PENGUJIAN PROYEK *WEBSITE* OTOMATISASI DENGAN PENDEKATAN INTEGRASI ANTARA *SELENIUM* DAN *TESTNG* PADA *ENVIROMENT JENKINS*

Fransiskus Andika Setiawan[1], Septafiansyah Dwi Putra[1], Dwirgo Sahlinal[1]
[1] mahasiswa manajemen informatika, [2] pembimbing

Abstrak

Pengujian aplikasi adalah elemen penting untuk menjamin tinjauan akhir dari spesifikasi kualitas aplikasi, karena menghasilkan aplikasi bebas cacat program tanpa melakukan pengujian hampir mustahil. Pengujian secara manual pemeriksaan hasilnya bisa rawan kesalahan dan memakan banyak waktu, ketika tekanan jadwal meningkat pengujian manual sering dilupakan. Penelitian dengan judul "Pengujian Proyek *Website* Otomatisasi Dengan Pendekatan Integrasi Antara *Selenium* Dan *TestNG* Pada *Environment Jenkins*", memiliki rumusan masalah metode pengujian aplikasi yang efektif untuk meningkatkan kualitas pengembangan aplikasi.

Penelitian ini menggunakan metode *Design Science Research Methodology* (DSRM). Sumber data yang digunakan adalah hasil dari wawancara secara terstruktur dan mengamati metode pengujian aplikasi pada PT. Eannovate Creative Agency sebagai lokasi penelitian.

Berdasarkan analisis data yang dilakukan, diperoleh bahwa metode pengujian aplikasi yang efektif untuk meningkatkan kualitas pengembangan aplikasi adalah metode pengujian aplikasi secara otomatis. Proses pengujian otomatis lebih efektif dan efisien dalam hal waktu, tenaga dan biaya. Penerapan pengujian otomatis merupakan salah satu konsep pengembangan aplikasi modern *DevOps*.

***Kata Kunci**: software testing, continuous integration, framework testing, unit testing, automation testing.*

PENDAHULUAN

Pengujian aplikasi menjadi lebih populer dan penting dalam industri pengembangan aplikasi dalam beberapa tahun terakhir. Pengujian aplikasi merupakan salah satu elemen penting untuk menjamin kualitas aplikasi serta mewakili tinjauan akhir dari spesifikasi desain dan penulisan program aplikasi (R. M. Sharma 2014). Pengujian aplikasi merupakan kegiatan yang memakan banyak biaya dan usaha, menurut sebuah penelitian oleh Pierre Audoin Consultants, pada tahun 2010 biaya pengujian secara manual di seluruh dunia berjumlah 79 miliar euro jumlahnya diperkirakan akan melampaui 100 miliar euro pada tahun 2014. Jumlah itu seharusnya akan terus meningkat (Garousi and Yildirim 2018).

Hasil wawancara dengan pihak *Quality Assurance* dari PT. Eannovate Creative Agency didapatkan bahwa metode pengujian *website* yang dilakukan yaitu pengujian manual yang dilakukan secara langsung oleh seorang *Quality Assurance* sesuai dengan *test cases* program dan kombinasi kreativitas pengujian. Pihak *Quality Assurance* menyadari bahwa pengujian manual sering kali dirasa kurang efektif dan kurang akurat sehingga menyebabkan pengujian harus dilakukan secara berulang untuk memastikan bahwa *bug* tidak akan muncul lagi selain itu pengujian manual juga membutuhkan waktu yang lama karena pengujian *website* hanya bisa dilakukan setelah aplikasi terbentuk, untuk pengujian *website* sederhana seperti *company profile* bisa membutuhkan waktu satu sampai dua hari pengujian. Pelaporan *bug* juga masih dilakukan secara manual dengan memasukan hasil pengujian kedalam file *excel* yang sering menyebabkan terjadi keterlambatan dalam pelaporan *bug*.

Tahun 2013 *Computer Sciences Corporation* (CSC) melakukan survei internal sebanyak delapan puluh insinyur uji. Tolak ukur yang paling utama untuk melakukan survei dan berinvestasi dalam analisis data adalah kepuasan pelanggan. Berdasarkan survei pelanggan peringkat kepuasan pelanggan 87% telah dimiliki oleh perusahaan, dalam hal ini penemuan bug pada aplikasi berdampak pada penurunan grafik kepuasan pelanggan (O'Broin 2015).

Pengujian merupakan proses mengidentifikasi semua *bug* yang ada dalam produk aplikasi dan sebuah proses mengevaluasi semua komponen sistem dan memverifikasi bahwa sistem telah memenuhi persyaratan yang telah ditentukan atau untuk mengklasifikasikan perbedaan antara hasil dengan yang diharapkan, pengujian aplikasi dapat dilakukan secara manual dan secara otomatis (Bhateja 2015). Pengujian manual merupakan pengujian yang dilakukan dengan membuat *test cases* pengujian secara manual dan menjalankannya tanpa bantuan alat apa pun, semua proses pengujian dilakukan secara manual oleh manusia yang mencoba berbagai kombinasi penggunaan dan input serta membandingkan hasilnya dengan perilaku yang diharapkan dan mencatat hasil pengamatan mereka secara manual (R. M. Sharma 2014). Pengujian otomatis merupakan metode terbaik saat ini untuk

melakukan pengujian area fungsional yang luas pada program aplikasi serta menghemat waktu dan biaya yang dibutuhkan untuk melakukan pengujian aplikasi (Chaini and Pradhan 2015).

Banyak organisasi yang telah melakukan transisi dari pengujian manual ke pengujian otomatis, hal ini dilakukan untuk mempercepat siklus penemuan *bug*, perbaikan *bug*, memberikan umpan balik yang lebih cepat serta memungkinkan untuk melakukan pengujian secara berulang yang sulit dilakukan oleh manusia. Pada era modern ini pengujian secara manual merupakan kegiatan yang kuno dan kurang efektif diterapkan pada era sekarang, karena memakan banyak waktu dan biaya pengujian, ditambah lagi pengujian secara manual tidak akan lepas dari *human error* (Klammer and Ramler 2017).

Eksekusi dari pengujian secara manual pemeriksaan hasilnya bisa rawan kesalahan dan bisa memakan banyak waktu. Ketika tekanan jadwal meningkat, pengujian manual sering dilupakan yang menyebabkan aplikasi cacat ketika sudah dipublikasi, karena menghasilkan aplikasi yang bebas cacat program tanpa melakukan pengujian unit hampir mustahil. Proyek berdasarkan *Agile Development Methods* memiliki beberapa daftar peluncuran aplikasi yang masing-masing membutuhkan pengujian unit yang berulang dalam pengembangan aplikasi yang terintegrasi. Penerapan pengujian secara otomatis sangat dibutuhkan dalam hal melakukan pengujian unit yang berulang dalam pengembangan aplikasi yang terintegrasi (Rajasekhar and Shafi 2014).

