Rancang Bangun *Automation Test Journey* pada *E-Commerce*

(Studi Kasus: *Marketplace* PT. Tokopedia)

Faiq Dhimas Wicaksono
Program Studi Informatika – Program Sarjana
Universitas Islam Indonesia
Yogyakarta, Indonesia
18523088@students.uii.ac.id

Septia Rani
Jurusan Informatika
Universitas Islam Indonesia
Yogyakarta, Indonesia
septia.rani@uii.ac.id

***Abstract*—Pengujian perangkat lunak adalah serangkaian kegiatan yang dilakukan untuk mencari cacat atau kejanggalan pada aplikasi serta memastikan aplikasi sudah berjalan dengan baik sesuai persyaratan yang diharapkan. Proses ini dahulu sering dilakukan secara manual tetapi saat ini sebagian besar sudah beralih ke otomatis karena pengujian manual sangat memakan waktu dan biaya sehingga rawan terjadi kesalahan (*human error*). Pengujian otomatis sendiri saat ini semakin dibutuhkan oleh semua perusahaan pengembang perangkat lunak karena sifatnya yang otomatis sehingga dapat dilakukan kapan saja secara berulang dan mengurangi upaya yang perlu dilakukan dalam pengujian secara manual. Penerapan pengujian otomatis dapat mempercepat siklus pengujian dan meningkatkan produktivitas suatu tim. Namun demikian, pengujian otomatis secara menyeluruh dengan banyaknya kasus uji akan membutuhkan durasi waktu yang cukup lama dalam proses pengujiannya. Oleh karena itu, diperlukan metode yang tepat dalam membuat skrip pengujian otomatis supaya dapat mengurangi langkah aksi yang berlebih dan berulang-ulang antar kasus uji. Makalah ini bertujuan untuk membahas implementasi metode *Automation Journey* dalam pengujian otomatis dengan cara menambah logika baru dalam skrip pengujian untuk mengurangi langkah aksi yang kurang efektif serta mengubah urutan kasus pengujian agar saling berkaitan satu sama lain. Dengan adanya implementasi ini dapat mempercepat proses *End-to-end testing* secara otomatis namun menghasilkan pengujian yang lebih akurat dan skalabel daripada sebelumnya.**

Keywords—Pengujian perangkat lunak, Pengujian otomatis, Automation Journey, End-to-end testing

I. PENDAHULUAN

Perusahaan teknologi saat ini menghadapi tuntutan kualitas produk yang semakin meningkat setiap harinya tetapi dengan waktu pengembangan yang lebih singkat. Hal ini dapat mempengaruhi keseluruhan proses pengembangan dari awal hingga akhir atau biasa disebut *Software Development Life Cycle* (SDLC) yang digunakan sebagai kerangka kerja dalam proses pengembangan aplikasi [1]. Salah satu tahapan krusial yang terpengaruh adalah tahap pengujian (*testing*). Pada tahap ini seorang *tester* akan melakukan pengujian secara menyeluruh pada sistem untuk menilai apakah *software* dapat bekerja sesuai fungsionalitas yang diharapkan dan memastikan bahwa sistem bebas dari cacat atau *error*. Proses ini dahulu sering dilakukan secara manual tetapi sangat memakan waktu dan biaya sehingga rawan terjadi kesalahan (*human error*) [2]. Padahal untuk menghindari banyaknya *bug* dalam perangkat lunak diperlukan pengujian secara menyeluruh, sebelum perangkat lunak yang dikembangkan dirilis kepada publik atau pengguna umum.

Aplikasi jual beli online (*marketplace*) merupakan salah satu jenis aplikasi yang memiliki ratusan hingga ribuan *developer* yang tiap harinya dapat melakukan perubahan pada repositori aplikasi. Sementara itu, pada setiap fitur yang dikembangkan harus terdapat *integration* dan *regression test* untuk memastikan bahwa fitur tersebut dapat berjalan dengan baik dan tidak menyebabkan munculnya *bug* baru maupun *bug* lama yang sudah pernah ada. Jika hanya puluhan kasus uji (*test case*) mungkin bisa dilakukan secara manual, namun jika aplikasi sudah sangat berkembang dan terdapat hingga ribuan kasus uji maka usaha *tester* akan habis jika harus melakukan semua pengujian tersebut secara manual.

Berdasarkan permasalahan di atas maka diusulkan pengujian otomatis sebagai solusi untuk mengatasinya. Pengujian tersebut dapat berjalan otomatis oleh sistem tanpa interaksi manusia sehingga frekuensi pengujian dapat meningkat dan memberikan umpan balik lebih cepat kepada pengembang. Saat ini fokus pengujian sudah bergeser yang sebelumnya semua pengujian dilakukan secara manual, saat ini sudah dilakukan secara otomatis oleh robot yang skripnya sudah dibuat sesuai kasus pengujian. Dalam membuat pengujian otomatis akan lebih banyak *setup* di awal. Hal ini dilakukan untuk memastikan bahwa implementasi kode dalam skrip itu sudah efektif sehingga dapat melakukan pengujian secara otomatis dengan hasil yang akurat, dijalankan secara cepat, dan tidak banyak proses perbaikan terhadap skrip seiring perkembangan waktu.

Dengan penerapan solusi tersebut maka diharapkan dapat membantu proses pengembangan perangkat lunak dalam proses SDLC khususnya dan menerapkan pengujian otomatis dalam *Continuous Integration* (CI) *Pipeline*. Dalam proses CI atau pengintegrasian kode ke dalam repositori terdapat proses verifikasi dengan cara menjalankan proses pengujian secara otomatis, cepat, dan sering yang dilakukan secara terus menerus setiap ada perubahan atau fitur baru pada aplikasi [3]. Sehingga jika terdapat *bug* atau *error* dapat diketahui sedini mungkin.

II. LANDASAN TEORI

A. *Manual Test*

Manual test adalah pengujian yang dilakukan secara manual menggunakan kasus uji yang sudah dibuat untuk memastikan tidak adanya *bug* dalam aplikasi [4]. *Manual test* berperan penting sebagai tahap pengujian paling awal pada

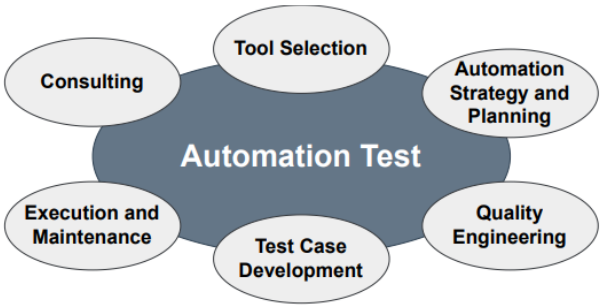
pengembangan fitur baru, fitur eksperimen, dan fitur dengan data dinamis. Untuk kasus-kasus tersebut maka pengujian manual yang akan dipakai oleh seorang *tester*. Pengujian manual pada *marketplace* secara umum terdapat 3 jenis, yaitu:

