa työryhmä koki olevan parantamisen varaa ja jotka koettiin mahdolliseksi ratkaista selkeällä ohjeistuksella. Työn tarkoituksena ei ollut luoda teknistä ohjekirjaa, jossa käytäisiin vaihe vaiheelta läpi tehtävät asiat. Sen sijaan työn on tarkoitus ohjata miettimään kunkin omaa kehitystyötä ja kuinka kehitystyön nostamia asioita voitaisiin hyödyntää tulevaisuudessa.

Avainsanat: Testiautomaatio, parhaat käytänteet, projektin hallinta, dokumentaatio, ohjelmistokehitys, laadunvarmistus, versionhallinta, kirjastokehitys, työnkulku

Contents

List of Abbreviations

BP: Best Practices. A procedure that has been shown by research and experience to produce optimal results and that is established or proposed as a standard suitable for widespread adoption [1]

TA: Test Automation. Type of software testing where the tests are automated.

QA: Quality Assurance. A program for the systematic monitoring and evaluation of the various aspects of a project, service, or facility to ensure that standards of quality are being met [2]

RF: Robot Framework. A commonly used framework for Test Automation.

1 Introduction

Test Automation (TA) has gotten away with subpar code and development cycles for too long. Old tests are not properly fixed and new things just get added on top. It is almost like programming students the night before a deadline, but this is in the corporate world and people seem to be fine with it. Until it breaks of course.

Inherently test automation is just another type of software development and should thus follow the guidelines of common sense and universally agreed upon good practices. It is also a type of testing, a field of work that traditionally requires a different skill set in comparison to that of software developers. This has created a sort of wild west within the TA community as most people have not been privy to the common-sense guidelines taught to all developers. This has led to a decline in code quality and code maintainability in the TA sector. Another leading problem is that the code will never be seen by anyone outside the TA development team if even them. Sometimes code is only seen by its developer.

To combat low-quality code and overtly complicated solutions a set of rules was commissioned from the author. This thesis contains the basics of these rules and guidelines in the form of Best Practises (BP). The rules were created with an innovation team comprised of people of different seniority levels from different product owner teams all with their speciality. With interviews and documentation received from this team the thesis was created to function as a starting point toward upgrading test automation quality.

The aim for the report is to work as a tool of quality assurance for the test automation team, just as the test automation functions as a tool of quality assurance for the product owner in software development. The thesis goes over the most basic areas of test automation projects and gives reasons why they should be considered before a project has begun.

The goal of the study was to provide understanding on how TA development could be guided in a better direction and what should be included for maximal coverage. This goal was achieved with a fully realized BP document that goes over the most basic areas of interest in a fully functional TA development. The final result was done through the constructive research method, were working within a team and in the field was combined with academic research.

This thesis has been split into three core components, the Academic research in Chapter 2, the overview of the Method in Chapter 3, and finally the result, the created guidelines in Chapter 4.

2 Software Testing – Literature Overview

Software testing will never make a program completely bug free.

Software testing is a part of the software development lifecycle. Its core purpose is to validate that the program fulfils the Acceptance criteria and that it functions as intended. It can ensure functionality, but by itself, it will not make the program better. Testing will only uncover its flaws.

Software testing consists of different types of testing, that all require different specializations. The following list is from IBM's page on software testing as it sums up different types of testing perfectly. [3]

- Acceptance testing: Verifying whether the whole system works as intended.
- Integration testing: Ensuring that software components or functions operate together.
- Unit testing: Validating that each software unit performs as expected. A unit is the smallest testable component of an application.
- Functional testing: Checking functions by emulating business scenarios, based on functional requirements. Black-box testing is a common way to verify functions.
- Performance testing: Testing how the software performs under different workloads. Load testing, for example, is used to evaluate performance under real-life load conditions.

- Regression testing: Checking whether new features break or degrade functionality. Sanity testing can be used to verify menus, functions, and commands at the surface level when there is no time for a full regression test.
- Stress testing: Testing how much strain the system can take before it fails. Considered to be a type of non-functional testing.
- Usability testing: Validating how well a customer can use a system or web application to complete a task.

All these types of testing emphasize different aspects of development and program quality. They all have their place and while some are more important in specific projects, all should be at least considered when creating a Test plan. In general, all but Unit testing is done by the Quality Assurance (QA) team, when they validate critical functions and acceptance criteria of the program.

2.1 Test Automation

Already in the 1990s, it was expected that test automation will take over a large part of the regular manual testing, the regression testing. [4] Regression testing is the repeated act of QA where an already functional part of the software is retested after a new patch is applied. This is in general extremely boring work, and large software projects can have hundreds of regression tests that need to be rerun after all updates. This used to be done manually by human testers but has quickly been shifted over to Test Automation (TA).

Automated testing is an integral part of today's software QA economy as multiple automated systems can be simultaneously run and regression testing results are quickly available. This cuts down the downtime between updating and bug fixing the software and saves time and resources for the software development team.

In general, automated testing can save a lot of resources during the lifecycle of software, as it can be used as a tool for example security testing, stress testing, and end-to-end testing. [5] What TA cannot do, is exploratory testing. This means that TA will never completely replace manual testing, it will just take over

the most boring and repeating part of the work, leaving testing personnel with more rewarding work.

TA is often done through Automation focused programs and code libraries. In this document, the Focus is on the Robot Framework.

2.1.1 Robot Framework

Robot Framework is a generic open-source automation framework designed for Test Automation (TA) and Robotics Progress Automation (RPA). It is widely used in Finland by most of the large IT companies including Nokia, Vero, and Kone, as well as outside Finland by Cisco and US Naval Research Laboratory. [6] The Framework itself is simple with simple syntax and it is completely based on Python. This means that anyone can create more functionalities simply through python programming and it will seamlessly function with the basic Robot Framework tools.

Alongside these Built-in Tools, there are also libraries for all needs. [7] These too are often Open source. This document uses Qweb-library to showcase TA programming, as it is meant to be easily readable and more easily understandable to people who might not have seen TA programming previously. [8]

2.1.2 Test Development

In TA tests are developed with a specific purpose in mind. The development process begins with a test script. The script can be given to the developers by the product owner, or it can be created in exploratory testing by manual testers. No matter where the script comes from it will be assigned to a developer that

manually checks its functionality. If problems are found the script goes back for revisioning. When everything is fine the actual development can begin.

The TA will follow rules from Project Management and Test Architecture when deciding on the approach, usually, these include the platform and libraries that are going to be used. The tests are automated step by step, and the result is an automated program that goes through the test script as it has been programmed to do. Before implementation of the master code, someone else should go over the code, just to see if any obvious mistakes had been missed, or if the solution is unnecessarily complicated. If the test passes it will be merged into the master code.

After a test has been implemented, usually to a cloud storage, it will be run within its group of other tests. These runs often take place in the early hours of the morning to minimize load times and interference caused to actual users, or manual testers.

Once the test is up and running, it will be maintained. Sometimes things in the User Interface change, e.g. an extra confirmation is removed, and the TA script needs to be edited for it to function. Inherently TA scripts are failure prone if any changes are made, but if they have been properly developed the TA can be changed to adapt. If the development has been keyword driven, more on that in Chapters 2.3 and 4.3.2, the changes are done to the keyword and then implemented. In these cases, the change only must be done once, and it will fix all tests that use the keyword. In large cases, this lessens upkeep time drastically.

2.2 Project Management

Project management is the framework that guides project processes, methods, knowledge, and experience towards agreed-upon acceptance criteria. It is what all decisions in development are based upon and where all prioritization and

limits are created. The primary responsibility of project management is to prioritize the project in a way it fits into the budget and timescale. This is done through risk management, communication, and proper work allocation. The main difference between just management and project management is that it is tied to the timescale of the project. When the project is done, so is the project management. [9]

In Test automation contexts project management is two-sided. It is the product owner side that works out all details in the software development lifecycle, and then there is the testing project management that is only managing the testing portion of the software development lifecycle. This document focuses on the second type of management. While the Best Practices outlined in this document can be useful for other specializations and project management overall it is not meant for that and thus some revisions are required.

