TSS Tool: Automation Tool Applied in the True Single SKU Setup Environment Preparation for Multiple Devices in Parallel

Carol Fernandes
carol.fernandes@sidia.com
SIDIA R&D Institute
Manaus, Amazonas, Brazil

Charles Araújo
charles.araujo@sidia.com
SIDIA R&D Institute
Manaus, Amazonas, Brazil

Adriano Oliveira
adriano.oliveira@sidia.com
SIDIA R&D Institute
Manaus, Amazonas, Brazil

Wellington Corrêa
wellington.correa@sidia.com
SIDIA R&D Institute
Manaus, Amazonas, Brazil

Luan Ipê
luan.ipe@sidia.com
SIDIA R&D Institute
Manaus, Amazonas, Brazil

Paulo Andrade
paulo.andrade@sidia.com
SIDIA R&D Institute
Manaus, Amazonas, Brazil

ABSTRACT

There is an extensive amount of mobile product models in the market. In this scenario, research institutes are involved in a stage previous to release, in order to ensure that the software meets quality standards, with stakeholders' requirements embedded, tested and validated. All of this in a high demand scenario. These institutes are presented with the challenge of delivering software ready for release increasingly faster than before. Replacing manual processes with automation can reduce execution time in the production chain. After analyzing our own context, we concluded that True Single SKU (TSS) mobile models were a strong candidate for process improvements. These devices' setup was a bottleneck, since execution was done 100% manually. Our objective is to assess the benefits of using an automated TSS Tool in the environment setup preparation of multiple devices in parallel. Through observational studies, we were able to reaffirm the relevance of using software automation as a strategy to reduce development time. Besides, it works in a supporting role when dealing with high demand. We observed a gain in time of up to 52% for beginner testers and 50% for experienced testers. These results were achieved when comparing manual setup activities executed with multiple devices one by one and automated setup activities with multiple devices at the same time. We concluded that we were able to save more development time and achieve more flow in the whole production chain for TSS mobile devices.

KEYWORDS

software testing, automated testing tool, TSS, mobile devices

1 INTRODUCTION

Technology projects aimed at mobile devices are evermore on the rise. Hence, the need to introduce automation to implementation processes increases. This is due to different specifications assigned to different mobile device models, which are launched every year [4, 13]. Keeping older devices' software updated and developing newer software for newer models escalates quickly. Our study scope is focused on the Android Operational System (OS). Each Android version is designed with different or updated features. Regarding these features, some require negotiations, which may involve manufacturer, mobile network operators, government and Google [3]. They are the ones who define new software requirements to be included in official releases.

Requirements implementation and their respective tests are essential, specially for the mobile device sphere. It's through these activities that high quality software is ensured. Without it, the manufacturer brand won't be trusted by it's costumers. Therefore, developers must execute manual or automated tasks in order to guarantee that customers' needs are properly met. SIDIA R&D Institute is an example of organization that deals with different mobile device models. The institute works with research projects on development and update of mobile devices' software. These Android software releases are aimed at a plethora of device models for Latin America.

The projects we execute in our institute can be classified as: (1) New Models; (2) OS Upgrades; and (3) Maintenance Releases. Our team, i.e., the User Interface (UI) Requirements team, is responsible for developing visual requirements related to mobile devices applications and their respective placement. That also includes carrier wallpapers and carrier boot animations. Since we aim to validate that all UI requirements are being met, it is of ultimate importance that we perform tests and software validations. These must be carried out when there's any update, addition or removal of requirements. And, in this scenario, developers are allocated to carry out both implementations and validations.

In this context of requirements implementation and validations, our developers execute some procedures that define new properties in specific mobile device models. These procedures establish a new context for those devices, which has to be done before validating UI requirements. This scenario where the new context is being applied is called True Single SKU (TSS) [16]. TSS is achieved by doing the following: we receive some carrier specifications and requirements that must be controlled through a provider, i.e., a server; a specific device model is connected to the server and a device input information is registered in this server; then, carrier requirements will be downloaded to the device automatically.

