Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2020, 7(7):66-72

Research Article ISSN: 2394 - 658X

Unique Challenges to Test and Automate Mobile Device Management Application on Multiple Devices

Amit Gupta

San Jose, CA
*gupta25@gmail.com

ABSTRACT
Mobile Device Management (MDM) suites, such as VMware Airwatch, MobileIron, and SOTI, are essential for managing and securing mobile devices in enterprises. Testing and automating these suites across multiple devices presents unique challenges due to the diverse range of hardware, operating systems, and configurations. This paper explores these challenges in detail, focusing on issues such as automating device unlocking after reboot, maintaining active sessions post-reboot, and addressing network and connectivity problems. Furthermore, it proposes a comprehensive solution using an advanced automation agent designed to interface seamlessly with MDM suites. The paper also includes detailed architectural designs, case studies, and future research directions to enhance the reliability and efficiency of MDM testing and automation processes.

Key words: Test Automation, Mobile Application, Mobile Test Automation, Android, Mobile Device management (MDM), Unified Endpoint Management (UEM), iOS, Software testing, Mobile Application Testing

INTRODUCTION

With the proliferation of mobile devices in corporate environments, MDM solutions have become crucial for ensuring security, compliance, and efficiency. The widespread use of these devices necessitates robust management solutions to protect sensitive data, enforce security policies, and streamline operations. However, testing and automating these solutions across a wide array of devices and operating systems pose significant challenges. These challenges include device diversity, varying operating system behaviors, security restrictions, and network connectivity issues. This paper identifies and explores these key obstacles in depth and proposes innovative solutions to enhance the automation and reliability of MDM testing. By addressing these challenges, the proposed solutions aim to improve the efficiency and effectiveness of MDM implementations, ensuring that they can meet the complex demands of modern enterprise environments.

CHALLENGES TO AUTOMATE MDM

A leading Mobile Device Management Solution (Unified Endpoint Management), is a comprehensive enterprise mobility management (EMM) solution designed to manage and secure mobile devices, applications, and content across various platforms. Here are few primary features which are challenging to automate from test validation prospective:

1. Device Management: MDM Solution enables organizations to manage a diverse range of devices, including smartphones, tablets, laptops, and IoT devices, across different operating systems such as iOS, Android, Windows, macOS, and others.
2. Mobile Application Management (MAM): It allows administrators to manage and distribute mobile applications securely to end-users' devices, including in-house developed apps and those obtained from public app stores.

3. Mobile Content Management (MCM): MDM Solution provides secure access to corporate documents and content on mobile devices, allowing organizations to enforce policies such as encryption, access control, and remote wipe capabilities to protect sensitive data.

Along with these MDM challenges, Test automation engineers are also challenged to solve operational automation problems such as

1. Device Diversity

Mobile devices vary in terms of hardware, operating systems, and configurations. This diversity makes it difficult to create a one-size-fits-all automation framework. Testing needs to cover a wide range of devices to ensure compatibility and performance across all potential use cases.

2. Device Unlocking After Reboot

Automating the unlocking of devices after a reboot is a critical challenge. Devices often require manual intervention to enter passwords, PINs, or biometric data, which impedes automated testing. Each operating system and device manufacturer might have different mechanisms and security policies for device unlocking, complicating the automation process.

3. Session Maintenance Post-Reboot

Maintaining an active session post-reboot is essential for seamless automation. However, reboots often disrupt sessions, causing tests to fail or necessitate manual reconfiguration. Ensuring session persistence requires sophisticated handling of session tokens and automated re-authentication mechanisms.

4. Network and Connectivity Issues

Fluctuations in network connectivity can interrupt automation processes, especially during remote testing of devices. MDM functionalities heavily depend on stable network connections for policy updates, compliance checks, and data synchronization. Testing automation must account for variable network conditions and include robust retry mechanisms.

5. Security Restrictions

MDM solutions enforce strict security policies that can hinder automation, such as preventing automated tools from accessing certain device features. Overcoming these restrictions without compromising security or violating policy compliance is a significant challenge.