Project management in general, when done properly, can take as much as 20% of the project budget. But without it, the team will be left with their own decisions over prioritization and budgeting, time and money. Not planning out proper management can and often has led to poorer outcomes, and delayed project completion. Especially in critical environments proper management in the testing sector can be the difference between a fully functional project and software that fails without collapsing, silently. In a fast-paced economy, this can be a deal breaker, where at best it causes costly revisions later and at worst it leads to fines and contract losses. [10]

2.3 Test Architecture

Test architecture is a way of organizing test automation code. It has two levels of function, a broad strategic level, and a technical detailed level. On the strategic level test architecture considers the most cost-effective holistic way to test an application to appease the acceptance criteria, when to automate, and

how much end-to-end testing is needed. On a more detailed level, it considers the precise areas that need to be tested more thoroughly and how to get maximal coverage with minimal effort. In TA this usually includes the amount of library development a project requires. [11]

In this case, the focus is on how architecture affects TA and its development. TA is a critical tool in an Architects tool kit to keep testing coverage high while keeping the cost manageable. While automation requires oversight it can easily handle long processes and time and be used more efficiently. For this to be possible the test architecture must decide how much effort is placed on TA. A test library can be created on a case-by-case basis where tests run by themselves and do not interact with a keyword library. However, if TA is to play a larger role a more keyword-centric development style can be adapted. The scope of which lies solely in architecture.

Keyword-oriented programming for test automation is the superior way to handle large projects with hundreds of tests, as keywords are easier to maintain when changes need to take place. It is also a lot more work. Keyword-driven development requires a lot of generic thinking and changing common keywords to include more and more overlapping cases.

In small projects, the creation of a keyword library can be seen as too much work. This is also the case if tests do not have much overlap. Generally, navigation and basic functions: Login, Log out, cookies, etc should be placed under an Appstate keyword. Appstate should include all the pre-conditions and it is used to set the software to a known state. This is to make sure that there are no surprises in the software landing, as for example a Cookie-pop up that could break the tests. The below example (Example Code 1) is from Qweb-library documentation, and it explains how an Appstate should be built. [12]

```
Appstate       Login
#with arguments:
Appstate       Login    fenix   rising123

#Example block:
Login
    [Arguments]     ${USER}=username     ${PASS}=password
    Goto        https://www.google.com
    TypeText    Username     ${USER}
    TypeText    Password     ${PASS}
    ClickText   Login
    VerifyText  Welcome, ${USER}
```

Example Code 1. Example of an Appstate, that automatically Logs in an user.

Inherent things that need to be done in every test, should be made into a keyword to lessen overlapping work and to make the developed TA more maintainable. A TA solution with only a few tests might not require this, but if the project were to ever grow, even a small library can make starting easier.

2.4 Documentation

Documentation is vital for the health of the test automation project, or any software development project in general. The documentation is a source of information about the project to oneself and to that outside of the test automation project domain area. It can be used to study and to get to know the project domain area, what is included in the project and what is not. For product developers, it can act as an archive of the test automation project.

Documentation is an integral part of product development and while talking about it also the software development lifecycle was discussed. Lifecycle documentation and project documentation are similar, but they take a different approach to the subject and should thusly not be confused with each other. [13]

Important aspects of documentation are the following:

- Quality over Quantity
- Agree with your shareholder’s practices regarding the documentation work
- Use design patterns even with documentation

- Availability to all shareholders in a secure way and manner
- Documentation is kept up to date when changes are applied
- You can learn from documentation
- Treat your technical documentation as you would treat your source code
- Make sure that your documentation is for the right subject group in an understandable way
- When possible, provide templates for documents that are used repeatedly
- In the modern era of computer science, the documentation should contain interactive methods such as
    - Links to other related materials
    - Multimedia files when needed
    - Pictures as well as diagrams

[13, 14, 15, 16]

Documentation is key when bringing in a new team member and thus poor documentation is usually caught when new members are struggling to get on track. Writing and updating documentation requires time and effort but doing a little every time there is a change, will minimize the struggle in getting it done. There will always be times when documentation and reality are not in sync. This downtime can be reduced by everyone reviewing the documentation periodically.

Documentation can also be used to refer to Test documentation. [16] This usually means to be a list of tests that have been done, and their results. In TA the documentation serves both purposes, but heavily leans towards the more technical type of documentation. The resulting side of test documentation comes inherently from the TA solution. When a test has been run, a file with the results is generated.

2.5 Version Control

Version control is often done through specific programs. Gone are the days of sharing a file and manually appending into it the changes. Now it is all automated.

Most commonly software such as GitHub is used to host versions of the code in the cloud. Having the code be in the cloud provides the possibility of multiple people working on the same file simultaneously. To do this efficiently a branch-like system is used. Everyone has their branch and possible changes are not done to the Master branch at all before they have been thoroughly tested and approved. [17]

3 Method

The study was accomplished using the constructive research method. The method is meant for innovative projects that are trying to solve a real-life problem. The guidelines created function as the core of the research, and they contain artifacts created by the author and the innovation team. These artifacts contain the figures presented throughout this document, as well as the documents created by the innovation team. The beef of the constructive method is that all artifacts are created and developed a specific purpose, instead of being found.

The key points are as follows:

- the research is focused on a real-life problem, that is seen as worthy to solve,
- a construct is created to solve the real-life problem
- the research is done in a team setting with the researcher and the practical advocates
- the research is tied to theoretical knowledge

- focuses on reflecting empirical knowledge to the theoretical knowledge

[18, 19]

In the thesis the construct is the Best Practices documentation that was created to function as an internal set of guidelines. It was created with the help of innovation team consisting of 8 experts, that had been selected from different projects, each with their own speciality and insight. The author functioned both as the researcher and from the practical side as she was responsible for the literature research as well as some of the BP research.

The work itself was began with a meeting of the entire team where the main areas that would be covered were decided upon and the work was delegated between the team members. Some were responsible for multiple areas whereas some only had one area of expertise. The team then created documentation over their own speciality.

From these documents the main document was created. Here the practical knowledge was reflected against the theoretical knowledge and the most technical details were left out, as they would now function for wider range of projects. This simplified version became the Draft 1 that was then sent back to the innovation team for review.

Review notes were then used to create Draft 2. That included the final list of contents. That too was then reviewed. With the final review, and the changes from that the final version of the Best Practices (BP) was created. Alongside this an introductory presentation was held for the company and then the result was published internally.

What is now included in this thesis is a further simplification of technical knowledge of the BP documentation with further references to the theoretical knowledge.

4 Best Practices

Best Practices (BP) are a way of producing good outcomes if followed. They can either be a standard or task specific set of guidelines. They can be created by the government or can be internally set for an organization. [20] In Europe the EU sets standards for example product quality and security. Nowhere is this more apparent than in the new directive EU 2017/745 that tightens the safety regulations of medical devices. [21] These directives are one type of BP, just on a stricter level. This thesis functions on the lower level and is meant to be an internal set of guidelines.

These guidelines are supposed to be a soft guide to areas that require attention and proper ways to give them that. It is not supposed to be a checklist the team goes through at the start of a new project. Not all aspects of these guidelines are useful in every project and in some cases, they can be harmful. The development team must make its conclusions over which portions they can achieve without stifling innovation and work motivation.

Best Practices are a way of prioritizing and optimizing group work. They are a set of rules and guidelines the group opts into. They can also be used as a way to make development work visible, where it often would not be. Improved visibility of methods and ways of work can be vital for a difficult project where the progress is often not linear or visible. For Test automation purposes this is exactly what is often needed.

The following chapters go through the created Best Practises and reasoning for them. The Thesis covers Workflows, Project Management, Test Architecture, Library Development and Version Control. These aspects were chosen to be covered, as they are the easiest to give general guidance to. They were also recognized by the innovation team to be the parts where projects often fail.

4.1 Workflows

Workflows are a way of visually establishing the way work should be done within the included project. In TA this can give a visual aid to explain steps of test automation, or other steps of QA to the product owner and to explain the requirements and responsibilities of the software development team. It should be seen as an asset for understanding [22].

Commonly there are three types of workflows: Process, Case, and Project. In TA most workflows are Project workflows as they are done for a specific case and often have been created for a specific project in mind. Most variables and steps are known in advance, but how the steps are completed is more flexible. [23]

Workflows can be done with any level of detail, below there are two examples of different level workflows. The first one tackles the overview of the test automation process as seen by the product owner. Whereas the second workflow is more intended for the test automation team.