Pengujian secara otomatis akan menjalankan *test cases* berdasarkan unit *testcase* dimana intervensi manual tidak digunakan lagi untuk melakukan proses pengujian. Pengujian otomatis dapat dilakukan dengan menggunakan aplikasi khusus untuk menulis dan menjalankan *test cases* yang telah dibuat untuk membandingkan hasil aktual dengan hasil yang diprediksi. Pengujian secara otomatis merupakan cara terbaik untuk meningkatkan efektivitas, efisiensi dan jangkauan pengujian aplikasi serta dapat melakukan apa yang tidak dapat dilakukan oleh pengujian secara manual. Dalam hal ini pengujian secara otomatis paling cocok diterapkan di lingkungan di mana persyaratannya sering berubah serta diperlukan sejumlah besar pengujian regresi dan *test cases* rumit yang harus dieksekusi secara berulang-ulang, pengujian otomatis juga dapat meningkatkan kualitas struktur pengujian serta mengurangi biaya perawatan di masa depan (S. Sharma 2012).

Proses pengujian aplikasi pada metode tradisional hanya bisa dilakukan setelah aplikasi selesai dibangun, dengan demikian proses pengembangan aplikasi menjadi tidak efektif dan efisien. Pengujian secara manual memberikan hasil yang kurang teliti karena dilakukan secara manual oleh manusia, selain itu juga jika menemukan banyaknya perubahan maka harus melakukan pengecekan secara manual kembali dari awal agar memastikan perubahan baru tidak akan merusak aplikasi yang sudah jadi. Hal tersebut sangatlah memakan banyak waktu dan tenaga karena lamanya proses pengujian, penemuan *bug* dan pelaporan *bug*, sehingga proses pengembangan aplikasi menjadi lebih lama.

Berdasarkan permasalahan yang ada maka dibutuhkan alat pengujian yang bisa berjalan secara otomatis dan terintegrasi, dengan begitu proses pengembangan aplikasi akan menjadi lebih efektif dan efisien dalam hal waktu, biaya dan tenaga. Selain itu banyak sekali keuntungan yang akan didapat dengan melakukan pengujian secara otomatis, dengan demikian juga salah satu konsep modern dalam pengembangan aplikasi yaitu konsep *DevOps* sudah turut diimplementasikan. Tujuan akhir dari pengujian secara otomatis adalah untuk mengurangi waktu antara konsep awal dan hasil akhir yang kemudian diimplementasikan kedalam program siap pakai (Soni 2015).

Tinjauan Pustakan

Definisi Aplikasi *Testing*

Aktivitas yang bertujuan mengevaluasi atribut atau kemampuan suatu program dan dapat menentukan bahwa program tersebut telah memenuhi persyaratan yang telah ditentukan merupakan definisi dari aplikasi *testing*. Pendekatan dalam melakukan pengujian juga tergantung pada metode pengembangan aplikasi yang dipakai, namun tujuan utama dari semua pendekatan metode pengujian adalah untuk menemukan masalah sesegera mungkin, untuk mengurangi biaya perbaikan di masa depan. Aplikasi *testing* akan menentukan apakah suatu produk aplikasi sudah berfungsi dengan benar dan efisien sesuai dengan persyaratan yang telah ditentukan (Zhang et al. 2014).

Pembagian Aplikasi *Testing*

Atribut aplikasi *testing* seperti seberapa besar bagian dari UAT (*Application Under Test*) yang sedang diuji atau pengetahuan penguji tentang aplikasi yang sedang diuji dan bagaimana cara melakukan pengujian aplikasi. Aktivitas pengujian dibagi menjadi beberapa jenis kelompok (Húska 2014).

Menurut Performa Pengujian

Cara pengujian aplikasi bisa dilakukan dengan pendekatan secara manual yang dilakukan oleh manusia dan pengujian aplikasi secara otomatis yang dilakukan oleh komputer.

1. Pengujian Manual
   Pengujian manual dilakukan secara manual oleh manusia yang mencoba berbagai kombinasi penggunaan dan input serta membandingkan

hasilnya dengan perilaku yang diharapkan dan mencatat hasil pengamatan mereka secara manual (R. M. Sharma 2014). Proses ini dapat dilakukan dengan mengikuti rencana pengujian atau dengan cara melakukan cara eksplorasi keseluruhan aplikasi (Húska 2014).

a. Rencana pengujian yang telah ditentukan

   Mengikuti rencana pengujian yang telah ditentukan serta yang perlu dilakukan untuk menguji fungsionalitas aplikasi.

b. Eksplorasi keseluruhan aplikasi

   Tidak terkait dengan rencana pengujian, oleh karena itu dengan cara ini akan membutuhkan sedikit waktu untuk persiapan supaya dapat menemukan masalah penting dengan cepat. Namun yang menjadi masalah adalah tidak semua penguji mengetahui detail aplikasi yang akan diuji karena dalam pengujian secara eksplorasi dibutuhkan keterampilan penguji (Húska 2014).

2. Pengujian Otomatis

Pengujian otomatis dilakukan dengan menjalankan *test cases* berdasarkan unit *testcase* dimana intervensi manual tidak digunakan lagi untuk melakukan proses pengujian. Pengujian otomatis dapat dilakukan dengan menggunakan aplikasi khusus untuk menulis dan menjalankan *test cases* yang telah dibuat untuk membandingkan hasil aktual dengan hasil yang diprediksi (S. Sharma 2012). Pengujian secara otomatis memiliki persyaratan lain yaitu:

a. Kemampuan untuk menjalankan subset dari semua daftar tes.
b. Pengaturan dan merekam *environment* variabel secara otomatis.
c. Menjalankan *test cases*.
d. Mencatat hasil dari setiap pengujian.
e. Membandingkan hasil aktual dengan yang diharapkan serta menyoroti perbedaan.
f. Menganalisis hasil dan memprosesnya secara komprehensif.

3. Kelebihan dan kekurangan

Pengujian secara manual dan otomatis memiliki kelebihan dan kekurangan. Konteks aplikasi harus membantu mana yang lebih cocok, konteksnya meliputi seberapa besar aplikasi tersebut, seberapa besar anggaran proyek atau kapan harus dirilis.

a. Kelebihan pengujian manual

   Waktu pengaturan pengujian manual lebih cepat dibandingkan dengan pengaturan ketika melakukan pengujian otomatis.

b. Kekurangan pengujian manual

   Tidak bisa melakukan proses pengujian secara berulang karena bisa menyebabkan hasil yang tidak akurat atau salah. Eksekusi pengujian secara manual juga lebih lambat dibandingkan dengan pengujian secara otomatis yang dilakukan oleh komputer.

c. Kelebihan dan kekurangan pengujian otomatis

   Pengujian otomatis memiliki banyak manfaat untuk proyek yang besar dan kompleks, karena biaya awal untuk otomatisasi dan pemeliharaan pengujian lebih mahal namun dengan begitu bisa menghemat banyak biaya di masa depan (Húska 2014). Beberapa manfaat dan kelemahannya dari pengujian otomatis yaitu:

   1) Otomatisasi dapat membantu mengurangi kesalahan yang bisa terjadi oleh *human error* serta masalah waktu yang menjadi kendala dalam pengujian manual yang dilakukan oleh manusia.
   2) Pengujian otomatis berjalan lebih cepat daripada pengujian otomatis, karena memiliki eksekusi pengujian yang lebih cepat hasilnya perlu mengakumulasi lebih cepat juga, oleh karena itu penguji perlu meninjau hasil dalam waktu yang lebih singkat.
   3) Pengujian secara otomatis bisa mengurangi biaya anggaran.