Currently, TSS process is performed manually, as seen on Figure 1. Developers working with a few devices register them and their respective final carriers in the server. Also, AT commands are used at this stage, as these commands play a vital role in smartphone communication via cable to a PC. They're able to extract a variety of data and control many pre-existing behaviors on devices [7]. Individually, specific AT commands are inserted into each device, with technical steps and instructions to set a base carrier that is related to the final carrier, which is configured in the server. To

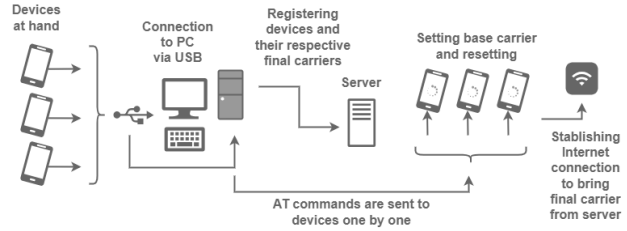

Figure 1: Overall manual process to set TSS.

finish, after executing a reset command in each device, an internet connection is established to bring the final carrier associated in each device from the server along with its carrier requirements. This process is slow and many times is susceptible to human errors.

In our paper we explain the development and validation of an automated tool to execute TSS process preparation in one or more devices in parallel. This tool will be associated with the current automated checklist tool, an internal software that applies carrier requirements, tests and validates them on the devices. Our goal is to obtain gains in terms of TSS device preparation time associated with automated checklists, reducing human errors and improving flow rate on developing and validating TSS device requirements.

Additionally, we will map which TSS models and countries (final carrier) assume certain standard times. This kind of information will help to identify the use, in the future, of a predictive model to suggest the best configuration of candidate TSS models to be executed in parallel by the developer, optimizing the process and achieving the due date with efficiency.

This work is sectioned as follows: Section 2 - Theoretical Background, Section 3 - Current Scenario and Our Proposal, Section 4 - Methodology, Section 5 - Results and Discussion, and Section 6 - Conclusion and Future Work.

2 THEORETICAL BACKGROUND

In a world of technology, there's always a need to develop, test and validate said technology and its features. Often, executing these steps can be done manually or through an automation [1, 2]. These steps have to be executed as tasks in a checklist for any device before delivering its software to the customer [15]. The main goal of software testing is to identify errors, failures, gaps and missing features in regard to requirements and specifications [11].

Manual testing tends to not require knowledge of testing tools. However, it requires lots more effort and more human resources, i.e., more people. The work by Mohammad et al. shows that testers and developers may have to run different parts of programs with their own procedures and inputs in order to carry out the validations they need [14]. The execution of these manual procedures can be slow and may lead to successive errors.

On that regard, the work by Alferidah et al. shows the goal of automated testing is to reduce errors that may happen due to users' slowness and susceptibility to mistakes. Therefore, reducing errors leads to cost decrease in software [2]. Gamido and Gamido's work complements this by stating that, through the usage of automated testing, it is possible to repeat and reuse different test scenarios, which can be executed as often as needed [10]. Using the same logic, the work by Mohammad et al. states that automated testing tools are used to write scripts and record testing in much less time than manual testing, with the ability to repeat the tests as much as needed [14].

Automated testing involves the use of automation tools or frameworks in tests execution [17]. It requires knowledge of automation tools and, sometimes, programming skills, where developers write test code that runs without human intervention. With automation, there is an increase in tests accuracy, an increase in system behavior validation, and an increase in testers' effort and time savings compared to manual testing.

Our research identified previous studies related to development and use of automation tools in test processes for mobile devices. These studies discuss about time gains obtained through automation when compared with manual process.

In the work by Franca et al., the authors state that there are great volumes of update releases, new specifications and OS updates. Based on this, the authors created an experiment aiming to assist the development of technical documentation, using simulations in a controlled environment, where a new automated approach was used by a group of employees. The process automation showed a reduction of up to 33.2% in documentation's total time [9].

The work by Itkonen et al. shows how software development performed by teams is affected by factors as environment, human effort and integrated technologies. Their proposal was to do a survey, where the testers contributed with their experiences and feelings, labeled as "tester's experience" (TX) [12]. Questions regarding perspectives on testing review were asked, as well as infrastructure and aggregated effort as manpower and work quality. Based on the results, it was perceived that there are certain procedures only executed by testers with a certain maturity in testing. That is because performing some procedures can be difficult, dealing with some difficulties by complex level or many procedures to execute. However, regardless of testers' experience level, the commitment is the main factor for achieving success and completing testing tasks on time. This work serves as a complement to improve the test scenario, evaluating the use of testing artifacts and their associated activities by user perspective.

Additionally, the work by Andrade et al. shows that mobile manufacturers need to find strategies to reduce time of production, and, still, need to ensure the quality of the software embedded with stakeholders' requests [3]. The proposal of this work was to eliminate tests repeated for each stakeholder, choosing a set of key stakeholders and showing that gains obtained can be expanded to the whole device model portfolio. Even regarding external factors, different kinds of hardware and software, and also distinct quantity of requirements by stakeholder, the study concluded that 58% of comparisons presented time gains in terms of SW development, contributing for internal process improvement. Like our research goal, they aimed to decrease time spent on validations without the risk of a decrease in quality.