1. *Sanity Test*: merupakan pengujian yang berfokus pada fungsi baru yang dikembangkan saja. *Sanity testing* biasa dilakukan pada fitur yang benar-benar baru dan tidak memiliki kaitan dengan fitur lain sama sekali, sehingga seorang *tester* cukup memastikan fitur ini dapat berjalan dengan baik [5]. *Sanity test* biasa dilakukan pada *staging environment*.
2. *Smoke Test*: pengujian yang berfokus kepada keseluruhan perangkat lunak, namun hanya pada fungsi-fungsi utama yang bersifat *critical* dan umum. Pengujian ini bertujuan untuk memastikan bahwa *flow* tetap berjalan dengan baik dan aman [5]. *Smoke test* biasa dilakukan di *production* setelah ada unggahan fitur-fitur baru, yang bertujuan memastikan bahwa fitur-fitur utama aplikasi aman dan dapat berjalan dengan baik.
3. *Regression Test*: pengujian secara menyeluruh dan sedetail mungkin pada semua fitur dari yang utama hingga yang jarang digunakan, untuk memastikan bahwa fitur dapat berjalan 100% [5]. *Regression test* biasa dilakukan setelah terdapat fitur baru yang berkaitan dengan fitur-fitur lainnya. Selain itu *regression test* merupakan pengujian yang rutin dilakukan oleh *tester* mulai dari seminggu sekali, seminggu dua kali, dan lain-lain. *Regression test* bertujuan memastikan perubahan yang dilakukan selama pengembangan tidak menyebabkan munculnya *bug* lama ataupun baru. *Regression test* juga untuk memastikan bahwa tidak ada *bug* tua yang muncul akibat *deployment* yang terus menerus terjadi.

B. *Automation Test*

Automation testing merupakan pengujian yang bergantung pada *script* uji yang telah dibuat oleh *tester* dan dilaksanakan secara otomatis untuk membandingkan hasil sebenarnya dengan hasil yang diharapkan. Kelebihan pengujian otomatis adalah dapat dilakukan secara berulang, memberikan umpan balik dengan lebih cepat, dan berjalan otomatis tanpa interaksi manusia [4]. Dengan demikian, pengujian ini dapat meningkatkan frekuensi pengujian dan memungkinkan untuk melakukan uji regresi tanpa intervensi dari *manual tester* sama sekali. Dahulu pengujian sering dilakukan secara manual, tetapi seiring berjalannya waktu proses tersebut sangat memakan waktu, banyak biaya, dan rawan terjadi kesalahan, sehingga dibutuhkan pengujian otomatis yang diusulkan sebagai solusi untuk ini.

Gambar 1 menunjukkan poin-poin penting dari pengujian otomatis untuk mencapai siklus hidup pengembangan perangkat lunak (SDLC) yang produktif dan skalabel. Dengan memasukkan pengujian otomatis ke dalam SDLC akan memungkinkan suatu tim mencapai hal-hal seperti *continuous integration and continuous delivery (CI/CD), DevOps, Quality Engineering, Risk Mitigation*, pengiriman lebih cepat kepada pengguna, dan masih banyak lagi.



Gambar 1. *Automation Test Component* (Sumber: https://infuse.it/quality-engineering/test-automation/)

C. *UI Testing*

User Interface (UI) merupakan tampilan program yang dilihat langsung oleh pengguna atau biasa disebut *front-end* sebagai media interaksi antara pengguna dengan perangkat lunak. Pengujian UI atau *front-end* adalah pengujian terhadap antarmuka dengan cara melakukan urutan peristiwa sesuai urutan *user story* dan kasus pengujian yang dibuat melalui sejumlah komponen antarmuka seperti klik tombol, membuka tab menu, memasukkan teks, dan lain-lain [2].

Pengujian antarmuka tidak memerlukan informasi apapun dari *back-end* seperti respon *web service*, *database*, dan sejenisnya, namun perlu memeriksa semua fungsionalitas aplikasi sudah berjalan sesuai yang diharapkan. Selain itu pada UI *testing*, *tester* perlu memeriksa waktu respon yang ditimbulkan dari kompleksitas rancangan antarmuka, karena kecepatan aplikasi dalam memproses suatu permintaan sangat berpengaruh terhadap pengalaman pengguna. Pengujian antarmuka bertujuan untuk memastikan aplikasi telah memberikan layanan terbaik kepada pengguna, supaya pengguna lebih mudah dan nyaman dalam menggunakan aplikasi.

Proses pengujian antarmuka dapat dilakukan secara manual maupun otomatis. Meskipun demikian, pengujian dengan cara manual memiliki lebih banyak kekurangan seperti cakupan pengujian yang kurang luas, karena *tester* harus melakukan hal yang berulang-ulang sehingga rawan terjadi kesalahan jika terdapat halaman antarmuka yang terlewat dalam pengujian. Selain itu, kekurangan lainnya saat pengujian manual yaitu *tester* tidak bisa melakukan pencatatan waktu respon secara otomatis, sehingga hasil pengujian menjadi kurang valid [2]. Kelemahan-kelemahan tersebut dapat ditutupi dengan pengujian otomatis, karena dengan *automation testing* memungkinkan pengulangan urutan pengujian secara lebih cepat dan tepat. Selain itu, pengujian otomatis juga memungkinkan pencatatan waktu respon secara otomatis.

D. *Back-end Test*

Back-end adalah segala hal yang berhubungan dengan *server* dan *database*. *Back-end* tidak berinteraksi secara langsung dengan pengguna akhir karena sifatnya hanya menyampaikan informasi. *Back-end* lebih berfokus pada *database*, *scripting*, dan *server website* (*server-side*) [6]. Tujuan pengujian *back-end* adalah untuk memastikan lapisan *web service* dan *database* dari perangkat lunak bebas dari cacat basis data seperti kebuntuan, kerusakan data, atau kehilangan data. Pengujian *back-end* juga dikenal sebagai pengujian basis data. Data yang dimasukkan di *front-end* akan disimpan di database *back-end*. Basis data dapat berupa SQL Server, MySQL, Oracle, DB2, dan lain-lain. Data akan diatur dalam tabel sebagai catatan dan digunakan untuk mendukung

konten halaman. Pengujian *database* atau *back-end* penting karena jika tidak dilakukan dengan benar dapat menyebabkan beberapa komplikasi serius seperti kebuntuan, korupsi data, dan kehilangan data.