MDM FEATURE DETAILS

Policies

1. Device Security Policies:

- Passcode Policy: Administrators can enforce passcode complexity, length, and expiration requirements to ensure device security.
- Encryption Policy: Require device encryption to protect data at rest.
- Network Security: Enforce VPN usage, Wi-Fi configurations, and firewall settings to secure network communications.
- App Security Policies: Define restrictions on app installations, including blacklisting or whitelisting specific apps based on security or compliance requirements.
- Jailbreak/Root Detection: Implement policies to detect and prevent jailbroken or rooted devices from accessing corporate resources.

2. Application Management Policies:

- App Distribution: Control the deployment of enterprise apps through various methods such as public app stores, enterprise app catalogs, or direct installation.
- App Configuration: Configure app settings and policies remotely, such as restricting copy-paste functionality or enabling app-specific VPN tunnels.
- App Containerization: Apply containerization techniques to isolate corporate apps and data from personal ones, ensuring data privacy and security.

3. Content Management Policies:

- Content Encryption: Enforce encryption for corporate documents stored on devices to prevent unauthorized access.

- Content Distribution: Control access to and distribution of corporate content, including documents, presentations, and media files.
- Content Collaboration: Facilitate secure collaboration by enabling features like document sharing and editing within secure containers.

Device Enrollment

1. User Enrollment:
   - Offer self-service enrollment options for users to onboard their devices securely.
   - Guide users through the enrollment process, including installing the AirWatch agent and configuring device settings.
   - Authenticate users during enrollment using single sign-on (SSO) or multi-factor authentication (MFA) for added security.
2. Device Registration and Profile Assignment:
   - Register devices with the AirWatch console to establish a management connection.
   - Assign device profiles containing security policies, configurations, and applications based on user roles or device attributes.
   - Ensure seamless integration with existing directory services for user and device authentication.
3. Post-Enrollment Management:
   - Monitor enrolled devices for compliance with configured policies.
   - Provide ongoing support and troubleshooting for enrolled devices, including remote assistance and issue resolution.
   - Decommission devices securely when they reach the end of their lifecycle or are no longer in use.

By focusing on these aspects in greater detail, organizations can establish robust mobile device management practices using VMware AirWatch, ensuring both security and efficiency in their mobile workforce operations.

PROPOSED SOLUTIONS

Testing passcode policy enforcement on mobile devices involves verifying various parameters such as complexity, length, expiration, and lockout behavior. Automating this process requires a combination of API interactions, device interactions, and validation of device settings. Here's how you can approach this:

USE CASE

Let's consider a scenario where an organization enforces passcode policies on mobile devices using Web Console or REST APIs. The objective is to automate the verification of passcode parameters and ensure that the policies are correctly applied during device enrollment and policy updates.

AUTOMATED TESTING APPROACH

- Automate Enrollment: Utilize automation scripts to simulate the device enrollment process.
- Policy Verification: After enrollment, fetch device settings using MDM APIs to verify that the passcode policy is correctly applied.

Test Scenarios

- Verify passcode length and complexity requirements.
- Validate expiration settings to ensure passcodes expire after the specified duration.
- Trigger lockout conditions by entering incorrect passcodes multiple times and verify the device behavior.

```
import XCTest

class DeviceEnrollmentTests: XCTestCase {

 func testDeviceEnrollment() {
 // Simulate device enrollment process
 enrollDevice()

 // Fetch device settings after enrollment
 let deviceSettings = fetchDeviceSettings()

 // Verify passcode policy
 verifyPasscodePolicy(deviceSettings: deviceSettings)
 }

 func enrollDevice() {
 // Simulate device enrollment process (Using MDM REST APIs)
 print("Simulating device enrollment...")
 // Perform device enrollment steps here
 }

 func fetchDeviceSettings() -> [String: Any] {
 // Simulate fetching device settings using MDM APIs print("Fetching device settings...")
 let deviceSettings: [String: Any] = [
 "passcodeLength": 6,
 "passcodeComplexity": "Alphanumeric",
 "passcodeExpiration": "90 days",
 "passcodeLockout": true,
 // Add more device settings as needed
 ]
 return deviceSettings
 }

 func verifyPasscodePolicy(deviceSettings: [String: Any]) {
 // Test scenarios for passcode policy verification
 XCTAssertTrue(deviceSettings["passcodeLength"] as! Int == 6, "Passcode length should be 6 digits")
 XCTAssertTrue(deviceSettings["passcodeComplexity"] as! String == "Alphanumeric", "Passcode complexity
should be alphanumeric")
 XCTAssertTrue(deviceSettings["passcodeExpiration"] as! String == "90 days", "Passcode expiration should be
90 days")

 // Trigger lockout conditions by entering incorrect passcodes multiple times
 let maxAttempts = 5
 for _ in 1...maxAttempts {
 // Simulate entering incorrect passcodes
 enterIncorrectPasscode()
 }
```