Brígido et al. developed the IMEIOFF Tool for automatic execution of integrity tests on mobile devices in order to reduce time on these test executions. They carried out a case study. Their results showed that their tool decreased the average execution time by up to 63% compared to manual execution [5]. Similarly, Chagas et al. developed an automated tool to perform sanity testing on

Android smartphones and tablets, set in a global software development context. Through observational studies, they were able to understand and showcase the impact of software automation usage as a testing strategy. They concluded in their analysis that there were time gains of up to 53% for testers, allowing the number of manual activities and the effort to be reduced [6].

Regarding our work, the scenario we have in our hands is True Single SKU (TSS), where devices have a common service provider, and, when these are connected through internet to the server, network specific features and services will be applied, in accordance to the region or country where each device was purchased [16]. Based on the TSS environment, automation tools represent a good solution when testers use it in concurrent activities. Costa et al. presented a script which automated the process of preparing TSS mobile device samples through a TSS Script. After carrying out some experiments, they registered a 46,67% time gain when preparing TSS samples with automations instead of using manual preparation [8].

After analyzing a work by Costa et al. [8], we noticed that we were using a similar TSS manual preparation process in our team, concentrating an expressive part of developers efforts. Our developers were continuously handling the devices one by one in TSS preparations, not being able to execute parallel tasks. Hence, our paper presents the development and implementation process of a TSS tool, which automates a group of manual procedures that were once decentralized, now setting TSS for one or more devices through automation and in parallel, along with a visual interface. Additionally, we discuss the effectiveness and gains obtained with our TSS tool when used in parallel with some mobile devices in the same scenario.

3 CURRENT SCENARIO AND OUR PROPOSAL

UI Requirements Team is responsible for developing visual requirements related to mobile device applications, ensuring each one is matching with customer specification through tests and software validations. To be able to do this, our developers handle mobile devices with customer full test binaries and set these to work in a TSS environment. Normally, the demands to develop and verify UI requiments in various TSS customer models is high due to many telephone carriers, which, in turn, are the manufacturer's customers. Associated to high demand, it was perceived a bottleneck in our process, specifically at the beginning of the chain, called TSS preparation steps on Device Under Test (DUT), where the procedures to follow are completely manual, as seen in Figure 2. In this flow, the execution time is slow with high risk of human error:

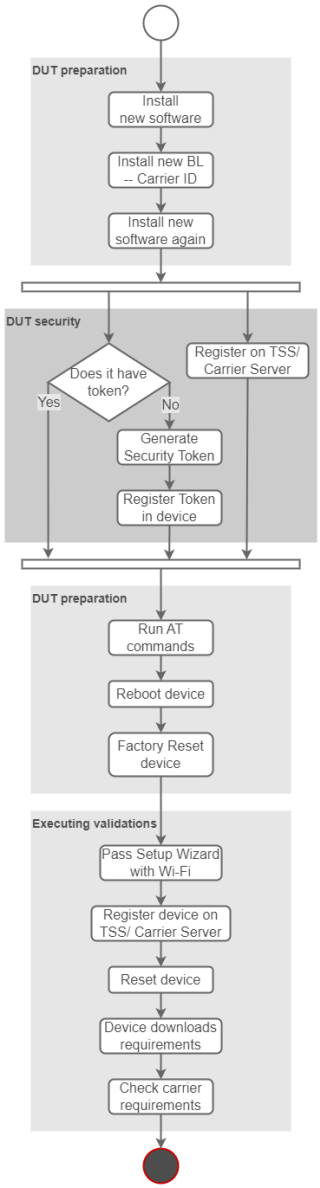

Figure 2: Manual process to set TSS, now in greater details.

(1) **Install new software:** The developer installs new customer full software (full binary) on the DUT.
(2) **Install new BL Carrier ID:** The developer installs boot loader (BL) software, that was generated through full binary and that contains base carrier identification (ID for carrier's country).
(3) **Install new software again:** When BL is installed, DUT is available, but not working to develop and test requirements. So, the developer installs full binary again.
(4) **DUT security - Token:** The developer checks if device has some permissions, called tokens, to be able to receive AT command and reset the device. If some token is needed, the developer requests to generate it and registers it in the device.
(5) **DUT security - TSS Carrier Server:** The developer registers in customer server the final carrier identification to be associated with a device.
(6) **Run AT commands:** The developer executes AT commands on the device using a specific console to clean it, keeping only the base carrier ID.
(7) **Factory Reset Device:** The developer restarts the device.
(8) **Pass Setup Wizard with Wi-Fi:** The developer executes steps from Setup Wizard and turns on Wi-Fi so that it establishes Carrier Server communication.