The main workflow is a good place to start with the project owner. It explains where and how the test automation process can be controlled and when the QA team requires outside participation. In Figure 1, the points to influence the QA have been colour coded as blue. This level of workflow can be used to explain how the team handles requirements given to them and in the case of delays, where those delays are happening. Visual aids are helpful when discussing with people whose technical knowledge is unknown.

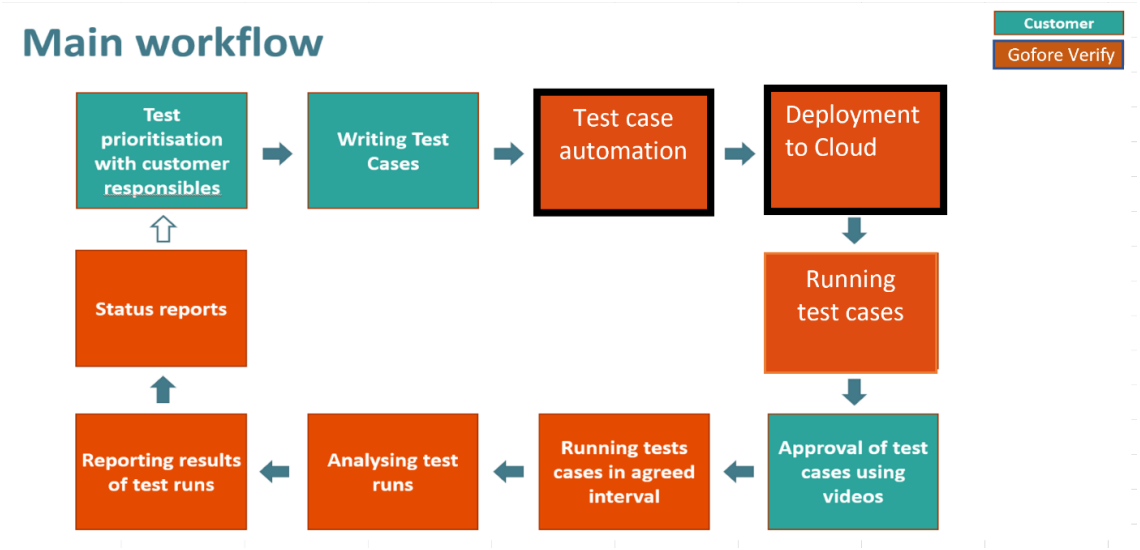

Figure 1. Example of a Main level Workflow

More detailed workflows can and should be used with a more technically oriented audience in mind and within the development team itself. Main-level workflows should be understood by the team, but their main work is done within the subprocesses of the workflow. Below in Figure 2 there is an example of a test automation subprocess, It explains the steps from the Main workflow that have been bolded. It deals with more day-to-day work of the automation team.

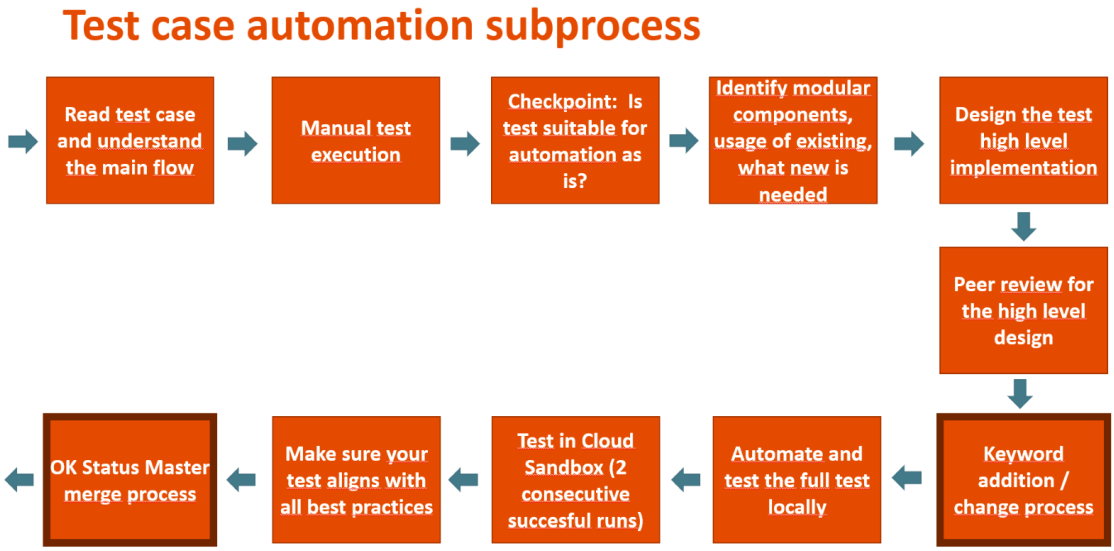

Figure 2. Example of a Test automation subprocess

This test automation subprocess includes a couple of special steps. For instance, in this process, all new tests are given a skeleton that is reviewed by the team before any actual scripts are written. This type of skeleton/code review is useful if many action keywords are used in the automation process. It is also useful if a theory of automation is to produce similar scripts between the automation consultants. These extra steps are something that should be agreed upon and written down in the BP document.

Creating the test automation subprocess workflow should be done within the team so that all team members can participate in the creation of it. This can motivate the team members as they can see their ideas implemented into their day-to-day work. Test automation consultants are also usually more knowledgeable about possible problems and their solutions within the automation process.

4.2 Project Management

Project Management is one of the most important cornerstones when handling Best Practices for any project. This is because management can make or break the implementation of BP in other aspects of the project lifecycle. Management handles the workload and workstyle of the project as well as outside communication of progress. Well-planned project management will lessen the workload experienced by the whole team. With proper communication of requirements and responsibilities management can affect work motivation and work quality.

Proper management does not happen quickly. It cannot be planned at the beginning of the project and then not touched at all, nor can it go with the flow and not have anything written down. This chapter goes over different points of the project lifecycle and different aspects that Project management to consider if a properly functioning project is prioritized.

4.2.1 Before Project Begins

Before a project begins the Management-team should scope out the specifics of the project. This should include meeting with the product owner, finding out their specific expectations, and getting to know the project in the product owner context. This includes the timeline and how much feedback they want from the testing team. This is also the time to scope out if what they want can be implemented fully or partially with test automation. Finding out the product owners' wants, and goals makes creating a testing plan more efficient. Not all testing can be automated as automation requires a stable environment. Not doing automation means that more manual testing is needed. Manual testing is good for finding quickly bugs from new features but is less useful for repeated regression testing as manually done repeated tasks can lower work motivation and work quality.

Communication with the product owner will also minimize risks, or at least make the team more risk aware. Identifying risks will decrease the likelihood of them affecting the timeline of testing. It will also make the team more prepared if these risks were ever come up during the project.

This identification process includes the following points:

Schedule related risks such as:

- Work does not get done in an iteration, spills out
- Poor tracking of key resources

Budget-related risks such as:

- Budget estimation not done correctly
- Costs overrun
- Project scope expansion

Operational-related risks such as:

- Not enough resources to be utilized
- Conflicts between project personnel and other product owner organization personnel
- Lack of clarity in roles and responsibilities
- Not enough trained resources or training is not done
- Poor productivity
- Poor test case quality

Technical-related risks such as:

- Requirements change from iteration to iteration
- Implementation is too complex
- Environment is not enough powerful
- Poor code quality

External related risks such as:

- Market change or rapid development of the market
- Rules and law changes/regulations change
- Change in consumer behaviour

From the product owner's perspective, the most important part is visible progress. Much progress can be made without it being visible to people outside the development team. These possibly invisible tasks can include for example custom library development, setting up version control, and getting required testing rights. These alongside just simple creation of test automation script needs to be shown to the product owner. This can be done using different methods:

- visual tools
- emails
- calls
- dashboard

Product owner awareness over timeframes and current workload is also important to minimize overworking and underworking the team. A version control system and an end-to-end toolchain to deploy and develop software make development more transparent and trackable.

4.2.2 During Development

While a project is ongoing, it is the responsibility of the project management team to take care of project health. This includes keeping up with team satisfaction and team commitment. To keep the project healthy, people working on it should be aware of the project goals. These goals should include small and large achievements. Small achievements keep the team motivated as progress is more visible.