One of the key here is test automation architecture to achieve MDM based application test automation.
When we are dealing with device lock and passcode, policy enforcement, we need an external system which drives the device under test. Below is the high level of architecture to achieve this.

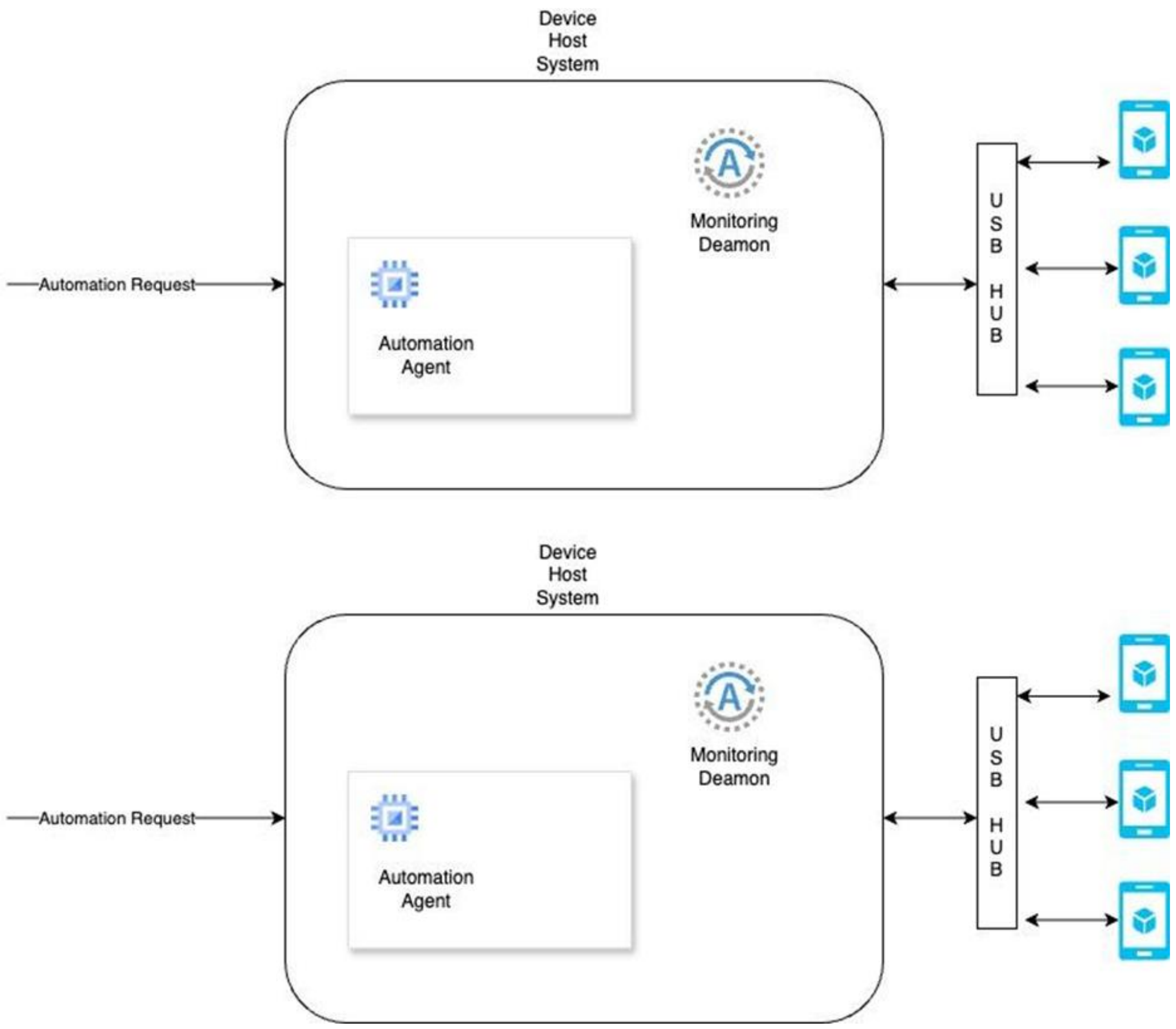

Figure 1: Architecture illustration of Automation Framework Setup

Figure 1 architecture illustrates a Device Host System responsible for handling automation requests and managing connected devices through a USB Hub. Here's a breakdown of the components and their roles

1. Device Host System:

Automation Agent: The primary component responsible for executing automation tasks on connected devices.

- Receives automation requests from a CI (Continuous Integration) system.
- Sends key commands to connected devices.
- Reboots devices while maintaining the automation session.