(9) **Register device on TSS/ Carrier Server:** The developer confirms the association of final carrier identification from carrier server.
(10) **Reset device:** The developer restarts the device.
(11) **Device downloads requirements:** After restarting the device, the developer executes steps from Setup Wizard until customer requirements are fully downloaded.
(12) **Check carrier requirements:** The developer executes our internal Model Tool test and validates carrier requirements on the device. The Model tool is our automated process for UI requirements validation, and, regarding the current process, this part is out of manual procedure and will not be covered nor detailed in this paper.

TSS manual device preparation causes execution delays because the procedures are decentralized, using parts of different programs and being handled manually by developers. This statement was also verified in the TSS Script paper [8]. Since we know that an extensive part of our development projects belongs to mobile models with TSS SKUs, as seen in Table 1, it was necessary to stop and invest time to create our own TSS Tool in order to execute all of these preparation decentralized steps in a single environment. We integrated this TSS Tool as an initial extension of our Model Tool.

Table 1: Number of TSS and Non-TSS Models.

Mobile Device Type	TSS	Non-TSS
Smartphone	90	57
Tablet	0	28

In this new scenario, which is represented by Figure 3, the developer continues flashing software binaries in the samples, but now they can use the TSS Tool. With the TSS Tool, the developer can visualize the status of each device and take the best decision to continue TSS process preparation:

(1) **TSS Tool configuration - Token**: The tool shows all permission tokens installed. If another permission is still needed, the developer must request it.
(2) **TSS Tool configuration - Carrier Server and AT commands:** Now, it is possible to choose more than one final carrier ID in samples being prepared simultaneously, as we can see in Figure 4. This means that executing AT commands, rebooting the device, installing some properties with final carrier designation, and replacing the use of carrier server are activities that can now be executed in an automated way, reducing preparation time between TSS model samples.

Keep in mind that developers can decide if they want to apply an action for one or more devices, i.e., parallelism. We explain the proposed solution in technological terms below:

- **Service**: The Service connects the devices by Android Debug Bridge (ADB) commands to get the Serial Number (SN) from each sample, and then sends these SNs to the Server. Additionally, it is used to start requests, i.e., actions, from the developer to go to the Server. To avoid conflicts of multiple devices, threads queuing is used to be processed in parallel. The Service uses Python, Flask and pure-python-adb, i.e., a library, as technology.

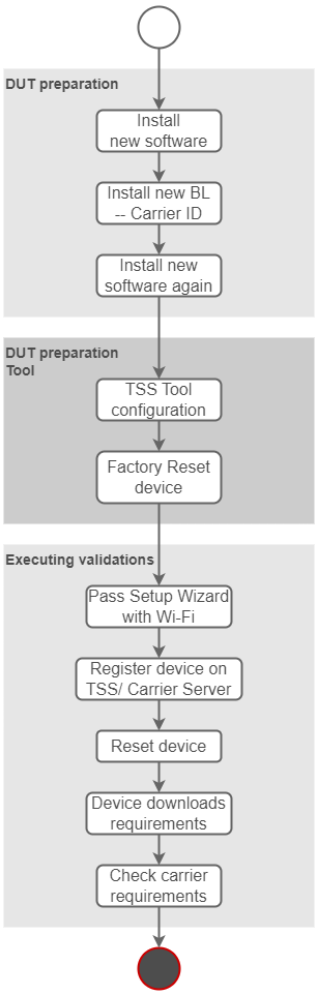

Figure 3: New automated process to set TSS.

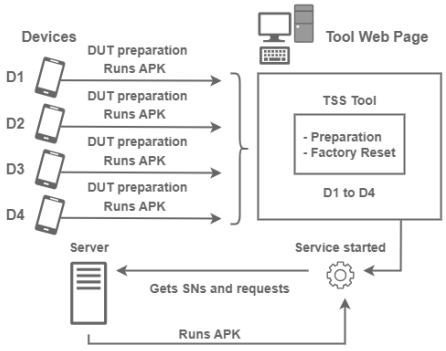

Figure 4: DUT preparation parallelism.

- **Instrumentation Tests APK - Requirements Test**: Android Application Package (APK) is an Android native application called by the Service to be executed under one or more devices. It simulates human actions on the interface. Here

it executes TSS preparation steps, including AT commands, automatically. APK uses Java and UiObject2 as technologies.