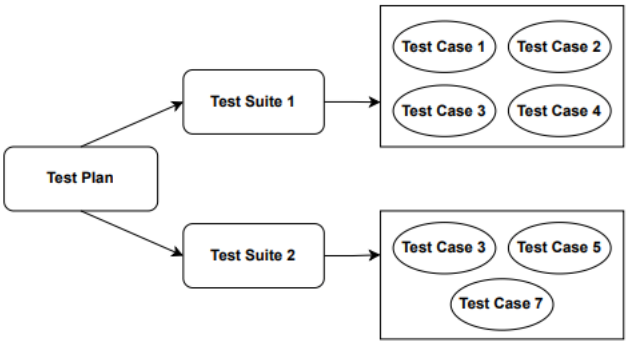
Proses pengujian *back-end* dapat dilakukan secara manual ataupun otomatis. Namun pengujian otomatis lebih sering dilakukan karena dapat mencakup banyak kasus uji sekaligus dalam satu waktu pengujian. *Back-end automation* diperlukan untuk deteksi awal pada kasus pengujian yang rawan terjadi *deadlock*, *data loss*, dan *data corruption*, sehingga pengujian ini dapat digunakan untuk memastikan bahwa proses memasukkan data ataupun menampilkan data pada *front-end* akan berjalan aman dan sesuai dengan harapan.

E. *Test Case*

Test case atau biasa disebut kasus uji adalah suatu dokumen dengan rangkaian tindakan yang dapat dilakukan oleh *tester* untuk melakukan verifikasi terhadap fitur atau fungsi tertentu dari sebuah aplikasi. *Test case* memiliki sekumpulan data uji, prasyarat, hasil yang diharapkan, dan kondisi pasca yang dikembangkan sesuai skenario kasus uji tertentu guna melakukan verifikasi apakah perangkat lunak sudah berjalan sesuai dengan yang diharapkan atau belum [7].

Dalam membuat kasus uji, skenario *testing* dibagi menjadi dua yaitu skenario positif dan negatif. Skenario positif adalah kasus pengujian menggunakan alur yang sesuai dengan harapan sistem pada *marketplace* sehingga tidak ada notifikasi atau *warning error* yang muncul saat sedang dijalankan, sedangkan skenario negatif adalah kasus pengujian dengan alur yang menyalahi aturan sistem seperti mengisi harga barang menjadi 0 atau minus dan sebagainya. Skenario negatif membutuhkan sistem untuk menolak permintaan tersebut dan mengirim notifikasi pesan peringatan.

F. *Test Suite*



Gambar 2. *Test Suite* dan *Test Case*

Test suite adalah sebuah wadah yang memiliki serangkaian *test case* seperti terlihat pada Gambar 2, yang dapat dijalankan secara bersamaan (pararel) ataupun satu persatu (*sequence*). Suatu kasus uji dapat ditambahkan pada beberapa *test suite* secara bersamaan, karena *test suite* dibuat berdasarkan siklus atau lingkup dari suatu rencana pengujian (*test plan*) [7].

Keuntungan dari *test suite* adalah memungkinkan tester menggabungkan beberapa *test case* untuk menciptakan kondisi pengujian yang unik dan bervariasi antar *test suite* satu dan lainnya. Dengan penggunaan *test suite*, pengujian dapat dilakukan dengan lebih lengkap seperti E2E (*end-to-end testing*) yang tidak dapat diproduksi oleh suatu *test case* itu sendiri.

G. *Automation Journey*

Automation journey adalah sebuah teknik yang dapat dilakukan saat membuat skrip pengujian otomatis yang bertujuan untuk mengurangi waktu eksekusi dan meningkatkan tingkat keberhasilan. Penerapan *automation journey* dapat dilakukan dengan cara mengubah susunan kasus uji, serta menambah percabangan kondisi di awal dan akhir kasus uji. *Automation journey* diterapkan pada *test suite* yang memiliki kasus pengujian dengan langkah aksi yang berulang dan serupa, seperti contoh kasus pengujian yang terlihat pada Tabel 1 berikut.

Tabel 1. Contoh *Test Suite*

Test Suite		
Test Case A	Test Case B	Test Case C
1. Open browser 2. Login 3. Onboarding X 4. Do checking A	1. Open browser 2. Login 3. Onboarding X 4. Do checking B	1. Open browser 2. Login 3. Onboarding X 4. Do checking C

Dalam contoh tersebut *tester* dapat melewati langkah berulang yaitu "*Open browser A*" lalu "*Login*" dan melakukan "*Onboarding X*" untuk mengurangi waktu eksekusi secara keseluruhan dan meningkatkan tingkat keberhasilan. Oleh karena itu, akan ada penambahan kecil pada langkah akhir dari contoh pertama dan kedua untuk mengembalikan keadaan di mana pengecekan B dan C dimulai. Dalam penerapannya tidak semua *test case* dan *test suite* dapat dengan mudah diubah menjadi *automation journey*, karena perlu rencana untuk membuat *test suite* supaya menjadi satu aliran tes.

Dengan penerapan *automation journey* seperti terlihat pada Tabel 2, maka tahap perulangan yang sama dan duplikat hanya dilakukan sekali saja yaitu pada *test case* yang pertama. Dengan demikian *test case* selanjutnya dapat langsung melakukan pengecekan pada langkah aksi *test case* selanjutnya. Namun jika *test case* sebelumnya memiliki hasil pengujian yang gagal, maka *test case* berikutnya harus melakukan tahap dari awal lagi.

Tabel 2. *Test Suite* dengan *Automation Journey*

Test Suite with Automated Journey		
Test Case A	Test Case B	Test Case C
1. Open browser 2. Login 3. Onboarding X 4. Do checking A 5. Press back	1. If IndividualRun 2. Do checking B 3. Press back	1. If individualRun 2. Do checking C

Secara keseluruhan, rangkaian pengujian yang berjalan dalam metode *automation journey* akan berjalan lebih cepat karena tidak perlu repot membuka dan menutup *browser* ataupun melakukan langkah-langkah yang sama berulang karena sudah terdapat kondisi percabangan untuk menangani kasus-kasus tersebut. Adapun kekurangannya yaitu *tester* perlu meluangkan lebih banyak waktu untuk menemukan dan memikirkan kasus uji yang dapat dijalankan melalui alur perjalanan logis *automation journey*.

H. *Agile dan Scrum*

Agile adalah metodologi pengembangan aplikasi yang didasari dengan prinsip kerja yang memerlukan adaptasi

dengan cepat, karena *agile* mengharuskan anggota tim untuk selalu siap terhadap segala perubahan yang akan terjadi, dan bersifat fleksibel ketika menghadapi suatu permasalahan [8]. *Agile* hanyalah sebuah *framework* yang berisikan prinsip atau sifat untuk menyelesaikan masalah secara adaptif. Maka dari itu penerapan *Scrum* dibutuhkan untuk mewujudukan *agile* tersebut menjadi sebuah langkah-langkah solusi. *Scrum* merupakan metode yang mengatur manajemen dan pelaksanaan pengembangan aplikasi. *Scrum* membantu koordinasi antar tim supaya lebih mudah, terstruktur, dan kolaboratif antar anggota tim. *Scrum* berguna untuk mempercepat rilis produk kepada pengguna dengan produktivitas dan kualitas yang tinggi.