This can be achieved with dailies or weeklies where progress can be shared, and problems can be troubleshot as a group.

On the product owners' side, visual progress is just as important. If no progress is being made available to them, they cannot be prepared for possible delays or problems with given tasks, they also cannot know if more progress is being made than was estimated.

The product owner can either get weekly or monthly updates on how work is progressing or if there is a system where made progress is being made visible, they can be given access to that. However, giving access to logs and progress does not replace progress meetings.

4.2.3 Commitment

Commitment and taking responsibility for the project is a part of project management, but everyone on the team must take responsibility for their work and project progression. Thus, it is important to get the team members to become committed to the project and its goals. This is often done by defining goals for the project. Everybody should know the set goals and be committed to achieving them. This can be achieved by talking openly about goals and their quality properties and planning things together. Joined decision-making makes people motivated and thus committed to the project. Each of the team should make interesting questions and try to motivate people to achieve remarkable results. This can be achieved by keeping up conversations within the team and then trying out the ideas that are put forth in these conversations.

Influencing their work usually leads to people taking more responsibility and to teams being more motivated to do their work. Offering a chance to make influence the TA team and teams outside of the TA team may well be the most important factor since developing processes into more useful and practical forms has a huge impact on actual work.

To achieve successful and satisfactory results a little more than just goals is needed. Constant progress monitoring is vital to achieving desired results. When this is lacking teams can suffer from ineffectiveness, overlapping work, and unanswered questions. In these situations, there are usually reasons within the team as to why the collaboration has failed and getting to know those reasons is necessary to move forward with the project.

Considering properties of the decision model in use is vital before one is chosen. In some circumstances shared responsibility model offers great power and it is effective to bring hoped results. Here shared decision model is defined as a responsibility that is spread across the TA team. The team is more relaxed about responsibility and can focus on main deliverables. Also, the team can produce joined decisions and solutions compared to personal solutions. The

team-based approach has its weaknesses. One of the biggest is people avoiding taking responsibility since there is no personal responsibility. [24]

The key is to get the test automation team to follow development goals and guides. There are a few steps that make this easier to achieve:

- Early guides for development and monitoring constantly the process
- Interesting questions
- Motivating people to follow set guides and rules of development
- Open talks about the spaghetti situation (direct follow-up of not following guides)
- Project is planned so that team members can track events and coordinate their work

4.2.4 Active Team Meetings

In today's climate of work, it is easy to take part in a virtual meeting and slip into the role of a listener or an observer. While it is important to listen to others, it can also become a hindrance if more people do not take an active role in the meeting. This hindrance will lower the productivity of the meeting and possibly cause the need for another meeting. To combat this, one can actively mitigate the passiveness of other participants. Hale, J. and Grenny, J have created a technique for exactly this. In their model there are five steps to having more active time in the meeting:

1. Meeting Attendees need to understand and be engaged to the problem or topic. This helps with engagement and creates an environment where problem solving can occur.
2. Everyone in the meeting has to have a meaningful role, they should not be left to passively listen.
3. Small groups with assignments. Time limits for problem solving to keep interest up.

4. Using as few slides as possible. A monologue with a huge slide deck will guarantee that most attendees will not take an active role.
5. Requiring activeness by giving problems to solve and minimizing time without doing any problem-solving task. [25]

Such techniques can be useful to establish a more active and productive team meeting. Larger meetings lead to fewer people being active on their own so in a large meeting, one should try to ask each participant their opinion and thusly add them to the conversation.

Slide shows in general are not recommended as they establish a more passive role to the listeners. So instead of passively following a slide show, one can have a list of topics and ideas about those topics visible to participants. This will make taking up a passive role in the meeting more difficult. During ordinary meetings, the above five rules can be applied. It is always better to leave slide deck templates out of meetings where they are not necessary. While helpful to the organizer, they will become boring to the listener if used too often. Once the listener becomes bored, it is exceedingly difficult to get them motivated and active participants again.

A common issue with team meetings is when a second meeting is required. The team agrees that it is needed, but no actions to organize one are taken. This can be fixed by making an agreement with the team and scheduling the time and place during the current meeting. Never end any meeting without planning the next one with the team. Make members feel responsible by asking straight how and when they can meet.

If there is vital information given out in the meeting, it should be recorded. This ensures that no information is lost. It is the meeting organizer’s job to record the meeting if nothing else has been agreed upon. Recording meetings is recommended if not all team members can participate. This way they can easily be given the information discussed during the meeting. Information sharing is vital when working towards a uniform team with agreed goals instead of individual experts with their ideas and goals.

Sharing thoughts and experiences makes everyone feel be part of a team and this in turn can induce better commitment. Open discussions can produce a healthier working atmosphere since there are conversations and members are being encountered. Everybody can openly say their opinions and participate in decision-making. Everyone will be heard. Knowledge sharing between team members makes it more homogenous.

In large test automation projects, entropy will grow like in a thermodynamics system. This means that the project loses its structure and is going into a mess. This is inevitable due to a large amount of test automation engineers with personal preferences. If there are no rules to guide test automation development many different solutions will arise. This may be good if versatility is preferred. Versatility may induce many candidate solutions where engineers can pick the closest to defined quality properties. But versatility can create problems since solutions are not uniform or aligned. There may even be unreadable test scripts with mixed layers of keywords.

To mitigate inconsistency, it is inevitable to share knowledge between test automation professionals within the team and outside of the team. Also, early defined rules or guides are very effective and useful. It is good to have meetings where team members demonstrate their weekly output and can discuss them openly and without too much criticism. This kind of discussion is too late to have during version control pull request reviews since the test automation developer has already taken their decisions. However, reviews are better than nothing. One issue with demonstrations is to maintain these meetings in a calendar continuously instead of just a few times. This kind of issue can be solved by making changes to the culture and how things are developed and decided. It could also be suggested that test automation needs its architect that makes designs of keyword layers and how each layer can be used during the development of automated tests.

4.2.5 Reporting Outcome

Reporting to the product owner should start with the project and not be left as the last thing done for the project. This includes regular reporting sessions with the product owner or other stakeholders. This ensures access to up-to-date documentation with the project progression including goals achieved and implemented. Meetings also make more apparent and inhibit problems from accumulating.

Keeping up stakeholder engagement is important, and it can be easily achieved by letting them have a meaningful and responsible role in the project. This includes all test automation and team members. Steering people of all levels to work together toward common goals ensures that the team stays motivated and responsible. This helps to keep attention to quality and attention to goals agreed upon by the team. Another good way to provide reports is to set out with a reliable meter and/or set of gauges to measure the quality and quantity properties of agreed goals. This is based on the Lean principles that have been proven to work. [26]

Currently, product owners tend to use logs of automated test runs to make informed decisions. This may not be as effective unless a history analysis of log data is first performed. This could be done by a system- or experienced engineer who regularly monitors those logs. But per se individual test automation run logs are not usable in the context of making informed decisions. More detailed knowledge of software development plans and effects of recent implementation needs to be available, and it has to be taken into consideration.

4.3 Test Architecture

Test architecture refers to how a test automation project is built internally. This includes everything from folder structure to naming conventions. Architecture is needed to stop the development from sliding into chaos. Figure 3 illustrates a table of variable architecture.

Suite Variable name	Desription	Subject area(s)	Input	Default	Output	Example data
${NIMIKE}	Item ID	Item	X			Z5100002
${NIMIKEKUVAUS}	Item description	Item	X			VEITSITERÄ LEIKKAUS KK STER NO 10;IRTOTERÄ FEATHER LTK/100.
${NIMIKELUOKKA}	Item class	Item	X			HF02001 RUUVI-IMPLANTTI TARVIKE
${MITTAYKSIKKÖ}	Unit	Item	X			Laatikko
${MÄÄRÄ}	Item quantity in sales order	Item	X			3
${MÄÄRÄ_OSTO}	Item quantity in purchase order	Item	X			3
${MÄÄRÄ_VASTAANOTTO}	Item quantity in goods receipt	Item	X			2

Figure 3. Table of variable architecture

Any type of visual representation of Test Architecture enables simpler maintenance. Even something as simple as the creating a spreadsheet with variables that are in use and how they are used, will inhibit the creation of duplicate variables.