- **Server**: The Server manages the requests from the Service and re-distributes them among devices with correct TSS application, signalizing the Service to call APK to run by device. The Server uses Python and Django REST Framework as technologies.
- **Web App - Tool Web Page**: The web app shows visual portions, with screens containing visual information about connected devices, APK output after AT command execution, and available actions, i.e., future requests, to be applied for each connected TSS model sample. The web app uses JavaScript and Next.js as technologies.

Table 2 shows the main differences of the TSS Script in contrast with our proposed TSS Tool. The main advantage of the TSS Tool compared with TSS Script is related to the visual status to take decision, as well as the parallel execution with more than one device, which helps operationalize the TSS process, which is now performed automatically using the TSS Tool.

Table 2: Comparison of TSS Tool with TSS Script.

Feature	TSS Tool	TSS Script
Visual Status to take decision	YES	NO
Automated execution	FULL	PARTIAL
Parallel execution	YES	NO

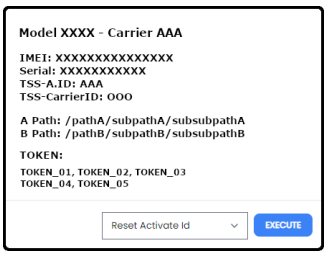

Figure 5: A screen capture concept of our TSS Tool's "Reset Activate ID" feature.

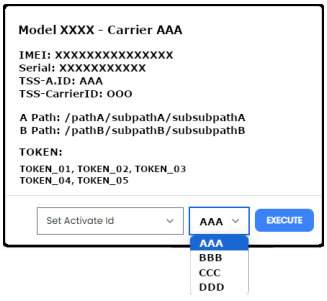

Figure 6: A screen capture concept of our TSS Tool's "Set Activate ID" feature.

The visual status of available carrier options is an important feature to guide the developer in a specific part of the TSS setup process. So, we implemented some screens to better display the available options and current settings. Figure 5 and Figure 6 represent specific features where the developer can choose one or more devices to change the device setup in order to continue the TSS setup process. These images had to be slightly edited to avoid sharing confidential information.

4 METHODOLOGY

This section presents the methodology we used to obtain the results from our research and the practical applications of the tool we proposed. Each step for this activity is described below:

(1) **Literature review:** Through this stage, we surveyed related works, which presented different tools created for automating processes and for improving solutions within the software testing context.
(2) **TSS Tool development:** A tool we developed for automating the process of multiple devices preparation for the TSS environment. It reduced the amount of repetitive manual tasks executed by developers. We implemented this tool using Python programming language along with AT commands. This tool is complementary to our current Model Tool, which was developed before in our team to develop, test and validate certain requirements.
(3) **TSS Tool validation:** This step consisted of the planning and execution of the following tasks:
   - **Selection of participants and TSS Tool introduction:** We selected four developers who already work with TSS product models daily, having executed at least once this manual process in their normal activities. Then, we trained them to use the TSS Tool for sample preparation in multiple devices simultaneously. After their training, they were instructed to use the TSS Tool in some tasks of selected models as part of this validation experiment. As the experiment was executed, we recorded the times of both manual and automated preparations, comparing the difference between them in a controlled scenario, without affecting their normal activities.
   - **Interviews:** We conducted the interviews during the TSS Tool trials, face to face with our subjects. We took notes as the interviews went on and recorded our insights in a notebook. The participants were free to express their opinions about using both manual and automated processes in the experiment, reporting the advantages and disadvantages of using the tool regarding time and allocation of multiples devices in parallel. It is important to emphasize the anonymity of the opinions we collected, preserving the authenticity of the answers and satisfaction level of the user during device setup using TSS tool.
   - **Observation study:** A study was performed related to TSS device setup preparation. It was conducted with four developers as subjects, where we recorded execution times of both manual and automated processes for each developer in separate sessions.

- **Analysis of the results:** The data collected through the interviews and the observation study with participants were summarized and analyzed. The context of such data is the execution time recorded from manual and automatic processes accomplished for all the members of the comparative experiment.

4.1 Experiments

Our goal with this experiment is to compare the manual and automatic TSS setup executions, which we believe will show the effectiveness of the TSS Tool regarding time decrement, mainly for parallel executions with multiple devices.

To collect the data of this study, four participants were selected. The first set of subjects was composed of two experienced developers, with more than 12 months of experience; and the second set of subjects was composed of two beginner developers, with less than 6 months of experience in requirements development for TSS devices, which can be seen in Table 3. These developers were trained on how to use the TSS tool, and, during the experiment, they answered some questions to identify and confirm the main issues found in manual process and the advantages and challenges perceived in the new automated process.