   Supaya bisa mengklaim otomatisasi dapat mengurangi biaya pengujian, maka perlu memaparkan biaya pengujian otomatis yang dibutuhkan secara keseluruhan yaitu:

   1) Biaya penerapan peranti uji.
   2) Biaya belajar menggunakan peranti uji.
   3) Biaya pengembangan peranti uji otomatis.
   4) Biaya pemeliharaan pengujian otomatis seiring perubahan produk.
   5) Biaya meninjau hasil pengujian.

   Biaya yang tersisa secara keseluruhan ini perlu ditambahkan untuk pengujian manual, karena beberapa pengujian manual masih diperlukan, terutama pada tahap pengujian ketika menguji *User Interface* (UI).

Menurut Tingkatan Pengujian

1. *White Box Testing*

White Box Testing dikenal sebagai metode pengujian berbasis kode atau pengujian struktural yang merupakan salah satu teknik pengujian aplikasi yang paling penting, dalam hal ini pengujian akan berjalan berdasarkan kode sumber dan cara kerja sumber internal. Dengan begitu menjadikan *white box testing* sangatlah efektif dalam melakukan validasi desain,

keputusan, asumsi serta menemukan kesalahan program dalam aplikasi (Barus et al. 2015).

2. *Black Box Testing*

Black Box Testing merupakan pengujian fungsional dan didefinisikan sebagai "pengujian yang mengabaikan mekanisme internal dari suatu sistem atau komponen yang hanya berfokus pada keluaran yang dihasilkan sebagai timbal balik dari masukan yang dipilih sebagai kondisi eksekusi" (Vilkomir, Baptista, and Das 2017).

3. *Gray Box Testing*

Gray box testing merupakan salah satu teknik pengujian yang menggabungkan dua teknik pengujian antara *black box testing* dengan *white box testing*. *Gray box testing* melakukan pengujian dengan menggunakan bahasa spesifikasi sesuai kebutuhan seperti teknik pengujian *black box testing* dan kemudian akan dilakukan verifikasi akan kebenarannya berdasarkan kode struktural seperti teknik pengujian *white box testing*.

Menurut Ruang Lingkup Pengujian

1. *Unit Testing*

Unit testing merupakan bagian terkecil dari aplikasi yang dapat diuji yang kemudian dapat dikompilasi dan uji coba. *Unit testing* biasanya melakukan pengujian berdasarkan modul yang biasanya dilakukan sebagai pengujian *white box testing* karena penguji perlu mengetahui kode dari aplikasi yang akan diuji. Ruang lingkup dari unit testing adalah penguji harus melakukan verifikasi dari bagian terkecil dari keseluruhan aplikasi (Húska 2014).

2. *Integration Testing*

Integration testing digunakan setelah melakukan pengujian unit dan sebelum fase pengujian aplikasi dilakukan. *Integration testing* mengambil sejumlah modul dan unit serta menggabungkan kedalam kelompok yang lebih besar yang disebut agregat. Tujuan utama dari *integration testing* adalah untuk menguji kelompok *unit testing* serta melihat apakah sudah berjalan secara bersama dengan benar dan terintegrasi (Vasˇkova 2014).

3. *System Testing*

System testing merupakan gabungan dari beberapa *unit testing* yang digabung menjadi satu komponen, beberapa komponen terintegrasi menjadi satu komponen besar yang disebut sistem. Hal ini bertujuan untuk menjelaskan masalah yang diekspos dengan melakukan pengujian seluruh sistem yang terintegrasi (Húska 2014).

Continuous Integration

Continuous integration berakar pada *extreme programming* (XP), yang merupakan salah satu metodologi pengembangan aplikasi modern yang bertujuan untuk meningkatkan kualitas dari aplikasi serta mengurangi waktu yang dibutuhkan dalam pengembangan aplikasi. Oleh karena itu *continuous integration* tidak hanya layak untuk pengujian integrasi, tetapi juga untuk melakukan pengujian unit dan pengujian sistem (Laukkanen and Mantyla 2015).

Key Principles

Proses *continuous integration* memiliki beberapa prinsip-prinsip yang perlu diperhatikan untuk meningkatkan kualitas aplikasi yang lebih baik yaitu :

1. Kode program harus disimpan kedalam repository penyimpanan kode, seperti *github, gitlab* dll.
2. Pengujian aplikasi harus dilakukan secara otomatis.
3. Perubahan kode program harus dikirimkan ke basis setiap kali terjadi perubahan kode program.
4. Setiap pengiriman perubahan kode program harus memicu pembangunan proyek dan proses pengujian pada mesin khusus.
5. Pembangunan dan pengujian harus dilakukan secepat mungkin.
6. Pengujian harus dilakukan pada lingkungan produksi.
7. Hasil harus jelas terlihat dan programmer harus tahu setiap kali perubahan kode menyebabkan *error* atau tidak.
8. Proses pembaruan aplikasi dan pelepasan aplikasi harus dilakukan secara otomatis (Húska 2014).

Problems

Manfaat yang diberikan oleh *continuous integration* lebih besar daripada kerugiannya, tetapi walaupun demikian masih perlu dilakukan pertimbangan juga, yang perlu dipertimbangkan adalah :

1. Investasi awal digunakan untuk keperluan perangkat keras, seperti mesin khusus untuk peranti *continuous integration*, pengembangan sistem dan pengujian *continuous integration* dilakukan. Biaya investasi dapat dikurangi dengan cara membuat mesin virtual.
2. Kerentanan keamanan perlu diperhitungkan, karena serangan terhadap sistem *continuous integration* dapat berakibat tersebarnya informasi rahasia atau pengambilan alih sistem yang bisa berdampak negatif pada proses pengembangan aplikasi.

3. Pengujian aplikasi perlu dilakukan secara otomatis, secara ekstensif dapat meningkatkan biaya pengembangan aplikasi secara keseluruhan. Tetapi dengan melakukan pengujian secara otomatis dapat menghemat biaya perawatan pengembangan aplikasi di masa depan (Húska 2014).

Definisi Aplikasi Perusahaan

Aplikasi perusahaan merupakan produk aplikasi yang secara khusus dirancang untuk memfasilitasi kerja sama dan koordinasi pekerjaan di seluruh perusahaan. Hal ini mencakup interkoneksi dari proses bisnis inti seperti penjualan, akuntansi, keuangan, sumber daya manusia dan lain sebagainya. Aplikasi perusahaan juga dibuat untuk menyediakan antarmuka untuk menghubungkan pelanggan, pemasok dan mitra bisnis lainnya kedalam sistem yang terintegrasi (Húska 2014).