- Sends device passcodes to lock/unlock devices.
- Erases and restores connected devices.
- Pull the test result file from device
- Maintains sessions with connected devices even after reboots.

Monitoring Daemon: Ensures the stability and reliability of the Device Host System by monitoring all critical services.

- Monitors the status of all services running on the Device Host System.
- Restarts any service that goes down.

2. USB Hub

- Connects multiple real devices to the Device Host System.
- Enables communication between the Automation Agent and the devices.

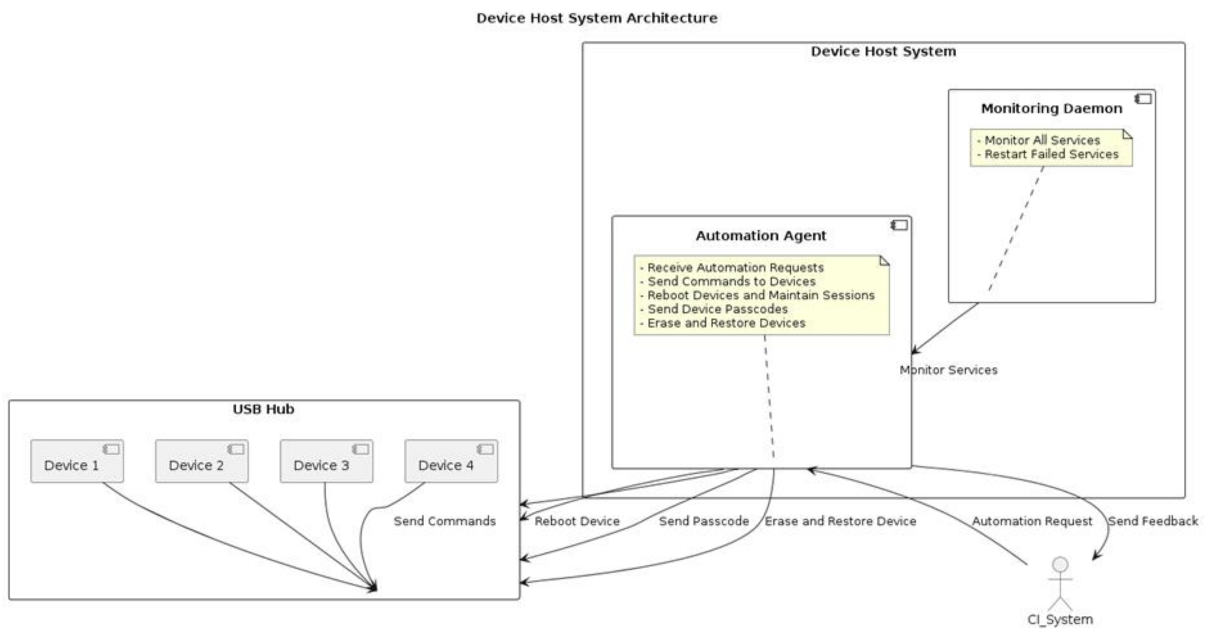

Figure 2: Data Flow for Device Host System

Figure 2 illustrates how data flows between a Device Host System, Monitoring Daemon and Connected devices.

FUTURE WORK

Future research can focus on refining the automation agent, exploring AI-driven automation strategies, and enhancing security measures to ensure robust and reliable MDM management.