Komponen penting dalam *Scrum* adalah *sprint*. *Sprint* merupakan sejumlah rencana pekerjaan yang harus diselesaikan oleh tim dalam periode waktu yang telah ditentukan. Tujuan dari *sprint* adalah untuk memecah *task* pekerjaan menjadi beberapa *task* yang berukuran lebih kecil, sehingga memungkinkan tim untuk merencanakan satu *sprint* dalam satu waktu dan menyesuaikan *sprint* ke depannya berdasarkan hasil *sprint* sebelumnya.

I. Tinjauan Pustaka

Terdapat beberapa penelitian yang berkaitan dengan pengujian otomatis menggunakan Katalon studio. Penelitian yang pertama berjudul "*Automation Testing Tool Dalam Pengujian Aplikasi The Point Of Sale*" [9]. Penelitian tersebut menjelaskan langkah-langkah pengujian otomatis menggunakan *automation testing tools* Katalon studio, mulai dari tahap pembuatan *test case* secara manual hingga akhirnya menjadi sebuah rancangan pengujian otomatis. Penelitian tersebut berfokus menganalisis efektivitas penerapan pengujian otomatis pada aplikasi *point of sale*. Berdasarkan hasil dari penelitian tersebut ditemukan bahwa pengujian otomatis sangat membantu terutama dalam pengujian yang memiliki data statis dan perlu dilakukan secara berulang.

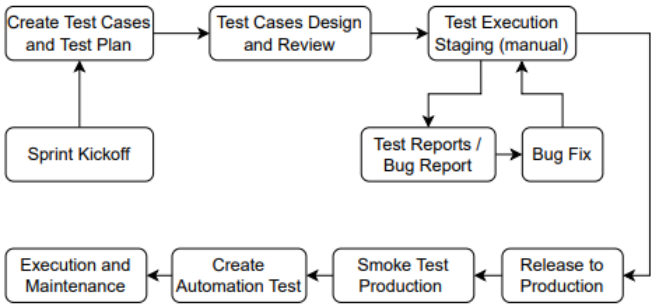
Penelitian kedua adalah penelitian yang berjudul "*Pengujian Black Box Aplikasi Mobile Menggunakan Katalon Studio*" [10]. Penelitian ini membahas tentang penerapan pengujian otomatis dalam pengembangan aplikasi, sebagai solusi dari pengujian manual yang kurang maksimal dan rawan terjadi kesalahan manusia. Dari penelitian tersebut diperoleh kesimpulan bahwa pengujian otomatis jauh lebih efektif dibanding dengan pengujian secara manual, terutama dari sudut pandang *Tester*, karena pengujian dapat dikukan lebih cepat, berulang, lebih lengkap, dan akurat.

Terdapat juga penelitian yang membahas mengenai *response time* dari halaman utama *website e-commerce.* Penelitian tersebut berjudul "*Analisis GUI Testing pada Aplikasi E-Commerce menggunakan Katalon*" [11]. Penelitian ini membahas mengenai analisis *response time* yang dibutuhkan dari halaman-halaman utama *website e-commerce,* seperti JD.ID, Tokopedia, dan Bukalapak. Halaman website yang dilakukan pengujian adalah halaman utama, halaman masuk, halaman pencarian, halaman detail produk, dan halaman transaksi. Hasil dari penelitian tersebut memperlihatkan bahwa *response time* dari antarmuka halaman *website* dipengaruhi oleh banyak faktor, tidak hanya dari kompleksitas GUI *website*. Oleh karena itu, penelitian tersebut masih perlu dilanjutkan untuk mengetahui faktor-faktor lain yang dapat mempengaruhi *response time* pada halaman *website*.

Berdasarkan hasil perbandingan dari berbagai penelitian tersebut, dapat disimpulkan bahwa pengujian otomatis antarmuka adalah jenis *black box testing*, karena pengujian ini dilakukan untuk menguji fungsionalitas aplikasi, tanpa melakukan pengecekan hingga kode internal program. Selain itu juga diperoleh bahwa Katalon studio merupakan *tools* pengujian otomatis yang paling populer saat ini, karena sifatnya yang mudah digunakan dan mendukung semua *platforms* pengujian (android, ios, mobile browser, dan website). Namun demikian, ternyata masih belum terdapat penelitian yang mengkaji terkait metode-metode yang dapat digunakan dalam membuat *script* pengujian otomatis, khususnya metode *Automation Journey.*

III. METODOLOGI

Metodologi yang digunakan dalam pengujian ini adalah *blackbox testing* dengan prinsip kerja *scrum.* Pengujian *blackbox* berfokus menguji tampilan aplikasi, fungsi-fungsi aplikasi, dan kesesuaian alur fungsi sesuai dari sudut pandang pengguna, tanpa menguji struktur internal atau *source code program* secara langsung [9]. Gambar 3 mengilustrasikan proses pengujian aplikasi dari awal hingga akhir pada *marketplace*.



Gambar 3. Tahap Pengujian pada *E-Commerce*

Tahap paling awal dimulainya *sprint* adalah *sprint kickoff*. Pada tahap ini *developer* dan *tester* akan berdiskusi untuk memilih *task* mana saja yang sekiranya butuh pengujian oleh seorang *tester* serta memberikan prioritas dari masing-masing *task* tersebut dan estimasi waktu yang dibutuhkan.

Setelah *sprint* dimulai maka *tester* akan mulai membaca *product requirements document* (PRD) yang berisi deskripsi, data-data yang diperlukan, serta persyaratan-persyaratan lainnya terkait *task* tersebut yang nantinya akan digunakan oleh *tester* sebagai referensi dalam membuat sebuah *test case* atau kasus pengujian.

Dalam membuat kasus uji, skenario *testing* dibagi menjadi dua yaitu skenario positif dan negatif. Kasus pengujian yang sudah selesai dibuat oleh *tester* akan dilakukan *review* terlebih dahulu oleh *product manager* (PM) dan *developer* terkait untuk memastikan bahwa kasus uji tersebut sudah sesuai dengan semua kemungkinan yang mungkin bisa terjadi serta tidak ada kesalahpahaman terkait *task* tersebut antara PM, *tester*, dan *developer*. Dengan demikian *test case* pengujian tersebut dapat digunakan sebagai validasi apakah sebuah *task* sudah sesuai dengan persyaratan yang diharapkan atau belum.