Table 3: Characterization of participants.

Participant Category	Experience Time
Beginner	1 to 6 months
Experienced	more than 12 moths

The experiment involved 18 distinct mobile devices, i.e., smartphones, which were selected from different kinds of hardware (HW) and software (SW) versions. The experiment was executed in three rounds with three randomly selected device samples. Since the experiment consists of a comparison between manual and automatic processes, this means we had to use the same round samples to execute both processes with the different approaches, i.e., manual and automatic. So, we calculated the total and average times for each round per approach. We chose average time to be reported because the execution time can vary due to devices SW and HW versions, as well as external factors that can influence mainly the manual process.

Regarding the manual process, the developers received 18 devices, where in each round each participant got three devices and the carrier code list to work with in the TSS manual setup. They were instructed to use only one USB port to execute the whole TSS setup process per device. We timed the experiment rounds, and also observed and wrote down the main external factors that impacted the device TSS preparation, e.g., if it was necessary to repeat some operations more than once or execute some other procedures that were not checked before.

External factors can mean human error or SW instability in some steps, as described below:

- **Steps before TSS preparation:** Some conditions are important to check in the devices before starting TSS setup process. Installation of permissions, i.e., tokens, and enabled USB debug are needed to use AT commands. There's also registration in the carrier server to set the final carrier code, but sometimes some kind of instability happens in the carrier server, being necessary to refresh the server page to continue. For developers, mainly the beginners, the lack of these manual checks in the appropriated time were observed during the rounds, increasing the manual execution time.
- **Executing TSS preparation:** A script with AT commands is used in a program, step by step, and executed manually, where each command awaits a valid answer. When this answer is blank or returns something invalid, the developer must repeat the command until they get a satisfactory result. This kind of failure was observed in almost all rounds for experienced and beginner developers.

Regarding the automated process, the developers used the same 18 devices distributed in three devices per round too. They were also given a carrier code list, a USB hub to support three device connections at the same time, and the TSS Tool to perform TSS preparation in multiple devices in parallel. During the experiment, we timed the rounds and observed if there was any external factor like the ones found previously. The significant factors transformed positively through the TSS Tool. Besides being able to work with three devices in parallel, there were these other factors: 1) Visual signaling of USB port and token status of each device sample connected in USB hub, since the beginning; 2) Replacing carrier server to register final carrier ID by equivalent automated command; 3) AT commands were programmed to execute automatically, without human interaction, and repeating steps in a high speed almost not perceived by the user, becoming more stable too.

Finally, we were able to collect enough information to create summarized tables with Total and Average Times for each round, analyzing and presenting the results of this study. Additionally, we collected personal opinions from our participants with the questions displayed in Table 4.

Table 4: Questions

Questions	
Q1	What were the improvements in terms of test activity that you got regarding the automated process?
Q2	What benefits did you find in our proposal?
Q3	Was there any difficulty or limitation not yet solved?

5 RESULTS AND DISCUSSION

Regarding recorded times, a comparison was made between Total and Average execution times of TSS device preparation performed manually (three devices in sequence) and automatically (three devices in parallel). Table 5 summarizes the results obtained for both categories of participants, where 52% (1h01m36s) is for beginner testers and 50% (53m07s) is for experienced testers.

Analyzing the results presented in Tables 6 and 7, it is possible to identify an expressive total time gain on the automated process, where the execution time is accumulated. When the setting is automated, there is almost no difference between types of participants, because the process assumes constant and progressive steps, without external factors to interfere with the result. Thinking in high

Table 5: Complete Rounds

Complete Rounds					
Participant	Manual		Automated		
	Total Time (TT)	Average Time (AT)	Total Time (TT)	Average Time (AT)	TT Gain
Beginner	1:58:26s	06m35s	56m50s	03m09s	52%
Experienced	1:46:13s	05m54s	53m06s	02m57s	50%

demand tides, the time gain between total and average times will be even more evident using the automated and parallel process. Still, even considering average time and a low demand scenario, which is how we executed this experiment, the manual process with devices one by one is more time consuming.