Semakin banyak fungsi yang aplikasi perusahaan berikan maka semakin kompleks pula aplikasi yang dibuat. Hal ini yang menjadi kendala adalah pengembangan aplikasi membutuhkan banyak biaya untuk menciptakan aplikasi dengan kualitas yang bagus. Aplikasi perusahaan digunakan untuk melakukan pekerjaan dengan data penting yang membutuhkan proses secara *real time*, fokusnya adalah pada aspek penting dan tantangan seperti :

1. Performa Aplikasi.
2. *Data persistence*.
3. *Accessing data concurrently*.
4. *User friendly*.
5. Terintegrasi dengan aplikasi perusahaan lainnya.
6. Keamanan aplikasi.

Performa aplikasi, sangatlah dibutuhkan dalam aplikasi perusahaan untuk meningkat kualitas kinerja dari perusahaan. Aplikasi yang tidak memiliki performa yang bagus bisa menyebabkan defisiensi kinerja yang berdampak pada pelanggan, yang mengakibatkan kerugian moneter yang signifikan.

Data persistence, Aplikasi perusahaan bekerja untuk menyimpan banyak data yang harus terus berjalan selama beberapa tahun. Oleh karena itu sistem harus dapat beradaptasi dengan perubahan struktur data saat ada penambahan persyaratan baru untuk fungsionalitas sistem yang baru.

Accessing data concurrently, hal ini akan sangat berpengaruh pada aplikasi perusahaan yang berbasis *website*, yang dimana sejumlah besar pengguna dapat mengakses aplikasi secara bersamaan. Berdasarkan hal tersebut harus ada mekanisme yang dapat diandalkan untuk memastikan bahwa akses yang dilakukan bersamaan ini tidak menyebabkan kesalahan pada sistem aplikasi.

User friendly, Dengan kemudahan penggunaan dan tampilan yang mudah dipahami akan mengurangi kesalahan masukan data yang bisa menyebabkan aplikasi *error*. Dengan hal ini maka perlu disajikan dalam berbagai cara yang dapat memudahkan pengguna dalam mengakses aplikasi sehingga dapat mengurangi kesalah pada aplikasi.

Terintegrasi dengan aplikasi perusahaan lainnya, Aplikasi perusahaan perlu berkolaborasi dengan aplikasi lain untuk meningkatkan kualitas perusahaan. Masalahnya adalah aplikasi lain sering dibangun dengan bahasa pemrograman yang berbeda yang mempersulit untuk melakukan sinkronisasi aplikasi.

Keamanan aplikasi, keamanan aplikasi perusahaan merupakan hal yang paling penting dalam perusahaan karena aplikasi perusahaan mengontrol uang dan sumber daya, setiap pelanggaran keamanan dapat menyebabkan hilangnya data rahasia perusahaan dan kerugian finansial yang signifikan.

Pendekatan Pengujian dalam Perusahaan

Selain bentuk pengujian yang dijelaskan sejauh ini, tidak hanya pada aplikasi perusahaan yang membutuhkan pengujian.

Bentuk Pengujian

1. *Alpha testing*

Hal ini dilakukan oleh tim pengembang aplikasi internal atau tim yang menjamin kualitas aplikasi. Hal ini bertujuan untuk menemukan masalah dan kendala sebelum aplikasi siap untuk dirilis.

2. *Acceptance testing*

Digunakan untuk aplikasi yang telah disesuaikan dan dirancang untuk pelanggan tertentu. Hal ini pelanggan yang akan melakukan validasi terhadap aplikasi sesuai spesifikasi yang dibutuhkan oleh pelanggan.

3. *Installation testing validates compatibility with hardware platforms*

Portabilitas merupakan kemampuan untuk menjalankan aplikasi pada berbagai *hardware* dan *software* tanpa mengalami perubahan tampilan dan perubahan fungsi aplikasi yang diperlukan dari pengguna akhir aplikasi.

4. *Stress testing*

Digunakan untuk menyimulasikan perilaku abnormal dalam suatu program aplikasi dengan mempertimbangkan situasi yang menyebabkan akhir program aplikasi tidak normal, yang digunakan untuk membuat tolok ukur stabilitas aplikasi.

5. *Smoke testing*

Digunakan untuk melakukan konfirmasi bahwa perubahan baru tidak akan mengganggu proses integrasi pada produk aplikasi. *Smoke testing* harus dilakukan dengan cepat, karena sering digunakan untuk memvalidasi integrasi pada produk aplikasi dengan cepat sebelum pengujian yang lebih dalam dilakukan.

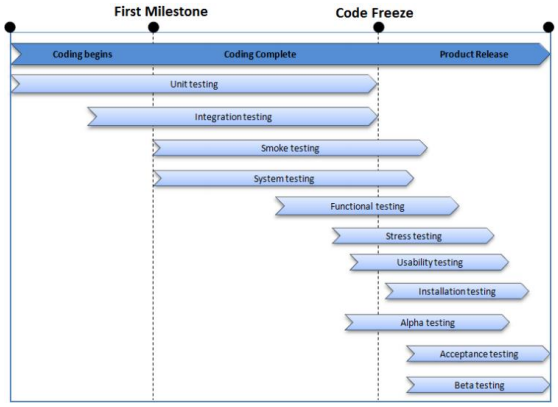
6. *Usability testing*

Digunakan untuk mengukur kesederhanaan menggunakan aplikasi. Dalam kasus ini aplikasi dengan *Graphical User Interface* (GUI), antarmuka ini merupakan subjek pengukuran misalnya akurasi penggunaan atau respons dari aplikasi.

Contoh Proses Pengujian pada Perusahaan

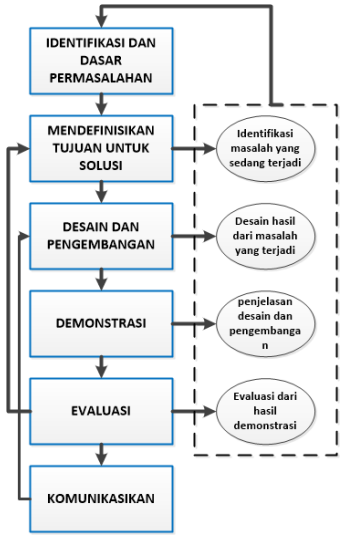
Untuk memenuhi persyaratan dari aplikasi yang dibutuhkan oleh perusahaan, aplikasi perusahaan perlu dilakukan pengujian dengan benar. Proses ini bisa menjadi proses yang sangat melelahkan, oleh karena itu dibutuhkan pengujian aplikasi yang efisien. Oleh karena itu dibutuhkan pengujian aplikasi secara otomatis, dengan kata lain apa yang bisa dilakukan secara otomatis maka dilakukan secara otomatis dan sisanya dilakukan secara manual. *Continuous integration* merupakan teknik lain yang bisa digunakan dalam perusahaan.

Gambar 1 menunjukkan contoh alur waktu proses pengujian aplikasi perusahaan. *First milestone* adalah istilah untuk pengembangan pada tahap awal dan *code freeze* adalah keadaan ketika tidak ada perubahan kode yang diizinkan untuk ditambahkan (Húska 2014).