Setelah kasus pengujian selesai di-*review* dan direvisi serta sudah mendapatkan persetujuan dari PM dan *developer*, maka *tester* dapat segera melakukan pengujian terhadap fitur tersebut pada *environment staging* terlebih dahulu. *Staging* adalah lingkungan tiruan dari *software* utama yang sudah rilis kepada pengguna namun tidak terhubung ke *production*

secara langsung sehingga cocok digunakan sebagai tempat pengujian awal di mana *tester* dapat menguji secara menyeluruh tanpa perlu mengawatirkan risiko-risiko yang dapat terjadi. Dalam melakukan pengujian, *tester* perlu mencatat kasus uji mana saja yang belum terpenuhi serta mengisi daftar *bug* yang ditemukan untuk diberikan kepada *developer* supaya diperbaiki terlebih dahulu. *Bug* yang sudah diperbaiki akan dilakukan pengujian oleh *tester* menggunakan semua kasus uji yang sama seperti sebelumnya. Apabila semua kasus uji sudah terpenuhi dan sudah tidak terdapat *bug* yang ditemukan, maka *tester* dapat memberi persetujuan kepada fitur tersebut untuk *deploy* pada *production*.

Smoke Test Production adalah tahap pengujian terakhir yang dilakukan setelah fitur tersebut sudah sepenuhnya rilis kepada pengguna. Pengujian ini bertujuan untuk memastikan bahwa fitur tersebut sudah terimplementasi dengan baik dan sudah dapat dibuat pengujian otomatisnya menggunakan kasus uji dari pengujian manual yang digunakan sebelumnya.

Katalon studio digunakan dalam membuat skrip pengujian otomatisasi antarmuka karena menawarkan solusi pengujian otomasi yang komprehensif dan dapat digunakan pada semua platform. Selain itu, Katalon studio juga sudah menggunakan mesin selenium dan appium yang merupakan standar industri dalam hal *tools* pengujian perangkat lunak.

Pengujian otomatisasi yang sudah dibuat tidak dibiarkan begitu saja, tetapi selalu dilakukan *monitoring* setiap hari melalui *daily test report* yang dilakukan oleh sistem setelah pengujian otomatis dilaksanakan setiap pagi hari. Jika terdapat kasus uji yang gagal maka *tester* bertanggung jawab untuk mencari tahu penyebabnya. Apabila bukan *bug* namun terdapat pembaruan sistem atau perubahan *flow* maka *tester* perlu memperbaiki *automation script* supaya mutakhir dan sesuai dengan kondisi perangkat lunak saat ini.

Pada akhir *sprint* terdapat *sprint review* dan *sprint retrospective. Sprint review* digunakan untuk membahas *task* kemarin bagaimana perkembangannya dan menanyakan alasan apabila terdapat *task* yang belum selesai sehingga harus dilanjutkan lagi pada *sprint* berikutnya. Adapun *sprint retrospective* adalah kegiatan yang dilakukan untuk menilai kinerja setiap anggota tim dengan memberikan umpan balik dari pekerjaan *sprint* sebelumnya baik secara keseluruhan ataupun perorangan yang bertujuan untuk membuat perencanaan peningkatan pada *sprint* berikutnya.

IV. PEMBAHASAN DAN HASIL

Automation journey adalah sebuah metode yang digunakan dalam membuat skrip pengujian otomasi supaya lebih optimal dan skalabel untuk dijalankan setiap hari. Pengujian ini dilakukan untuk menguji tampilan aplikasi, fungsi-fungsi aplikasi, dan kesesuaian alur perjalanan pembeli saat melakukan transaksi (*buyer journey*).

Pada penerapan *automation journey*, secara umum terdapat 4 poin utama yang perlu diperhatikan, yaitu pemilihan kasus uji yang mendukung alur perjalanan logis, memberi percabangan bersyarat pada langkah awal setiap kasus uji, menambahkan variabel baru yang dibutuhkan untuk alur *journey* pada *test case* dan *test suite*, serta menambahkan *connector* pada bagian akhir setiap *test case*.

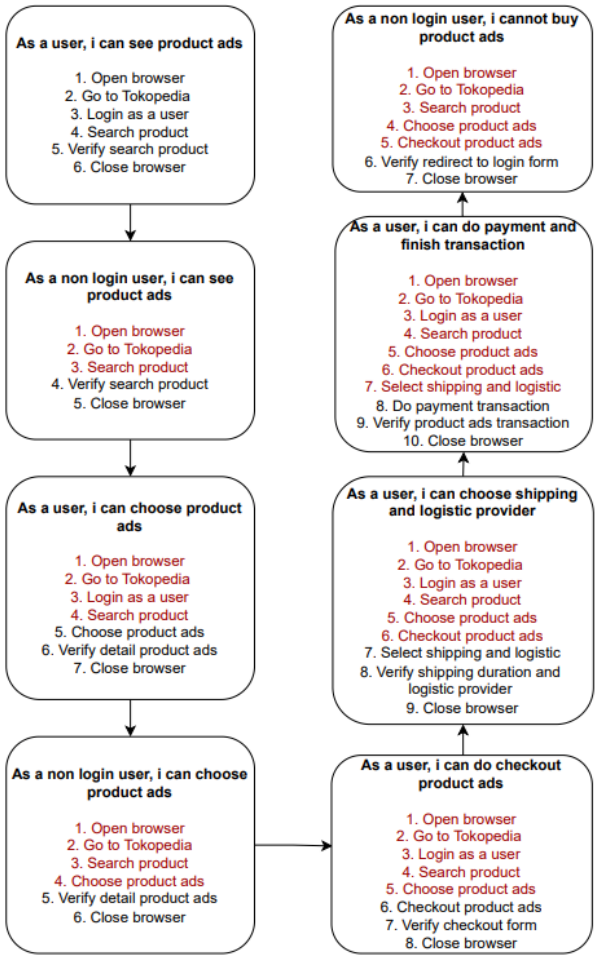
A. Memilih kasus uji dan mengatur urutannya

Pemilihan *test case* untuk dijalankan dalam *test suite* harus disesuaikan dengan alur perjalanan logis pengguna mulai dari awal hingga akhir. Dengan demikian langkah aksi dari satu kasus uji dengan kasus uji berikutnya tidak boleh jauh berbeda dan harus saling berkaitan karena akan membentuk alur perjalanan yang sama untuk mensimulasikan perilaku pengguna.

Tabel 3. *Test Suite* yang berisi 8 kasus pengujian dan belum disusun sesuai penerapan *Automation Journey*

Test Suite without Automation Journey	
No	Test Case
1	As a user, I can see product ads
2	As a non-login user, I can see product ads
3	As a user, I can choose product ads
4	As a non-login user, I can choose product ads
5	As a user, I can choose shipping and logistic
6	As a user, I can do payment and finish transaction
7	As a user, I can do checkout product ads
8	As a non-login user, I cannot buy product ads

Susunan *test case* pada *test suite* pada Tabel 3 terlihat tidak saling berkaitan satu sama lain. Selain itu, kasus uji untuk pengguna yang *login* dan *non-login* juga saling terpisah tidak berurutan sehingga perlu disusun ulang supaya lebih optimal dan dapat diterapkan *automation journey* pada *test suite* tersebut. Langkah aksi pada *test suite* yang belum menerapkan *automation journey* dapat dilihat pada Gambar 4. Terdapat beberapa langkah aksi berulang ditandai dengan warna merah yang harus dieksekusi berkali-kali sehingga jika pengujian tersebut dijalankan akan membutuhkan waktu yang cukup lama.