4.3.1 Libraries

Test Automation requires automation libraries and with web-based automation, the Selenium library is currently used by default if the product owner does not have a preference. This can change in the future. For expanded Robot Framework testing, one should refer to RF standard libraries before checking external ones. Preferentially the team should use libraries that the team has previous knowledge of, to ease the integration process.

In searching for new libraries, one should always refer to open-source libraries that are actively maintained. If in doubt about licenses, a good rule of thumb is if a license is OSI approved and then check if the product owner has set some limits on what licenses can be used. [27]

4.3.2 Common Principles

The main principle is that tests should be written as readable test scripts. e.g. using standard keywords as much as possible and avoid creating complex structures with action keywords. This will make test scripts more readable.

- Indentations. Replace tab with four spaces.
- Columns are spaced evenly to help readability.
- Use spaces instead of tabs.

Below is an example of what a test should look like (Example Code 2):

```
a.1 Individual start up
  # Henkilölle lisätään pakolliset tiedot
  ClickText            Henkilön rekisteröinti
  DropDown             Rekisteröinnin laji         Henkilön perustaminen
  DropDown             Tunnuslaji                  Henkilötunnus
  …                    Henkilötunnus
  TypeText             Tunnus                      010144F3EW
  …                    Henkilötunnus
  TypeText             Sukunimi                    Kuttunen
  …                    Kutsumanimi
  TypeText             Etunimi                     Hannu
  …                    Kutsumanimi
  TypeText             Katu                        Mannerheimintie
  TypeText             Talo                        40
  DropDown             Kunta                       Helsinki
  ClickText            Tarkista: Kotikunta.
  DropDown             Kansalaisuus                SUOMI
  DropDown             Lähde                       Virkailija
  TypeText             Kansalaisuuden alkupvm      01.01.1944
  TypeText             Postinumero                 00100
  ClickText            OK
  ClickText            Seuraava
  ClickText            Sulje kaikki
```

Example Code 2. How properly spaced Code should look like

Action keywords can be considered in long test scripts with steps that are repeated identically in multiple tests. Using that logic increases re-usability, which in turn helps keep the total number of keywords down. Many keywords lead to situations where nobody remembers what is implemented and people keep reimplementing existing keywords again and again. To avoid

situations like this comprehensive documentation is required, as well as detailed merge descriptions with code review.

Variables

The use of variables and arguments should be limited within the test scripts. The avoidance of variables makes reading the test script easier.

- Values that need to be set dynamically, eg. ${TODAY} and values read from the application and needed later, must be stored in variables
- Parametrizing test suites for different environments can be done using variables
- Usernames and passwords should always be stored outside test scripts
    - Access them using variables
- Testing localization can be done using variables
- Same value (e.g. input) is needed multiple times in the test
- There is a need to call the same test with different values (e.g. input)
- Limit the number of tests, suites, and global variables to a bare minimum.

Local variables should be avoided in action keywords and values need to be passed as arguments when outside data is required. This keeps the data and logic flow clean and understandable.

Condition and Loop statements

Avoiding complex conditions and loop statements in test scripts makes code more readable. Because of this, it is better to divide some very complex and branching business scenarios into multiple different straightforward tests instead of implementing everything in one test. That way tests remain maintainable. Conditions and loops can be used though when iterating over lists or test data in random order.

In complex scenarios, custom keyword implemented in Python is usually the best choice, as they can be a lot faster than robot-only implementation providers better features for programming, and complete control over logging.

Action Keywords and when to use them

Action keywords should not be the first option when scripting. If the following criteria are met, action keywords should be considered as a solution.

- Test would have very many steps if written in an atomic way (hundreds or even thousands)
- There are logical actions with clear and constant boundaries
- Same logical actions are needed in several different tests
- Number of variations that would reduce the reusability is limited
- When action keywords are used some guidelines need to be followed. The guidelines can be split into a couple of reasons why they are important:
- Helping with reusability
- Preventing overlapping / duplicate work

Usually, one keyword should do only one logical action at the application under test. This can be for example creating a sales order. Limiting the logical actions done by the keyword helps with reusability. In the example is a code that adds items to a Wishlist. This is something that is repeated hundreds of times during webshop testing, even multiple times during a single test, and is a prime example of code that should be made into an Action keyword (See Example Code 3).

```
ClickText           ${ADD_TO_WISHLIST}
VerifyText          ${ADDED_TO_WISHLIST_VERIFY}
VerifyText          ${WISHLIST_WARNING}
VerifyText          ${SHOW_WISHLIST}
VerifyText          ${CONTINUE_SHOPPING}
VerifyElement       ${WISHLIST_ICON}
```

Example Code 3. Locators placed into Variables

Naming conventions need to be followed when creating a new action keyword. A good place to start is to name the keyword by the subject of the action followed by the action verb (e.g. SalesOrderCreate and SalesOrderModify instead of CreateSalesOrder and ModifySaleOrder) and store keywords in alphabetical order. This will help avoid the creation of duplicate/overlapping keywords.

Defining a review/approval process for the keywords creates a quality assurance to the scripting process and can mitigate future problems when more than one person in the team knows of the keyword.

End-to-End (E2E) business process testing with for example Enterprise Resource Planning (ERP) and related systems is an example of a case where using action keywords can be a good choice.

The case against XPATHs

These guidelines assume that Robot Framework is used, with other libraries possibilities can be a bit different. XPATHs are a bit difficult. They break more easily than other commonly used locators and are difficult if not impossible to read. For that reason, they should always be the last resort when locating elements. Robot Framework allows for a wide variety of locators to be used, that end up with a more readable solution. In case there are multiple hits, indexes and anchors can be used.

In case of text is not the best solution, one can also use tags and other HTML identifiers to reach the specific element.

If everything else fails, XPATHs can be used. Then the XPATH needs to be established so that restructuring elements on the tested page do not immediately break it. This means that instead of copying the (full path of) XPATH using Developer Tools in the browser, the XPATH should be constructed by using the element type and/or the path from elements close by.

4.3.3 Naming Conventions

Naming should be done in the project language or following the language conventions of the project. This is the reason why the project language should be established at the beginning of the project. If no language is established, English should be used as a safe default.

Things that may influence language choices:

- Product owners' requirements/preferences
- Language(s) of the system under test
- Language(s) of the project group

It is also possible to create a hybrid solution that names directly related to the system under test are in the language of the system and everything else is in English.

Generally, it takes considerable extra effort to support doing test automation in multiple system languages. Usually, it is not worth the trouble. This must be defined very early in the project. As it will have a big impact on the needed implementation and maintenance work. A common solution is to make all texts in the automation script variables and store those UI text-related variables in each language into one or multiple variable files.

In general, naming should be done in a way that the reader will easily be able to understand what they are looking at. For variables, this needs to be descriptive like ${current_date} for dates. In the case of tests, the test name should tell immediately what the test does. This does not need to be detailed as it can be

expanded upon within the documentation of the test case, but it should be descriptive.

For Custom keywords this can be done in multiple ways. Custom keywords should be named with a descriptive name as possible. In Python code, one would use mainly Python function naming conventions (all in small cases, words separated with underscore). This is to ensure that custom keywords stand out better from standard keywords.
Examples of descriptive naming include:

- Date_First_Of_Month
- Number_Change_Format
- SearchField_Add

4.3.4 Folder Structure

The folder structure is something that might not seem that important in the beginning but in the case of a large product owner project a clear and easily understood folder structure will make working on the project a much nicer and less stressful experience as everything can easily be found. In a professional project, different structure levels should be used.

Below is an example folder structure for a simple project. There are structures for each environment. Libraries, resources, and test scripts are in the second layer.

Libs folder contains standard and system/environment-specific libraries. Library-specific test cases are located here.

The resources folder contains the implemented keywords as well as project-specific resources.

The tests folder contains test scripts and test data. Each area contains sub-areas and test data which covers all scripts belonging to the area. Sub-areas include short lists of the area. In the below example (Example Code 4) a simple version of Library subcategorizing is presented visually.

```
Environment_XX
        |
        |--- Libs
        |       |--- Sut
        |       |--- QWeb
        |       |--- LibTests
        |
        |--- Resources
        |--- Tests
               |--- Area1
               |        |--- SubArea1
               |        |      …
               |        |--- SubAreax
               |        |--- TestData
               |    …
               |--- AreaX
```

Example Code 4. Proper Folder Structure

No matter how complex a Project grows during development, a clear Folder Structure will keep work more organized and help with a more concise workflow.