Table 6: Manual Setting Experiment

Manual Setting with 3 samples per round					
Participant	R1	R2	R3	Total Time	Average Time
Beginner1	19m48s	19m12s	18m59s	57m59s	06m27s
Beginner2	19m10s	20m45s	20m32s	1h0m27s	06m43s
Experienc.1	18m10s	18m11s	18m08s	54m29s	06m03s
Experienc.2	17m32s	17m10s	17m02s	51m44s	05m45s
All participants:				3:44:39	24m58s

Table 7: Automated Setting Experiment

Automated Setting with 3 samples per round					
Participant	R1	R2	R3	Total Time	Average Time
Beginner1	09m47s	09m47s	09m40s	29m14s	03m15s
Beginner2	09m02s	09m20s	09m14s	27m36s	03m4s
Experienc.1	09m10s	08m54s	08m47s	26m51s	02m59s
Experienc.2	09m05s	08m38s	08m32s	26m15s	02m55s
All participants:				1:49:56	12m13s

When the participants were asked the questions in Table 4, the opinions were almost all positive too. They were summarized with the following points without identifying the participant:

- **Q1:**
  - Now, we are able to prepare different TSS samples simultaneously and faster;
  - While TSS samples are being prepared, in parallel, we are able to continue the next requirements validation in other TSS samples that were already prepared;
  - Since we can follow visually what is happening, we can connect a new TSS sample to be prepared if some other sample finishes as soon as possible, without leaving any sample in stand-by for a long time.
- **Q2:**
  - In high demand tides, we can deliver more TSS samples prepared for requirements development and validation in less time.
- **Q3:**
  - It is important that we mention that we test different kinds of models with multiple embedded hardware and software versions, which may present some rare behaviors that can cause some difficulties as perceived by developers.
  - Some device models are processed faster than others. And for slower samples, the visual answers are also slower.

Through the subjects' answers, we identified that they noticed a time gain on TSS preparation due to the tool instance running simultaneously. Although, it is important to emphasize that, in some rare moments, the TSS tool didn't display visually some results in one or other device in some rounds, not demonstrating to the user, by visual perception, the need to restart the action. This happens because the proposed solution has a communication limit between the Service and the Server with the APK. When the APK is ordered to run the TSS steps, the answers appear only on web-app side, without an invalid return associated with APK performance as there is no answer from the device to show.

6 CONCLUSION AND FUTURE WORK

In this work, the use of a TSS Tool with multiple devices in parallel was proposed to replace the manual TSS samples preparation, since almost all models we work with in this institute have the TSS feature. We conducted an experiment with different participant categories, i.e., beginners and experienced. We also interviewed our subjects to gather their reports about improvements, benefits and limitations that came along with the new automated process, which used our TSS Tool. This also encompassed practical observation along the interviews with our four subjects. The results were summarized in tables with accumulative and average execution times, categorized in TSS manual preparation and TSS automatic processes. A final table was filled with a gain time of 52% for beginner testers and 50% for experienced testers, demonstrating that the new process is effective and that it diminishes common errors executed in the old manual process. Furthermore, more TSS devices are set to follow to the next requirement validation process. Therefore, the TSS Tool becomes a strong ally when high demand tides come, because more devices are set in parallel, decreasing the wait moment between TSS candidate devices to be prepared and validated in the Model Tool, which is another tool we use to help in tests automation.

As future work, improvements can be made in the TSS tool, including more steps after TSS preparation, integrating setup wizard automation in multiple devices before requirements embedding and validation, and reporting problems in execution when a visual answer delays.

Since the quantity of TSS models is big, varying in different hardware and software versions, we verified that, even when working with an automated software process with devices in parallel, there was evidence of different times of TSS preparation. With this, we identified the opportunity to store information to build a machine learning model, using information such as models, final carriers, and preparation times to suggest the best model sets to run TSS preparation, in order of execution, to save time in high demand seasons, and deliver ready TSS devices on time to develop and validate UI requirements in the next phase.

Additionally, the TSS Tool can be integrated to a remote server, where candidate model devices not physically available in the institute can be configured and receive an appropriate TSS preparation. This kind of integration is important because, sometimes, in an initial new project moment, there are some model samples allocated only in another country, and work in TSS preparation in advance would anticipate important tasks in a research project.

Currently, the Technical Manager (TAM) or Quality Assurance Project Leader (QA PL) send a previous e-mail requiring TSS configuration, in TSS carrier server, for one or more field samples already available in the market, creating a dependency of product configuration with our team when some punctual studies or qualitative analysis is required. Based on this scenario, integrating the TSS Tool in a remote server with some samples to perform TSS preparation automatically can be the right solution to speed up some actions without our consent.

ACKNOWLEDGMENTS

This work is the result of the RD&I Project ASTRO, carried out by SIDIA R&D Institute, in partnership with Samsung Eletrônica da Amazônia Ltda., using resources from Brazilian Federal Law 8.387/1991, in accordance with the provisions of article 39 of Decree 10.521/2020.