Gambar 1. Proses Pengujian Aplikasi Perusahaan.

DSRM merupakan gambaran dari serangkaian keputusan yang secara keseluruhan dalam bentuk strategi untuk menjawab pertanyaan dari penelitian dan melakukan pengujian hipotesis. DSRM memandang desain penelitian sebagai bagian terstruktur untuk mengambil keputusan rasional atau sebagai pedoman yang membantu dalam menghasilkan hasil penelitian yang valid. Desain penelitian mencakup keputusan dari metode pengumpulan data, prosedur pengukuran, penskalaan, instrumen, sampel dan analisis data. Desain penelitian yang baik harus memastikan bahwa informasi yang diperoleh sudah relevan dengan masalah yang sedang diteliti dan dikumpulkan melalui prosedur objektif. Keluaran dari DSRM dapat dikategorikan sebagai *constructs*, *models*, *methods*, *instantiations* and *better theories*.(Azasoo and Boateng 2015).



Gambar 2. *Design Science Research Methodology.*

1. Identifikasi Dan Dasar Permasalahan

Proses dan hasil pengenalan masalah atau inventarisasi masalah, identifikasi masalah merupakan salah satu proses penelitian yang paling penting dari proses yang lain. Identifikasi dan dasar permasalahan akan menentukan kualitas suatu penelitian yang bisa dilakukan dengan melakukan studi literatur, observasi, dan lain sebagainya (Azasoo and Boateng 2015).

2. Mendefinisikan Tujuan Untuk Solusi

Proses membuat perencanaan tujuan yang telah didapat dari identifikasi dan perumusan masalah, kemudian diimplementasikan dalam bentuk solusi yang menjawab semua dari permasalahan yang terjadi (Azasoo and Boateng 2015).

3. Desain dan Pengembangan

Proses membuat desain dari mendefinisikan tujuan untuk solusi yang telah didapat. Tahapan ini merupakan tahapan yang paling penting untuk mendukung terbentuknya sistem yang baik (Azasoo and Boateng 2015).

4. Demonstrasi

Penjelasan dari tahapan pengembangan yang telah dibuat, pada tahapan ini dilakukan demonstrasi pada desain dan pengembangan. Hasil dari demonstrasi berupa laporan dari beberapa fungsi yang telah berjalan dalam sistem (Azasoo and Boateng 2015).

5. Evaluasi

Kegiatan yang dilakukan berkenaan dengan proses untuk menentukan nilai hasil dari sistem yang telah dibuat. Tahapan ini berisi hasil dari pengujian demonstrasi yang telah dilakukan (Azasoo and Boateng 2015).

6. Komunikasikan

Pada bagian akhir langkah penelitian adalah melakukan pemaparan atas penelitian yang telah dilakukan. Selain itu agar penelitian ini menjadi wahana transformasi pengetahuan bagi yang akan melakukan penelitian (Azasoo and Boateng 2015.

Jenkins

Jenkins adalah peranti *continuous integration* yang *open source*. *Jenkins* dibangun menggunakan *java* yang mendukung manajemen pembangunan proyek dan pengujian proyek aplikasi secara virtual. Tugas dasar dari *jenkins* adalah menjalankan beberapa tugas yang telah ditentukan atau dengan kata lain *jenkins* hanya menjalankan pekerjaan berdasarkan pemicu. Pemicu bisa berupa sebuah perubahan dalam *version control system* (VCS) misalnya *git*(Seth and Khare 2015).

Jenkins merupakan *cross-platform* dan *serveropen-source* yang paling banyak dipakai dan merupakan yang paling populer dalam platform *java*. *Jenkins* dibuat oleh Kohsuke Kawaguchi pada tahun 2004 yang awalnya memiliki nama *hudson* tetapi pada tahun 2011 namanya diganti menjadi *Jenkins* karena perselisihan dengan *oracle*. Peranti ini dibuat untuk menyederhanakan proses integrasi antara perubahan dan pelaporan. *plugin* yang tersedia sudah ada lebih dari 1.200, yang membuat jangkauan fungsionalitas *jenkins* lebih luas, selain itu *jenkins* juga memiliki komunitas yang memiliki pengguna pada 28 Desember 2016 sekitar 127.000 pengguna (Polkhovskiy 2016).

Selenium Web Driver

Selenium web driver adalah salah satu komponen dari *toolkitselenium*. *selenium web driver* diperkenalkan sebagai lanjutan versi pendahulunya *selenium rc*. Tidak seperti pendahulunya, pada *selenium web driver* tidak perlu menjalankan *server* terlebih dahulu untuk menjalankan pengujian *test cases*. Oleh karena itu, *selenium web driver* dianggap sebagai yang tercepat diantara semua komponen lain dari *toolkitselenium*.

Selenium web driver memberikan dukungan yang lebih baik untuk melakukan pengujian halaman *website* secara dinamis. *Selenium web driver* mendukung pengujian lintas browser seperti *safari*, *chrome*, *firefox* dan bahkan *browser* langka seperti *htmlunit*. Interaksi *selenium web driver* dengan halaman *website* yang lebih realistis. Contohnya *driver website* tidak akan dapat mengisi kotak teks yang tidak di aktifkansama seperti pengguna lain dari aplikasi. Oleh karena itu hasil tes dari *driver web* lebih realistis dan akurat (Ramya, Sindhura, and Sagar 2017).

TestNG

TestNG adalah kerangka kerja pengujian *java* yang dirancang untuk mencakup berbagai kategori tes, dari unit ke fungsional kemudian ke tes integrasi. *Testng* terinspirasi dari *junit* dan *nunit* tetapi memperkenalkan beberapa fungsi baru yang dibuat itu lebih kuat dan lebih mudah digunakan, seperti: satu set pernyataan untuk menguji kondisi logis, penjelasan dan itu didukung oleh berbagai peranti dan *plug-in* seperti *selenium, eclipse, maven* dll (Bentes et al. 2016).

Maven

Maven adalah peranti standar yang populer untuk membangun dan mengembangkan proyek *java*. Fitur utama dari *maven* adalah kemudahan dalam penggunaanya dan memenuhi kebutuhan dalam proyek pengujian dan penyebaran secara otomatis. *Maven* memiliki tiga siklus yang terpisah, *clean* digunakan untuk membersihkan proyek, *site* digunakan untuk membuat dan menerbitkan situs proyek dan *default* digunakan untuk menentukan semua langkah-langkah yang diperlukan untuk membangun dan mengeksekusi proyek, *default* merupakan bagian inti dari seluruh siklus *maven*(Xiong Zhen-hai and Yang Yong-zhi 2014).

Eclipse

Eclipse adalah *IDEopen source* yang diluncurkan oleh IBM pada tanggal 5 November 2001. *Eclipse* digunakan untuk membangun program komputer dan dapat dijalankan di semua sistem operasi serta memiliki banyak kelebihan seperti mendukung pengembangan aplikasi dengan berbagai bahasa pemrograman antara lain seperti *c/c++*, *cobol*, *python*, *perl*, *php* dan lain sebagainya, selain itu dapat digunakan juga dalam siklus pengembangan aplikasi seperti dokumentasi, tes aplikasi, pengembangan *web* dan lain sebagainya (Alfa Satyaputra 2014).