Gambar 4. Langkah aksi yang dibutuhkan oleh masing-masing kasus uji sebelum menerapkan *Automation Journey*

Oleh karena itu supaya *test suite* di atas lebih optimal dan dapat menerapkan *automation journey*, perlu diatur ulang susunannya serta memisahkan antara kasus uji untuk pengguna *non-login* dan pengguna *login*. Pada Tabel 4 terlihat *test suite* yang sudah diatur ulang memiliki urutan yang saling berkaitan satu sama lain dan terdapat pemisahan antara kasus uji *login* dan *non-login*. Dengan demikian tidak perlu melakukan aksi *login* dan *logout* lagi setiap kali berpindah kasus uji seperti sebelumnya yang membutuhkan tiga kali *login-logout*. Setelah diatur ulang hanya membutuhkan satu kali *login-logout* saja.

Tabel 4. *Test Suite* yang sudah diatur ulang susunannya sesuai penerapan *Automation Journey*

Test Suite with Automation Journey	
No	Test Case
1	As a non-login user, I can see product ads
2	As a non-login user, I can choose product ads
3	As a non-login user, I cannot buy product ads
4	As a user, I can see product ads
5	As a user, I can choose product ads
6	As a user, I can do checkout product ads
7	As a user, I can choose shipping and logistic
8	As a user, I can do payment and finish transaction

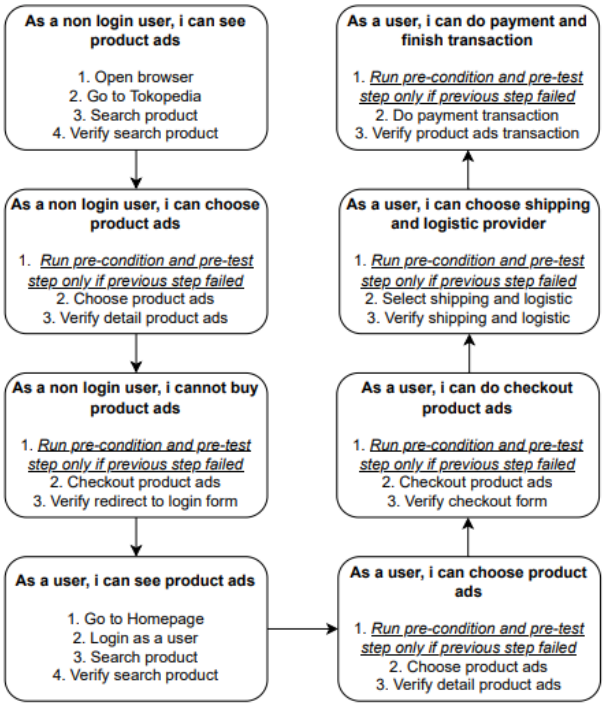
B. Memberi percabangan bersyarat diawal kasus uji

Kasus pengujian yang sudah diatur ulang sebelumnya jika dijalankan masih berjalan cukup lama karena masih terdapat langkah-langkah yang berlebih atau berulang pada setiap kasus uji. Hal ini dapat dimigrasi supaya hanya dijalankan satu kali saja dengan cara menambahkan percabangan bersyarat seperti terlihat pada Gambar 5 terjadi penambahan setiap awal kasus pengujian kecuali kasus uji pertama sehingga lebih efisien dan optimal pada saat dijalankan.

```
if (individualRun) {
// do login steps here
// also precondition steps here
}
```

Gambar 5. *Setup Test Case individualRun*

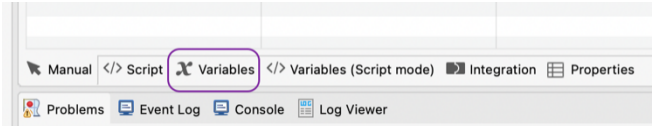
Pada Gambar 6 terlihat langkah aksi yang dibutuhkan setelah menerapkan *automation journey* menjadi lebih sedikit. Hal ini dikarenakan sudah ditambahkan logika percabangan untuk tidak menjalankan langkah aksi berulang yang sudah dijalankan sebelummnya. Kecuali jika kasus uji sebelumnya gagal atau terdapat *bug* atau cacat baru, maka langkah aksi berulang tersebut dijalankan. Hal ini akan menghemat banyak waktu pada saat dijalankan dalam pengujian otomasi.



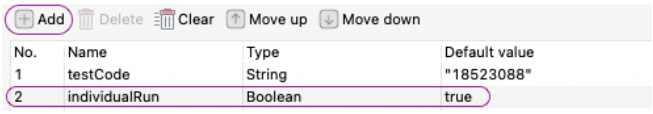
Gambar 6. Langkah aksi yang dibutuhkan setelah mengatur ulang susunan *test case* dan melakukan migrasi aksi yang berulang.

C. Menambahkan variabel baru pada *test case* dan *test suite*

Pada step sebelumnya telah ditambahkan percabangan bersyarat pada skrip pengujian menggunakan variabel *individualRun*. Namun jika dijalankan sekarang akan terdapat *error* yang disebabkan oleh *test case* yang belum mengenali variabel tersebut. Oleh karena itu perlu ditambahkan variabel *individualRun* beserta nilainya pada setiap kasus uji dengan cara memilih tab *Variables* pada kasus uji di sebelah kanan *script* seperti terlihat pada Gambar 7 dan 8.



Gambar 7. Tab *Variables* pada tools katalon studio



Gambar 8. Variabel *individualRu*n pada kasus uji

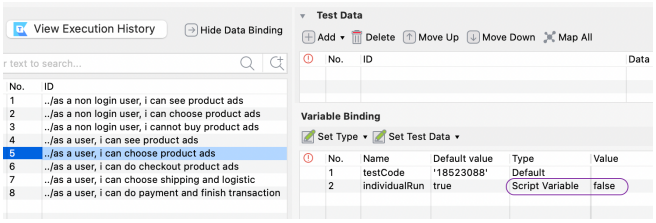
IndividualRun merupakan percabangan yang digunakan sebagai langkah mundur untuk menjalankan ulang langkah aksi *precondition* seperti *login* dan navigasi awal yang sebelumnya berulang (*redundant*) pada *test suite* tersebut. Aksi ini hanya dijalankan pada saat *test case* sebelumnya terdapat kegagalan atau *error* pada saat dijalankan yang mengakibatkan nilai *individualRun* menjadi '*true*' dan masuk logika percabangan bersyarat tersebut. Hal ini dilakukan supaya kasus uji berikutnya terbebas dari kegagalan atau *error* yang terjadi pada *test case* sebelumnya. Dengan demikian kasus uji yang sudah di-*reset* dari awal dapat membuat pengujian otomasi berjalan dengan valid dan meningkatkan kepercayaan diri *tester* terhadap hasil pengujian otomatisnya.