4.4 Library Development

As was discussed in the Architecture subsection, test automation usually runs on libraries. Usually, these are developed to do specific things, for web automation there is have Selenium, for Excel automation, Excel library, and so forth. There usually exists a library for most purposes. If this is not the case or when a project has grown massive a library development can take place. Library development can also refer to a library of Action keywords that have been created for a specific client. [28, 29]

Library development can be used to ease future test automation. This is done by taking current solutions and making them easily accessible to other projects or other platforms. Library development can also be more project-based, this makes them usually tied to their development environment. In both types of library development, the goal is to minimize future work by creating simple, easily accessible solutions that are as general as possible depending on the scope of library development.

Library development should be based on a few guidelines: A well-developed library architecture and documentation.

4.4.1 Knowledge Sharing

A low threshold for knowledge sharing needs to be used. New code and ideas are created daily to solve problems for product owner projects. This creates a high probability that there are existing solutions for many problems encountered daily. Thus, whenever a solution is created it should be shared with others to minimize overlapping work. This is the easiest way to share knowledge and for the developers to benefit from their colleague's work. Knowledge sharing can even heighten work motivation as it can shorten the time spent struggling with issues that turn out to have ready-made solutions.

4.4.2 Architecture in Library Development

As in everything else that gets done in test automation, in library development proper architecture needs to be developed and implemented. This is done to ease the use of the developed library as well as ease the learning of the new library. The proper architecture includes developing test automation libraries for a single problem for a single platform. To ease the development of simple architecture the Innovation team came up with the following three rules to be considered.

1 Self-explaining solutions

- Naming for implemented keywords is an important factor. Users should understand after one look what is the purpose of a given keyword. One good practice for naming is to use library-specific prefixes This way the same self-explaining naming policy can be used for every library and avoid naming conflicts.
- Minimizing the number of required parameters for any keyword to make it simple to use.
    - If more than three parameters are needed by default, it should be considered if the given action is possible to be split into smaller parts. Maybe using two simple keywords, instead of a single but complex one, would be easier for the users.
- The principles also apply to the code base.
- Readability and simplicity over unneeded complexity
- Using project / language-based logging to help with later debugging

2 Goal should be always to create as general and simplified solutions as possible.

- Reusable libraries and a decent level in terms of reusability and simplicity. It is not good practice to create libraries that might solve all problems in one package for one project.

3 Handling dependencies

- Regardless of if the library is an independent whole or some wrapper/extension to complete some existing libraries be aware of what kind of dependencies are used and how to limit those as minimum as possible.
- Start your file listing using imports instead of doing imports inside of functions/methods.
- No unused imports to code should be left in the library. Test automation languages have examples of group imports that should be followed.
- Keeping a list of needed dependencies and using requirements.txt to define those needs.

When all the above rules are carefully considered and applied accordingly, the resulting library becomes infinitely easier to use. A well-crafted library will make future work easier and faster. Now all that is left to do, is to create a documentation for the library that makes people want to use it.

4.4.3 Documentation in Library Development

In library development, like in any development, documentation is of importance. Thus, all keywords and their use should be documented. Not all detail needs to be documented as in library development it is understood that future users have at least a basic knowledge of automation libraries.

In library development the minimum documentation includes the following:

- Usage information
- Required parameters
- Optional parameters
- How to change the parameters if possible

There should be always at least README and release notes included with any library. [30]

The bullets listed below are the minimum for a good README:

- What this library does and what are the problems you can solve with it?
- Specific requirements regarding operating systems, software, or execution environments
- Instructions on how to install and/or take it into use.
- Update README and user instructions always when making changes to an existing library if the changes affect behavior.
- If the library uses some third-party libraries, interfaces, etc. and there are links to sources and upper-level documentation available add those links to the instruction part.

4.4.4 Coding Rules

Programming should always follow some common guidelines. This includes arranging all rows to fall into columns like in the following example (Code Example 5).

```
${rivi}=            GetTableRow             ${nimike}
VerifyTable         r${rivi}/c1             ${nimike}*
```

Example Code 5. Example of code arranged into columns

Whenever publishing anything to a shared library, repository, or modifying already existing code, one should always check that the written code is up to standard. This can be done easily by using static analysis tools like Pylint for Python. [31] If there are no static analysis tools available, a code review should be organized to make sure that the code is understandable and up to standard.

Things to keep in mind while using variables are:

- It is good practice to use actual (correct) words in variable and method naming
- It is good practice to use a spell-checker while coding to avoid bugs caused by misspelled variables etc.

The purpose of this is not to make the publishing process heavy and difficult, but to make sure that everyone is following certain standards and best practices of used programming language to keep the shared code as clean and readable as possible.

Being consistent with naming conventions and programming styles throughout to library will ease the learning and adaptation process for other people. Libraries are meant to be used and modified to fit their purpose better, so keeping the quality consistent will ensure its usability in the long run. The more consistent it is, the more likely it will stay that way in the future too.

4.5 Documentation

Documentation means many things in the programming world. It can be the comments scattered throughout the written code that explain how specific parts of code function and what they are for. Documentation can also be a separate

file that contains information about the file itself. How it is used and when and what has been changed. Generally, all types are vital, just for different situations. Proper documentation makes it possible to keep the project owner informed about the daily progress of the project. It can also be used whenever a new person joins the project as a start of training.

Professionalism is key when working on any project. Documentation is one part of that. This needs to be kept in mind as the team decides on details about the documentation and how it is presented. This chapter goes over some marks of professionalism in the documentation.

4.5.1 Preplanning

Before starting the documentation, some of it needs to be planned. A few points to consider is that documentation should be readable and that different parts of it should not include overlapping information

There is no reason to write one conclusive documentation that contains hundreds of pages with changing themes. Documentation should be kept short, simple, and with the least overlap possible. This minimizes the risk of having major errors hiding within the pages and keeps documentation easier to maintain. When referencing other documents containing the same information, provide a link or tell which page from that document contains the reference.

4.5.2 Automatic Documentation

Auto generating technical documentation is a great tool and there are no reasons not to use it. Regardless of the test automation project approach, manually creating technical documentation is not worth it. It is prone to

errors. Not all errors can be avoided, but they can be minimized with the proper application of automation.

There is always the risk that the technical documentation does not match with reality, where the code was changed but not the docs have not been updated. This is still easier to fix than having the whole documentation manually provided.

The real benefit of automatically generated documentation is how the software development and technical document development can be changed. Here the developer does not have to exit the code editor to write the technical documentation, as the documentation can be inserted straight into the code (See Figure 4).

```
Test Job localization
    [Documentation]       Check that localized job listings pages have translated content
    [Tags]                mobile    careers
```

Figure 4. Example of inserted documentation

Choosing the right tool depends much on the choice of other tools and coding language. It is also recommended to use the most used documentation tools because other users can easier use the documentation. The above example is an example of the most simplistic type of inserted documentation. For anything more complicated a more comprehensive description should be given with examples of use and function.

4.5.3 Visual tools, Illustrations, and Other Drawings

A picture is worth a thousand words. Sometimes many pages of complex information and IT jargon can be turned into a simpler picture that offers more value. There are a different set of illustrations that can be used efficiently as

well as a high-level discussion starter. The figure below (Figure 5) shows a timeline of product development that shows how different business units push forward their work simultaneously. This would be inefficient to explain in words, so a visual aid is used.