Java

Java adalah sebuah platform dan bahasa pemrograman tingkat tinggi yang memiliki karakteristik yang sederhana, berorientasi objek, terdistribusi, dinamis, aman dan lainnya. *Java* dikembangkan dengan model

yang hamper sama dengan seperti bahasa *c++* dan *smalltalk* namun memiliki kelebihan dalam kemudahan cara pakai nya, serta dapat dijalankan di semua sistem operasi. Dalam bahasa ini seluruh *source code* memiliki ekstensi *.Java* kemudian *javac compiler* akan mengubahnya menjadi *bytecode* dengan ekstensi *.class*(Alfa Satyaputra 2014).

IDE (*Integrated Development Environment*)

IDE adalah program computer yang menyediakan fasilitas yang dibutuhkan untuk pembangunan program aplikasi. IDE dapat juga diartikan sebagai program atau alat bantu yang terdiri dari *editor*, *compiler*, *debugger* dan *design* yang terintegrasi dalam sebuah aplikasi. Tujuan utama IDE adalah menyediakan semua fasilitas dan utilitas yang dibutuhkan untuk proses pembangunan program aplikasi (Alfa Satyaputra 2014).

GIT

Git adalah salah satu VCS (*Version Control System*) yang menerapkan DVCS (*Distributed Version Control System*) yangdiciptakan oleh Linus Torvalds. Git memiliki tugas mencatat setiap perubahan pada file proyek yang dikerjakan oleh banyak orang maupun sendiri(Putra, Rochimah, and Hakim 2014).

BUG

Bug adalah *error* yang pada aplikasi atau aplikasi yang tidak sesuai dengan kebutuhan atau *rule* yang telah dibuat dalam sebuah program aplikasi, oleh karena itu sebelum aplikasi disebarluaskan maka harus melalui tahapan pengujian supaya ketika aplikasi sudah disebarluaskan sudah tidak ada *bug* yang ditemukan yang membuat aplikasi *error*(Azhar 2016).

Integrasi

Menurut Kamus Besar Bahasa Indonesia Integrasi adalah pembauran atau penyatuan dari unsur-unsur yang berbeda sehingga menjadi kesatuan yang utuh atau bulat. Secara harfiah integrasi berlawanan dengan perpisahan, suatu sikap yang meletakkan tiap-tiap bidang dalam kotak-kotak yang berlainan. Integrasi memiliki sinonim dengan perpaduan, penyatuan, atau penggabungan, dari dua objek atau lebih.

Test Case

Test case adalah kondisi digunakan oleh penguji untuk memverifikasi aplikasi yang diuji berfungsi dengan benar atau tidak. Pengembangan *test case* dapat membantu untuk menemukan celah dan persyaratan yang dibutuhkan oleh aplikasi (Sneha and Malle 2017).

Penelitian Terkait

Penyusunan laporan penelitian ini sedikit banyak terinspirasi dan mengambil referensi dari penelitian-penelitian sebelumnya yang berkaitan dengan latar belakang masalah pada laporan penelitian ini. Berikut ini penelitian terdahulu yang berhubungan dengan laporan penelitian ini antara lain:

Penelitian yang dilakukan oleh (Zhang et al. 2014), tentang "*Softwaretesting*: *A case study of a small Norwegian software team*" yang membahas tentang pengujian secara otomatis, aplikasi berkembang lebih cepat dan semakin kompleks, pengujian aplikasi menjadi tolak ukur untuk memastikan bahwa aplikasi telah terbebas dari *bug* yang dapat merusak aplikasi. Namunsecara umum dianggap pengujian aplikasi secara manual tidak dapat memenuhi persyaratan pengujian yang lebih serius dari setiap *unit test* pada bagian aplikasi. Pengujian aplikasi yang tidak dilakukan dengan benar akan berdampak pada kualitas yang dapat membuat perusahaan kehilangan uang dan reputasi perusahaan.

Penelitian yang dilakukan oleh(Húska 2014), tentang "*Automated Testing of the Component-based Web Application User Interfaces*". Tujuan dari penelitian ini adalah mengeksplorasi serta menggambarkan metode modern dalam pengujian aplikasi *web* perusahaan. Hasil penelitianpengujian aplikasi otomatis akan dipilih dan dibandingkan dengan pengujian manual. Tujuan utama dari penelitian ini adalah menciptakan aplikasi pengujian secara otomatis yang terintegrasi dan berkenlajutkan.

Kontribusi dari penelitian ini adalah pembuatan *API*, karena untuk meningkatkan kualitas pengujian otomatisdiperlukan kemudahan dalam akses aplikasi pengujian. Hal ini juga membuktikan bahwa menggunakan *API* dapat menghemat jumlah baris kode untuk pengujian dalam beberapa kasus hingga 39%. Penggunaan *API* juga dapat meningkatkan kualitas pengujian.

Penelitian lain yang dilakukan oleh (Seth and Khare 2015) tentang "*ACI (Automated Continuous Integration) using Jenkins: Key for successful embedded software development*" *jenkins* memberikan banyak solusi yang lebih baik dalam menjalankan otomatisasi pada *continuous integration.* Dalam penelitian ini mempresentasikan implementasi dari program otomatisasi untuk membangun dan menjalankan *CTS* menggunakan *jenkins.*

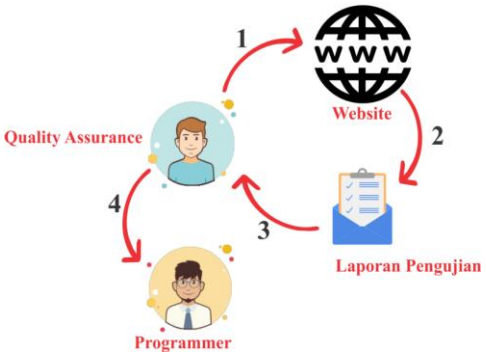
Hasil dan Pembahasan

Hasil Identifikasi dan Permasalahan

Pengujian aplikasi secara manual merupakan pengujian yang dilakukan dengan membuat *test cases* pengujian secara manual dan menjalankannya tanpa bantuan alat apa pun, semua proses pengujian dilakukan secara manual oleh manusia yang mencoba berbagai kombinasi penggunaan dan input serta membandingkan hasilnya dengan perilaku yang diharapkan dan mencatat hasil pengamatan mereka secara manual. Kekurangan dari pengujian aplikasi secara manual adalah tidak bisa melakukan proses pengujian secara berulang karena bisa menyebabkan

hasil yang tidak akurat. Eksekusi pengujian secara manual juga lebih lambat.

Pengujian aplikasi yang sedang berjalan pada PT. Eannovate Creative Agency masih dilakukan secara manual yang mengandalkan pengujian dari seorang *quality assurance* secara manual. Alur pengujian aplikasi yang sedang berjalan pada PT. Eannovate Creative Agency dapat dilihat pada gambar 3.