Pada *test suite* yang akan dijadikan sebagai *automated journey* juga perlu menambahkan variabel baru namun berbeda dengan *test case* yang menambahkannya pada *tab* variables. Pada *test suite*, variabel ditambahkan pada skripnya seperti terlihat Gambar 9 dengan cara mengubah nilai *setupskipped* menjadi *false* dan menambahkan variabel *isjourney* dengan nilai *true* di dalam *function def setup*.

```
// Setup test suite environment.
@SetUp(skipped = false)
def setUp() {
 GlobalVariable.isJourney = true
}
```

Gambar 9. *Setup Automation Journey* pada *Test Suite*

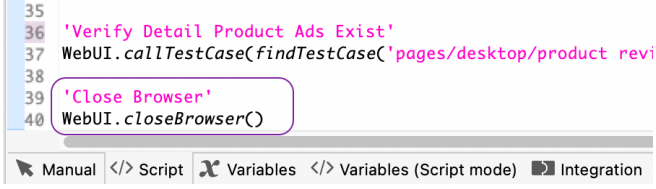
Semua *test case* sebelumnya sudah diberi nilai *true* untuk variabel *individualRun* pada nilai *default*-nya. Namun untuk menerapkan *automation journey* perlu dilakukan *binding* dari *individualRun* tersebut pada *test suite* dengan cara menambahkan tipe *script variable* dengan nilai *false* pada semua *test case* yang memiliki *individualRun* seperti terlihat pada Gambar 10. Pemberian nilai *false* bertujuan agar *test case* tidak menjalankan langkah aksi *setup* dan *precondition* yang berada di dalam percabangan bersyarat (*individualRun*) jika *test suite* tersebut masih berjalan lancar dan belum terdapat *test case* yang gagal atau *error* sebelumnya.



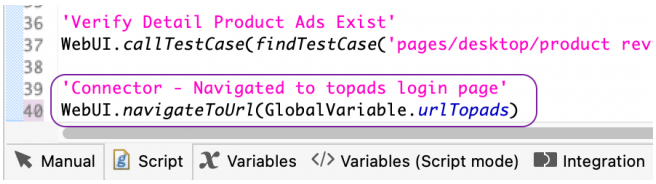
Gambar 10. Memberi nilai *false* pada individualRun tipe *script variable* untuk semua *test case* pada *test suite*

D. Menambah *connector* di akhir kasus uji

Connector digunakan untuk menghubungkan satu *test case* dan *test case* lainnya pada *test suite* yang sama supaya langkah paling akhir pada suatu *test case* dapat terhubung langsung ke langkah aksi pertama pada *test case* berikutnya. Pada langkah akhir dari *test case* yang belum menerapkan *automation journey* akan terdapat aksi '*Close Browser*' atau menutup *browser*. Awalnya aksi tersebut digunakan untuk mengakhiri *individual test case* dengan cara menutup *browser* yang digunakan untuk pengujian. Namun jika *test case* tersebut menerapkan *automation journey* maka aksi tersebut tidak dibutuhkan lagi dan dapat diubah menjadi *Connector* untuk mendukung *test case* berikutnya seperti terlihat pada Gambar 11 dan 12. Dengan menerapkan *connector*, alih-alih menutup *browser*, maka pengujian dapat berjalan lebih cepat karena *test case* berikutnya tidak perlu melakukan *setup* ulang dan hanya perlu melanjutkan langkah aksi baru yang belum dijalankan dari langkah-langkah *test case* sebelumnya.



Gambar 11. Langkah aksi *close browser* pada *test case*

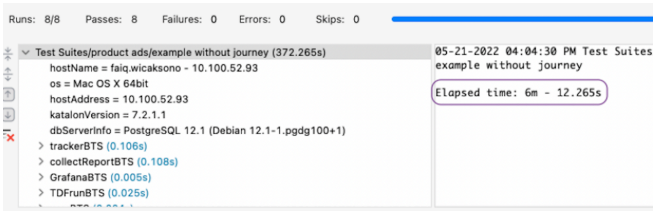


Gambar 12. *Connector* yang ditambahkan untuk persiapan *test case* berikutnya pada penerapan metode *automation journey.*

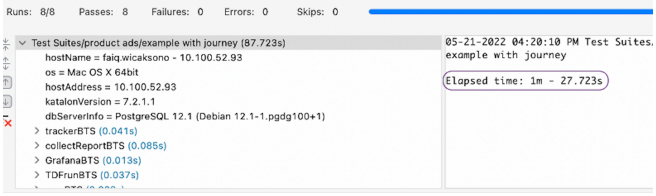
E. Hasil pengujian

Dalam rangka membandingkan hasil pengujian antara *test suite* yang menerapkan metode *automation journey* dan *test suite* yang tidak menerapkan metode *automation journey*, maka dibuat 2 eksperimen yaitu: menjalankan pengujian UI otomasi pada perangkat lunak tanpa *bug* dan menjalankan pengujian pada perangkat lunak yang terdapat *bug* atau cacat. Pada eksperimen pertama, pengujian akan dilakukan pada perangkat lunak yang valid dan tidak terdapat *bug* atau *error* pada tampilan antarmuka dan fungsi-fungsinya, sehingga diharapkan semua *test case* dapat berjalan dengan lancar.

Hasil pengujian pada *test suite* yang belum menerapkan metode *automation journey* membutuhkan waktu hingga 6 menit 12 detik untuk menyelesaikan seluruh rangkaian pengujian dengan hasil semua kasus uji lulus seperti yang diharapkan, seperti terlihat pada Gambar 13. Sedangkan terlihat pada Gambar 14, *test suite* yang sudah menerapkan metode *automation journey* hanya membutuhkan waktu 1 menit 27 detik untuk menyelesaikan seluruh rangkaian pengujian dengan hasil pengujian yang sama.