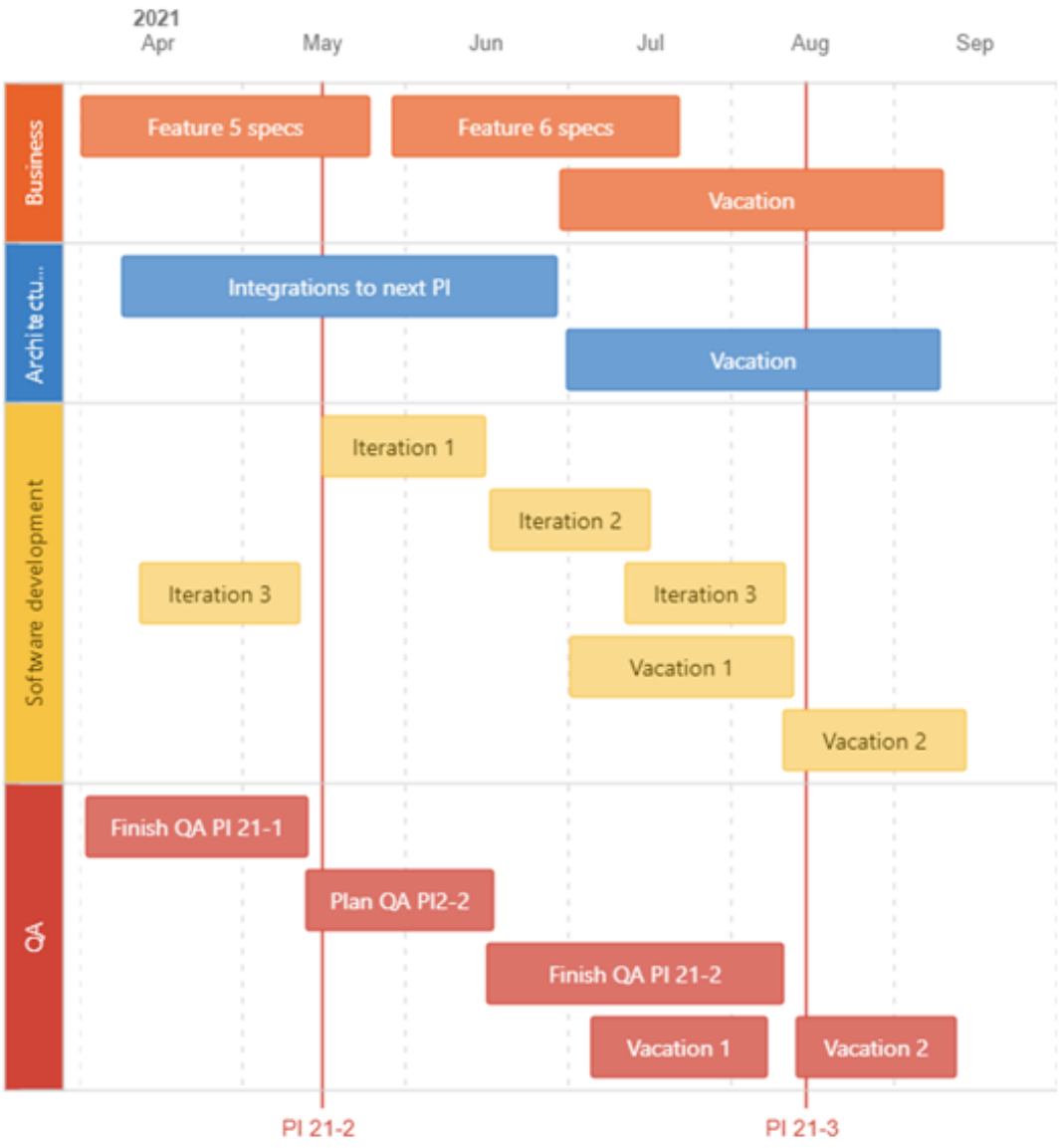

Figure 5. Diagram, Timeline

Different types of illustrations, such as mind maps and graphs can be used to visualize workflows and software architecture for people in all stages of

development. In general visualization tools should be used when complicated information is recorded to ease understanding.

4.6 Version Control

As stated earlier in Chapter 2.5 Version control, is a prerequisite for a functioning project the implementation of anything new into the main code needs to be regulated. This is done using version control. To use version control efficiently some basic rules, need to be decided upon. Mainly this considers who is allowed to make decisions over new implementations or if those decisions should be done with the help of the team.

This portion of the Best Practices opines upon the requirements of version control. The needs of a project are largely influenced by the project size and the number of team members. In a small project with a few team members rigid version control might seem unnecessary but with every added person the likelihood of conflicts from unregulated version control rise. To combat this more detailed merge requests and code reviews used together should be implemented. These are both standard practices on the software development side, so much so that there are automated solutions for it [32]. However, this is something that is often left out in TA solutions. Mainly because only the team will ever see the code and others will only be given results once the code has been executed.

Test automation can only benefit from a wider acceptance of code reviews and detailed merge descriptions. When used together they improve code quality and team knowledge over implemented code and especially implemented library development including new keywords.

When it comes to a smaller project, not all rules need to be followed, but even the tiniest of TA developments will benefit from detailed merge notes. They can function as a work diary and can be used to make work visible to others.

4.6.1 Merge Description

Merge description refers to the description given in the merge request when a new code is finished and gets implemented in the master code. Comprehensive merge notes are the easier and less time-consuming way of informing the team of made changes, an assignee will go over the merge request and either allow it or give feedback on fixing it.

Below in Figure 6 there is an example of what a merge request can look like.

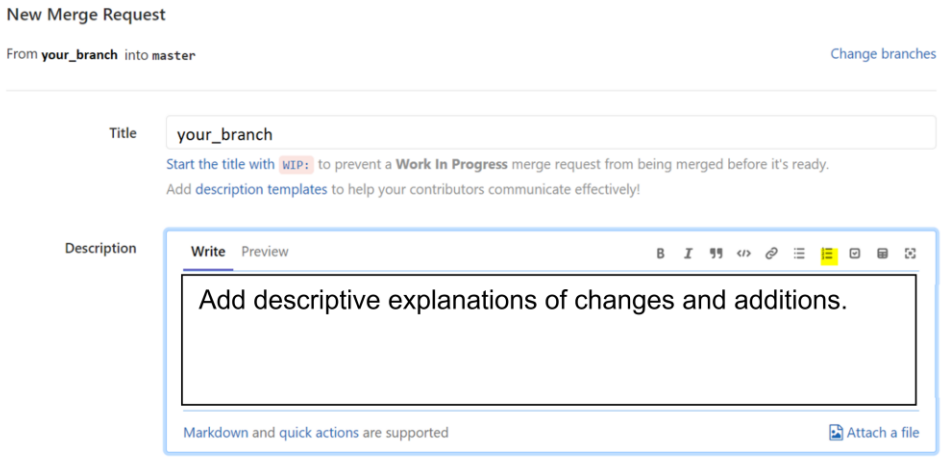

Figure 6. An Example of Merge Request in GitHub

Written merge description that explains the changes made are important when conducting code reviews, and it is best practice to have an assignee check over the changes. In code review, the point is to go over interesting bits of code and discuss if there is anything that could be done better. In these code reviews, the project naming conventions and scripting conventions are also checked. Whereas in merge descriptions of the changes made and their reasonings are given. The merge description then works as the skeleton for the code review and if it is done well, no actual meeting might be needed. A simple email on the mistakes or possible changes can be sent.

4.6.2 Code Review

Code review is a widely used tool in program development for increasing code quality. It is a way of quality control and learning as team members go through each other's code and give constructive feedback on it. It is something that is often not included in TA processes as they are not shown outside the development team. This type of functionality over quality ideation leads to poor quality code that breaks easily and is difficult to fix. [33]

Especially in projects where TA runs are long or linked together, implementing a code review is a tool for streamlining the merge process. It also ensures that all code has been checked for compatibility and its functionality is understandable by others. In projects that implement keyword libraries code reviews are essential as it functions as a checkpoint where changes to existing code can be looked over by people whose work could be affected by a poor understanding of existing work. It is always more productive to stop a bug before it is implemented than to track it down after everything has broken down.

In these reviews, another team member goes over the new changes. This is to ensure that the new merge aligns with the best practices of the project. This is also to make sure that another team member knows at least on a surface level what is happening with each new test case.

Code reviews can be useful in large projects that have rigid rules and when someone has just joined the project. They can also be used to teach other members how some difficult pieces of code function. Code reviews are more often seen on the programming side of software development, but it does not mean they cannot be useful in test automation.

So, in short, code review should always be done before a merge to master. Whether a meeting is needed or not is up to the test automation team.

In most cases, version control should not be just developers merging their branches without any merge descriptions and without anyone else checking over the changes.