Advanced AI Techniques

The integration of AI and machine learning techniques can further enhance the capabilities of the automation agent. AI-driven automation can predict and adapt to device behaviors, improving the reliability and efficiency of automated testing for MDM enabled applications.

Extended Device Coverage

Expanding the automation framework to cover new devices and operating systems as they are released will ensure that the MDM solution remains up-to-date and effective. This includes using Device farm or dedicated device lab and can provide support by integrating for emerging technologies such as new OS versions, IoT devices and wearables.

IoT Integration

As IoT devices become more prevalent in enterprise environments, integrating these devices into the MDM solution will be crucial. Automation strategies need to be developed to manage the unique challenges posed by IoT devices, such as limited computing resources and varied communication protocols.

Wearable Devices

Wearable devices, such as smartwatches and fitness trackers, are increasingly used in corporate settings. The automation framework should be extended to include these devices, ensuring comprehensive management and security.

CONCLUSION

Testing and automating MDM suites across multiple devices present unique challenges, but with the right strategies and tools, these can be effectively managed. The diverse range of devices, varying operating system behaviors, and stringent security policies all contribute to the complexity of this task. However, an automation agent offers a robust solution for overcoming these hurdles by automating critical processes such as device unlocking after reboot, maintaining active sessions post-reboot, and handling network issues that can disrupt automated testing. Implementing such an agent ensures that these tasks are performed consistently and accurately, reducing the need for manual intervention and minimizing errors. Moreover, integrating this agent with a cross-platform framework and continuous integration/continuous testing (CI/CT) pipelines can significantly enhance the efficiency and reliability of MDM testing. This integration allows for automated tests to be run seamlessly across various devices and configurations, ensuring comprehensive coverage and early detection of issues. Overall, the strategic use of an automation agent combined with advanced testing frameworks and methodologies can lead to more reliable and effective MDM solutions, ultimately supporting better security, compliance, and operational efficiency in enterprise environments.

REFERENCES

[1]. B. Kirubakaran and V. Karthikeyani, "Mobile application testing — Challenges and solution approach through automation," 2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering, Salem, India, 2013, pp. 79-84, doi: 10.1109/ICPRIME.2013.6496451.

[2]. A. Pandey, R. Khan and A. K. Srivastava, "Challenges in Automation of Test Cases for Mobile Payment Apps," 2018 4th International Conference on Computational Intelligence & Communication Technology (CICT), Ghaziabad, India, 2018, pp. 1-4, doi: 10.1109/CIACT.2018.8480303.

[3]. Muhammad Mudassar Yamin and Basel Katt. 2019. Mobile device management (MDM) technologies, issues and challenges. In Proceedings of the 3rd International Conference on Cryptography, Security and Privacy (ICCSP '19). Association for Computing Machinery, New York, NY, USA, 143–147. https://doi.org/10.1145/3309074.3309103

[4]. Majdi, Elmira Bagheri, et al. "Evaluation of Mobile Device Management tools and analyzing integration models for mobility enterprise." *Umee University* (2013).

[5]. Alotaibi, Ashwaq A., and Rizwan J. Qureshi. "Novel framework for automation testing of mobile applications using Appium." *International Journal of Modern Education and Computer Science* 9.2 (2017): 34.

[6]. Tobias Griebe and Volker Gruhn. 2014. A model-based approach to test automation for context-aware mobile applications. In Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC '14). Association for Computing Machinery, New York, NY, USA, 420–427. https://doi.org/10.1145/2554850.2554942

[7]. J. Gao, W. -T. Tsai, R. Paul, X. Bai and T. Uehara, "Mobile Testing-as-a-Service (MTaaS) -- Infrastructures, Issues, Solutions and Needs," 2014 IEEE 15th International Symposium on High-Assurance Systems Engineering, Miami Beach, FL, USA, 2014, pp. 158-167, doi: 10.1109/HASE.2014.30.

[8]. VMware Airwatch: https://www.vmware.com/products/workspace-one.html

[9]. MobileIron MDM Solutions: https://www.mobileiron.com/

[10]. SOTI MDM Suite: https://www.soti.net/

[11]. Appium: Cross-Platform Mobile Testing: https://appium.io/