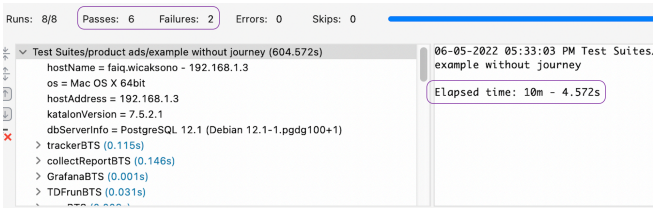


Gambar 13. *Test suite* yang dijalankan tanpa metode *automation journey*



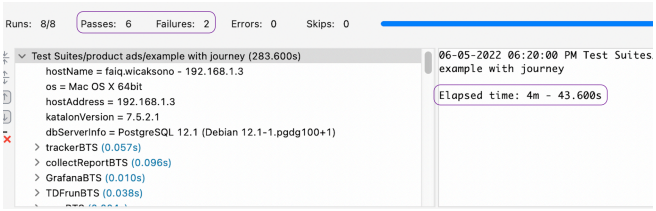
Gambar 14. *Test suite* yang dijalankan dengan metode *automation journey*

Pada eksperimen kedua, pengujian akan dilakukan pada sistem yang memiliki 2 *bug* atau cacat pada fungsi-fungsinya sehingga laporan pengujian yang diharapkan yaitu terdapat 2 *test case* yang gagal atau *failure*. Hasil pengujian pada *test suite* yang belum menerapkan metode *automation journey* terlihat pada Gambar 15. Terdapat 6 kasus uji yang lulus dan 2 kasus uji yang gagal sehingga hasil pengujian sudah sesuai harapan namun dengan catatan waktu pengujian 10 menit 4 detik.



Gambar 15. *Test suite* yang dijalankan tanpa metode *journey*

Sedangkan pada *test suite* yang sudah menerapkan metode *automation journey* seperti terlihat pada Gambar 16. Diperoleh hasil pengujian yang sama yaitu 6 kasus uji lulus dan 2 kasus uji gagal, namun untuk menyelesaikan seluruh rangkaian pengujian hanya membutuhkan waktu 4 menit 43 detik saja.



Gambar 16. *Test suite* yang dijalankan dengan metode *journey*

Dari kedua eksperimen tersebut dapat disimpulkan bahwa pengujian otomatis dengan metode *automation journey* memiliki hasil pengujian yang sama dengan pengujian otomatis tanpa *automation journey*. Meskipun demikian, waktu yang dibutuhkan dalam melaksanakan seluruh rangkaian pengujian jauh lebih cepat diselesaikan pada *test suite* yang menerapkan *automation journey*, seperti terlihat pada Tabel 5. Walaupun kasus uji yang digunakan sama persis, hanya berbeda urutan *test case* pada *test suite* serta metode yang diterapkan.

Tabel 5. Perbandingan 2 tipe *Test Suite* yang berbeda

***Test Suite* Eksperimen**	**Eksperimen tanpa *bug***	**Eksperimen dengan *bug***
Test suite* tanpa *Automation Journey	6 menit 12 detik	10 menit 4 detik
Test suite* dengan *Automation Journey	1 menit 27 detik	4 menit 43 detik

V. KESIMPULAN

Berdasarkan penerapan metode *automation journey* pada pengujian otomatis yang perlu dijalankan setiap hari, dapat disimpulkan bahwa penerapan metode ini dapat membuat pengujian otomatis yang dijalankan menjadi lebih cepat, efektif, dan skalabel. Siklus pengujian yang berjalan lebih cepat dapat meningkatkan persentase keberhasilan secara umum dan meningkatkan produktivitas suatu tim. Selain itu penerapan *automation journey* akan memudahkan *tester* dalam identifikasi *bug* yang ditemukan karena dengan dihapusnya langkah aksi yang berulang menyebabkan proses pencarian *bug* menjadi lebih mudah. Dalam hal perawatan dan penggunaan kembali kode sebagai skrip pengujian juga menjadi lebih mudah (*maintenable and reusable*) karena sudah tidak terdapat skrip yang *boilerplate* lagi, yaitu penulisan kode yang sama berulang-ulang (*redundant*) pada langkah aksi yang sama.

REFERENSI

[1] C. Klammer and R. Ramler, "A Journey from Manual Testing to Automated Test Generation in an Industry Project," *Proc. - 2017 IEEE Int. Conf. Softw. Qual. Reliab. Secur. Companion, QRS-C 2017*, pp. 591–592, Aug. 2017, doi: 10.1109/QRS-C.2017.108.

[2] E. Martantoh, "Pengujian Otomasi Vs. Pengujian Manual: Apa Perbedaannya," Jun. 2020. https://ekomartantoh.net/artikel/2020/06/29/pengujian-otomasi-vs-pengujian-manual-apa-perbedaannya/ (accessed May 26, 2022).

[3] F. NKD, "Agile vs DevOps vs CI/CD: Apa Saja Perbedaanya?," *logique*, 2020. https://www.logique.co.id/blog/2020/12/16/agile-vs-devops-vs-cicd/ (accessed Jun. 07, 2022).

[4] D. Rizky, "Manual Testing VS Automated Testing | DOT Intern | Medium," *medium*, 2019. https://medium.com/dot-intern/manual-testing-vs-automated-testing-9244aaed1ed5 (accessed Jun. 07, 2022).

[5] Z. T, "Sanity, Smoke, Regression Testing," *Linkedin*, May 2020. https://www.linkedin.com/pulse/sanity-smoke-regression-testing-zepri-togatorop/?trk=portfolio_article-card_title (accessed May 26, 2022).

[6] A. Damayanti, "Frontend Testing vs Backend Testing | Medium," *medium*, 2020. https://medium.com/@amaliadmyt/frontend-testing-vs-backend-testing-529d6a940835 (accessed Jun. 07, 2022).

[7] A. Damayanti, "Jenis-Jenis Test Deliverables | Medium," *medium*, 2020. https://medium.com/@amaliadmyt/jenis-jenis-test-deliverables-a8808f21c707 (accessed Jun. 08, 2022).

[8] A. Binar, "Mengenali Konsep Agile, Scrum, dan Sprint ala Perusahaan IT," Jan. 2022. https://www.binaracademy.com/blog/mengenali-konsep-agile-scrum-dan-sprint (accessed May 26, 2022).

[9] Y. Kosasih and A. Budi Cahyono, "Perancangan Sistem Dalam Pengujian Aplikasi The Point Of Sale (Studi Kasus TPOS PT. JAVASIGNA INTERMEDIA)," *Tek. Inform.*, vol. 3, no. 2, pp. 24–30, 2020.

[10] F. Ardi and H. P. Putro, "Pengujian Black Box Aplikasi Mobile Menggunakan Katalon Studio (Studi Kasus: ACC Partner PT. Astra Sedaya Finance)," 2021.

[11] M. M. Muhtadi, M. D. Friyadi, and A. Rahmani, "Analisis GUI Testing pada Aplikasi E-Commerce menggunakan Katalon," *Pros. Ind. Res. Work. Natl. Semin.*, vol. 10, no. 1, pp. 1387–1393, 2019, [Online]. Available: https://jurnal.polban.ac.id/proceeding/article/view/1443%0Apolban.ac.id