5 Discussion and Conclusion

The study was carried out with the explicit hope of making day to day work of software development teams easier. That expectation lead the project into being centralised around two main issues concerning TA. The first being invisible work. It includes all day-to-day tasks that are not directly reported or will never become public to others. This need to make the work visible caused the work to prioritise types of documentations. In the end this was accomplished.

The other issue concerns the lack of role clarity in TA development. Today the IT field seems to be moving towards a more comprehensive type of work. This is clear when it comes to the TA side of testing. The work is not only Quality Assurance, but also development of scripts and libraries. This often leads to the TA developer wearing multiple roles in the project. There are times when a single developer must function as the Test Architect, the Test Manager and the Test Automation Specialist. To prioritise work satisfaction and work life balance, clear boundaries must be in place. The study was done to help clear the different aspects of each role. The effectiveness of this remains to be seen.

In theory and in literature the problem of invisible work comes up often. It might be because it is not just an issue with TA but with IT work in general, and thus it there are plenty of examples of how people are dealing with this, and how it concerns the BP of software development cycle.

Unfortunately, the second issue was not as widely covered in literature. IT work is taxing. It requires continuous learning, and a lot of pressure is placed on the individual. Burnout especially is described as an epidemic among the Tech

Industries, and often role clarity or more importantly the lack thereof, is seen as a possible cause to it. [34] However, it was not prioritised in any of the literature that was used as the base of the thesis. Weirdly it did come up often during the discussions with the team. Therefore, prioritising knowledge of responsibilities and clarity of communication is highlighted here. More studying of the burnout phenomenom is required.

In the business unit of the commissioner, the Best Practices rules were established in January of 2022 with a presentation and a decision to have a look at the set of rules later once everyone had had the time to try them out. Further development is naturally needed, and the BP rules need to be adapted to different types of projects and project needs. In general more research into the effectiveness of BP is needed. As explained in Chapter 3 the current study was the work of one business unit specialising in testing through Test Automation and while the experts were picked from multiple customer teams, the spread of experience was still not large.

The limits of the thesis are the scope and reliability of the BP guidelines as they only consider the ideas from one team in one company. It can also be that other companies with similar business units do not consider the problems identified here as high of a priority. Thus, the thesis and the BP documentation in it might end up useless to them. In the future, the present study can, however, be used as a starting of point to further research, but more importantly it is useful to those new to the Tech world. For students and career changers the areas outlined in the thesis should function as a thought experiment on how a project could be handled.

During the study, the main conclusion that the innovation team came to was that guidelines are helpful as long as they are used to guide, and not as a checklist. For BP to be effective they need to be customized for every project, and possibly changed during the project if they are found not function as expected, or if they do not suit the workflow of the team. This document does not go into too much detail. It is meant to be a simple overview and that is also

its weakness. A more detailed, more technical assessment of tools and practices is needed in the future. Alongside that, the guidelines must be tested in a proper environment and not just in a small innovation group determining what is best over a large group of people.

For future studies, it would be helpful to put these guidelines in place and then do surveys to gauge the effects they had. The BP can affect innovation, work motivation, and accountability if used correctly, and in the future, it should be looked into, how much it affects people. Another future study should consider adherence to the guidelines. Even if they are established at the beginning of the project, their use might fall off. For a longer project it would be interesting to see how the processes have evolved in 1-3 years of being used or forgotten in a project.

References

1 Merriam-Webster. Dictionary. Checked 19.9.2022. https://www.merriam-webster.com/dictionary/best%20practice

2 Merriam-Webster. Dictionary. Checked 19.9.2022. https://www.merriam-webster.com/dictionary/quality%20assurance

3 IBM. Article. Checked 19.9.2022. https://www.ibm.com/topics/software-testing

4 Kujala, Anna-Liisa. 1995. Masters Thesis.

5 Hamilton, Thomas. 2022. Guru99. Checked 19.9.2022. https://www.guru99.com/automation-testing.html#which-test-cases-to-automate

6 Robot Framework. 2022. Documentation. Checked 19.9.2022. https://robotframework.org/

7 Robot Framework BuiltIn. 2022. Documentation. Checked 19.9.2022. https://robotframework.org/robotframework/latest/libraries/BuiltIn.html

8 Qentinel. 2022. Documentation. Checked 19.9.2022. https://github.com/qentinelqi/qweb

9 APM. Checked 19.9.2022. https://www.apm.org.uk/resources/what-is-project-management/

10 Aston, Ben. DPM. Checked 19.9.2022. https://thedigitalprojectmanager.com/why-is-project-management-important/#:~:text=Project%20management%20is%20important%20because%20it%20brings%20leadership%20and%20direction,to%20do%20their%20best%20work.

11 Shawn, J. 2021. Testim. Checked 19.9.2022. https://www.testim.io/blog/test-architecture/

12 Qentinel. Documentation. Checked 19.9.2022. https://help.pace.qentinel.com/qwords-reference/current/qwords/advanced/appstate_qweb.html

13 Editorial Team. 2022. Bit.AI Checked 19.9.2022. https://blog.bit.ai/it-documentation/

14 Editorial Team. 2022. Bit.AI Checked 19.9.2022. https://blog.bit.ai/technical-documentation/

15 Editorial Team. 2022. Checked 19.9.2022. Bit.AI https://blog.bit.ai/project-documentation/

16 Performance Lab. Checked 19.9.2022. https://performancelabus.com/importance-of-software-testing-documentation/

17 Atlassin. Checked 19.9.2022. https://www.atlassian.com/git/tutorials/what-is-version-control#:~:text=Version%20control%2C%20also%20known%20as,to%20source%20code%20over%20time.

18 Lukka, Kari. 2001. Method Article. Checked 19.9.2022. https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/

19 Kasanen, E., K. Lukka and A. Siitonen. 1993. The constructive approach in management accounting. Journal of Management Accounting Research Vol 5, pages 243-264

20 Wright, Gavin. 2022. TechTarget. Checked 19.9.2022. https://www.techtarget.com/searchsoftwarequality/definition/best-practice

21 Fimea. 2021. News. Checked 19.9.2022. https://www.fimea.fi/-/laakinnallisia-laitteita-koskeva-uusi-eu-asetus-voimaan-26.5.2021#:~:text=L%C3%A4%C3%A4kinn%C3%A4llisi%C3%A4%20laitteita%20koskevan%20Medical%20Devices,sek%C3%A4%20implantoitavista%20laitteista%20(AIMDD).)

22 Fluix. Marketing. Checked 19.9.2022. https://fluix.io/help/workflow-definition

23 KissFlow. 2022. Checked 19.9.2022. https://kissflow.com/workflow/what-is-a-workflow/

24 Beyond the Boardroom. 2017. Checked 19.9.2022. https://www.beyondtheboardroom.com.au/blog/2017/how-to-improve-teamwork-through-shared-responsibility

25 Hale, J. & Grenny, J. 2020. Checked 19.9.2022. https://hbr.org/2020/03/how-to-get-people-to-actually-participate-in-virtual-meetings

26 Do, Doahn. 2017. Lean Way. Checked 19.9.2022. https://theleanway.net/The-Five-Principles-of-

Lean#:~:text=The%20five%20principles%20are%20considered,%2C%20and%205)%20pursuing%20perfection

27 Opensource. Licenses. Checked 19.9.2022. https://opensource.org/licenses

28 GitHub. Library Document. Checked 19.9.2022. https://github.com/SeleniumHQ/selenium,

29 GitHub. Library Document. Checked 19.9.2022. https://navinet.github.io/robotframework-excellibrary/

30 GitHub. Documentation. Checked 19.9.2022. https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-readmes

31 Pylint 2.15.0-a0. 2022. Documentation. Checked 19.9.2022. https://pylint.pycqa.org/en/latest/

32 Safak, umutcan. 2022. AImultiple. Research. Checked 19.9.2022. https://research.aimultiple.com/auto-code-review/

33 Mendes, Alexandra. Ferreira, Rodrico. 2022. ImaginaryCloud. Checked 19.9.2022. https://www.imaginarycloud.com/blog/what-is-code-review-and-when-should-you-do-it/

34 Petrova Sandra, AdevaIT. Checked 19.9.2022. https://adevait.com/blog/workplace/burnout-tech-industry