REFERENCES

[1] Rabiya Abbas, Zainab Sultan, and Shahid Nazir Bhatti. 2017. Comparative analysis of automated load testing tools: Apache jmeter, microsoft visual studio (tfs), loadrunner, siege. In *2017 international conference on communication technologies (comtech)*. IEEE, 39–44.

[2] Saja Khalid Alferidah and Shakeel Ahmed. 2020. Automated software testing tools. In *2020 International Conference on Computing and Information Technology (ICCIT-1441)*. IEEE, 1–4.

[3] Paulo Andrade, Kathrian Marques, Luiz Ribeiro, Marcelo da Silva, Hendria Fragata, Adriano Oliveira, and Juan Nogueira. 2023. Test volume mitigation for mobile devices software development: An improvement approach considering shared requirements. In *Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing*. 1–4.

[4] Mário Barros and Eric Dimla. 2021. From planned obsolescence to the circular economy in the smartphone industry: An evolution of strategies embodied in product features. *Proceedings of the Design Society* 1 (2021), 1607–1616.

[5] Antonio Brígido, Camilo Souza, Fedrik Moura, Marcelo Reis, Andre Neto, and Bruno Bonifácio. 2021. An Industrial Case Study on Applying Software Testing Automated in Global Software Development Environment. In *23rd International Conference on Global Software Engineering and Technology on January, 28-29, 2021 at New York, USA (2021)*. 1456–1459.

[6] Ana Carolina Chagas, Davi Gonzaga, Leonardo Albuquerque, Flavia Oliveira, Renata Castro, and Lennon Chaves. 2023. BSA Tool: An Experience Report of Software Automation to Perform Sanity Tests in a Global Software Development Environment. In *2023 3rd International Conference on Information Communication and Software Engineering (ICICSE)*. IEEE, 15–20.

[7] Wellington Correa, Carol Fernandes, Victor Medeiros, Marcelo Diez, Paulo Lopes, Bárbara Santos, and Paulo Andrade. 2024. X9: Tool for Tracking Mobile Devices Among Teams. In *2024 IEEE 14th Symposium on Computer Applications & Industrial Electronics (ISCAIE)*. IEEE, 83–88.

[8] Antônio B da Costa, Maria Meireles, Francisco Caio de Barros, Gaspar Mota, Lennon Chaves, and Lidia Roque. 2022. TSS Script: Automation Tool Applied in the Preparation of True Single SKU Testing Environment. In *2022 IEEE 2nd International Conference on Information Communication and Software Engineering (ICICSE)*. IEEE, 231–231.

[9] André José De Franca, Gaspar Henrique Alver Mota, Klirssia Isaac Sahdo, Leonardo Tiago, Flávia Oliveira, and Lennon Chaves. 2023. LinkDoc: An Automated Process in the Delivery of Documentation in a Global Software Development Environment. In *2023 The 5th World Symposium on Software Engineering (WSSE)*. 9–16.

[10] Heidilyn Veloso Gamido and Marlon Viray Gamido. 2019. Comparative review of the features of automated software testing tools. *International Journal of Electrical and Computer Engineering* 9, 5 (2019), 4473.

[11] Hussam Hourani, Ahmad Hammad, and Mohammad Lafi. 2019. The impact of artificial intelligence on software testing. In *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*. IEEE, 565–570.

[12] Juha Itkonen, Mika V Mantyla, and Casper Lassenius. 2009. How do testers do it? An exploratory study on manual testing practices. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, 494–497.

[13] Volker G Kuppelwieser, Phil Klaus, Aikaterini Manthiou, and Othman Boujena. 2019. Consumer responses to planned obsolescence. *Journal of Retailing and Consumer Services* 47 (2019), 157–165.

[14] Duaa R Mohammad, Sajedah Al-Momani, Yahya M Tashtoush, and Mohammad Alsmirat. 2019. A comparative analysis of quality assurance automated testing tools for windows mobile applications. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 0414–0419.

[15] Chadatarn Raksawat and Pattama Charoenporn. 2021. Software testing system development based on ISO 29119. *Journal of Advances in Information Technology* 12, 2 (2021).

[16] Samsung. 2024. What is True Single SKU? https://www.samsung.com/uk/support/mobile-devices/true-single-sku/. Accessed: 2024-01-26.

[17] Mubarak Albarka Umar and Chen Zhanfang. 2019. A study of automated software testing: Automation tools and frameworks. *International Journal of Computer Science Engineering (IJCSE)* 6 (2019), 217–225.