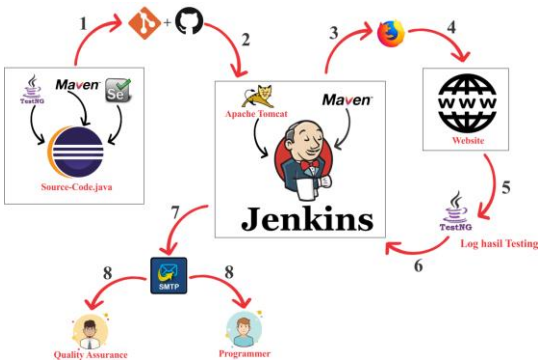
Gambar 3. Alur Pengujian Aplikasi Yang Sedang Berjalan.

Berdasarkan alur pengujian aplikasi secara manual dilakukan oleh *quality assurance* langsung berinteraksi dengan *website* yang diuji, berikut penjelasan alur pengujian aplikasi secara manual.

1. *Quality assurance* akan melakukan pengujian *website* secara manual sesuai dengan *test cases* yang telah dibuat.
2. Kemudian *quality assurance* mencatat semua hasil dari pengujian yang telah dilakukan.
3. Selanjutnya *quality assurance* membuat laporan hasil pengujian dalam bentuk dokumen laporan pengujian.
4. Terakhir *Quality assurance* akan menyerahkan dokumen laporan pengujian *website* kepada programmer.

Hasil Mendefinisikan Tujuan Untuk Solusi

Berdasarkan dari mendefinisikan tujuan untuk solusi permasalahan pada pengujian secara manual yang masih berjalan, diketahui bahwa pengujian secara manual yang berjalan masih belum optimal, maka diperlukan metode pengujian baru yang dapat memecahkan masalah dari pengujian manual. Metode pengujian yang bisa diterapkan adalah metode pengujian secara otomatis, dengan menggunakan metode ini pengujian aplikasi akan lebih efektif dan efisien. Hasil mendefinisikan tujuan untuk solusi dapat dilihat pada gambar 3.



Gambar 4. Desain Arsitektur Pengujian Aplikasi Otomatis

Berdasarkan gambar diatas telah menjawab dari kekurangan pengujian aplikasi secara manual, semua proses pengujian dan pelaporan hasil pengujian dilakukan secara otomatis. Hal ini membuat proses pengujian menjadi lebih cepat dan akurat.

Hasil Desain dan Pengembangan

Hasil desain dan pengembangan diambil dari rancangan desain dan pengembangan yang telah dibuat. Hasil ini memiliki dua proses tahapan sebelum pengujian otomatis dilakukan yaitu:

1. Tahap *Initialization testing* atau persiapan awal

```
Started by user dikakoko
Building in workspace C:\Users\Dika
Koko\.jenkins\workspace\TestRun
No credentials specified
> git.exe rev-parse --is-inside-work-tree #
timeout=10
Fetching changes from the remote Git repository
> git.exe config remote.origin.url
https://github.com/Fransiskusandika/automation-
test-with-selenium-testng-form-validation-.git #
timeout=10
Fetching upstream changes from
https://github.com/Fransiskusandika/automation-
test-with-selenium-testng-form-validation-.git
> git.exe --version # timeout=10
> git.exe fetch --tags --force --progress
https://github.com/Fransiskusandika/automation-
test-with-selenium-testng-form-validation-.git
+refs/heads/*:refs/remotes/origin/*
> git.exe rev-parse
"refs/remotes/origin/master^{commit}" # timeout=10
> git.exe rev-parse
"refs/remotes/origin/origin/master^{commit}" #
timeout=10
Checking out Revision
8facb520341ab26e2329e59356d48847be9b0975
(refs/remotes/origin/master)
> git.exe config core.sparsecheckout # timeout=10
> git.exe checkout -f
8facb520341ab26e2329e59356d48847be9b0975
Commit message: "cek mail no"
> git.exe rev-list --no-walk
8facb520341ab26e2329e59356d48847be9b0975 #
timeout=10
Checking for pre-build
Executing pre-build step
Checking if email needs to be generated
No emails were triggered.
Parsing POMs
```

Tahap *initialization testing* merupakan tahapan awal yang akan dilakukan sebelum eksekusi pengujian otomatis dilakukan, pada tahapan awal *jenkins* akan melakukan *git clone* versi terbaru dari program pengujian otomatis yang tersimpan di dalam *repositoryhttps://github.com/Fransiskusandika/automation-test-with-selenium testng-form-validation-.git*kemudian *jenkins* akan memanggil file *pom.xml* yang merupakan *file* eksekusi untuk melakukan pengujian *website* secara otomatis.

2. Tahap *testing* atau tahap pengujian *website* secara otomatis.

```
Running TestSuite
Testing is run on: Toko Aksessori Komputer
dan Gadget
Tests run: 14, Failures: 0, Errors: 0,
Skipped: 0, Time elapsed: 290.774 sec - in
TestSuite

Results :

Tests run: 14, Failures: 0, Errors: 0,
Skipped: 0

SNAPSHOT\com.seleniumtestngtest-0.0.1-
SNAPSHOT.pom
[INFO] ------------------------------------
-----------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------
-----------------------------------
[INFO] Total time:  05:11 min
[INFO] Finished at: 2019-07-
16T14:54:08+05:30
[INFO] ------------------------------------
-----------------------------------
```

Tahap *testing* merupakan tahapan yang akan dilakukan setelah proses *initialization* telah berhasil dilakukan, pada tahapan ini *jenkins* akan menjalankan *test cases* pengujian *website* secara otomatis.

3. Tahap *email notification* dari hasil pengujian aplikasi yang telah dilakukan.

```
messageContentType = text/html; charset=UTF-
8
Request made to attach build log
Request made to compress build log
  Collecting change authors...
    build: 53
Adding recipients from project recipient
list
Adding dikakoko with address
dikakoko04@gmail.com
Analyzing: dikakoko04@gmail.com
Looking for: dikakoko04@gmail.com
Successfully created MimeMessage
Sending email to: dikakoko04@gmail.com
fransadikawan@gmail.com
Finished: SUCCESS
```

Tahap *email notification* merupakan tahapan akhir dari proses pengujian *website* secara otomatis, pada tahapan ini akan dilakukan pengiriman hasil pengujian yang telah dilakukan pada tahapan sebelumnya.

Algoritm Pseudocode

Tabel 1. *Pseudocode* pengujian *website* secara otomatis.

Algoritma 1. Pengujian *Website* Otomatis

Input: T_c=*testcases*pengujian, T_p=jumlah *test cases* program *website*, E_t=*error test*, F_t=*finish test*
Output: H_t=*email nitification*

1: **for** $T_{c(1)}$= 0 to T_p **do**
2: $T_{c(1)} = T_{p+} T_p$
3: **end**
4: $H_t = F_t - E_t$
5: **end**

Hasil Demonstrasi

Hasil demonstrasi merupakan hasil dari pengujian yang dilakukan dengan melakukan perbandingan pengujian secara otomatis antara dua buah website yang sama, dengan kondisi *website* A merupakan *website* yang sudah jadi 100% memenuhi syarat dari program yang akan dibuat dan *website* B, merupakan *website* yang masih 80% telah memenuhi syarat dari program yang akan dibuat. Ringkasan hasil dari pengujian dapat dilihat pada tabel 3.

	Unit Test	*Duration Test*		*Error Test*	
		Web A	*Web B*	*Web A*	*Web B*
1.	*Title*	*0,29*	*0,29*	✓	✓
2.	*cekFormDataBarang*	*31*	*31*	✓	×
3.	*cekFormDataKasir*	*6.5*	*6.5*	✓	✓
4.	*cekFormDataKategori*	*4.3*	*4.3*	✓	×
5.	*cekFormDataSuplayer*	*7.7*	*7.7*	✓	✓
6.	*cekFormLogin*	*2.7*	*2.7*	✓	✓
7.	*cekLoginFailed*	*2.2*	*2.2*	✓	✓
8.	*cekLoginTrue*	*4.1*	*4.1*	✓	✓
9.	*cekLogout*	*3.6*	*3.6*	✓	✓
10.	*cekMenuDataBarang_CURD*	*9.6*	*9.6*	✓	✓
11.	*cekMenuDataKasir_CURD*	*6.2*	*6.2*	✓	✓
12	*cekMenuDataKategori_CURD*	*5.2*	*5.2*	✓	✓
13.	*cekMenuDataSuplayer_CURD*	*6.2*	*6.2*	✓	✓
14.	*cekTransaksiPembelian*	*14*	*14*	✓	×

Pengujian secara otomatis akan dilakukan menggunakan satu aplikasi pengujian. Hasil dari pengujian dari *website* Adan *website* B dapat dilihat pada gambar dibawah ini.

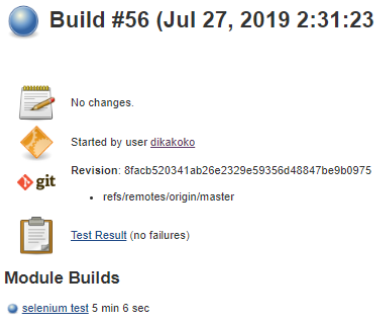
1. Hasil pengujian dari *website A.*

a. Hasil *console output*

```
Running TestSuite
Testing is run on: Toko Aksessori Komputer
dan Gadget
Tests run: 14, Failures: 0, Errors: 0,
Skipped: 0, Time elapsed: 284.481 sec - in
TestSuite
Results :
Tests run: 14, Failures: 0, Errors: 0,
Skipped: 0
```

Hasil diatas menunjukkan hasil dari jumlah *unit test* : 14, *failures test* : 0, *errors test* : 0, *skipped test* : 0dan waktu : 284.481 detik yang dibutuhkan dalam pengujian *website* secara otomatis.

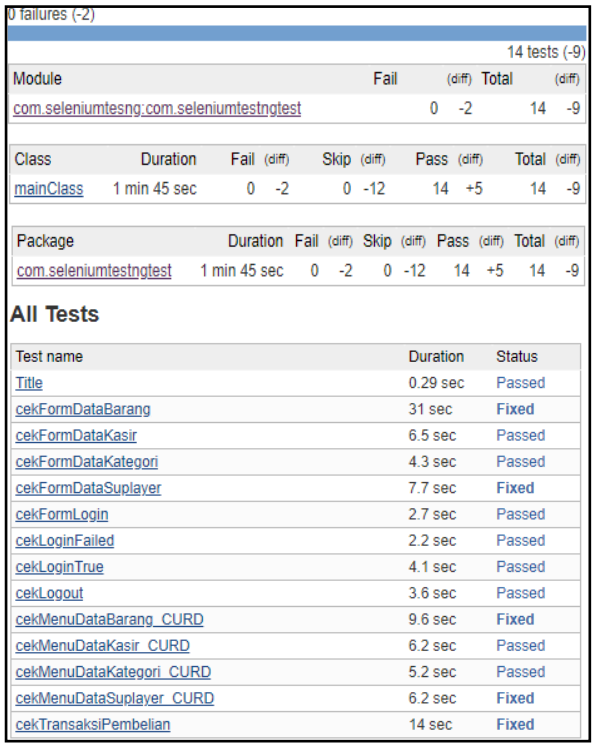
b. Status Pengujian



Gambar 5. Status Pengujian *Website A.*

Hasil diatas menunjukkan hasil dari status pengujian *website* secara otomatis, gambar diatas menunjukkan bahwa pengujian *website* yang telah dilakukan telah berhasil berjalan 100% benar tanpa ada *error* pada program. Warna biru menunjukkan status pengujian *passed*.

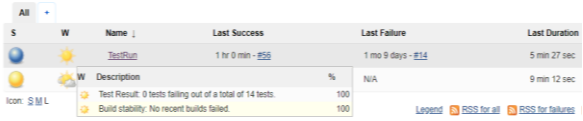
c. Test Result



Gambar 6. *Test Result Website A.*

Hasil diatas menunjukkan hasil dari *test result* pengujian *website* secara otomatis, pada gambar diatas menunjukkan bahwa tidak ditemukan *failures test* dari *website* yang telah dilakukan pengujian.

d. Summary Report



Gambar 7. *Summary Report Website A.*

Hasil diatas menunjukkan hasil dari *summary report* dan jumlah pengujian yang pernah dilakukan dari pengujian *website* secara otomatis, pada gambar diatas menunjukkan *health icons sunny* yang memiliki arti bahwa pengujian 80% telah dilakukan dengan baik dan tidak ditemukan *errors* pada *website* yang diuji.

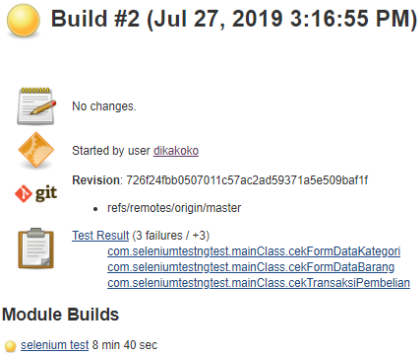
2. Hasil pengujian dari *Website B.*

a. Hasil *Console Output*

```
Running TestSuite
Testing is run on: Toko Aksessori
Komputer dan Gadget
Tests run: 14, Failures: 3, Errors: 0,
Skipped: 0, Time elapsed: 441.883 sec
<<< FAILURE! - in TestSuite
cekFormDataKategori(com.seleniumtestngt
est.mainClass)  Time elapsed: 37.545
sec  <<< FAILURE!
org.openqa.selenium.NoSuchElementExcept
ion:
Unable to locate element:
{"method":"css
selector","selector":"div[class='mssgBo
x']"}
Command duration or timeout: 20.04
seconds
```

Hasil diatas menunjukkan hasil dari jumlah *unit test* : 14, *failures test* : 3, *errors test* : 0, *skipped test* : 0 dan waktu : 441.883detik yang dibutuhkan dalam pengujian *website* secara otomatis serta menunjukkan letak pada *error test case* pada *website* yang diuji.

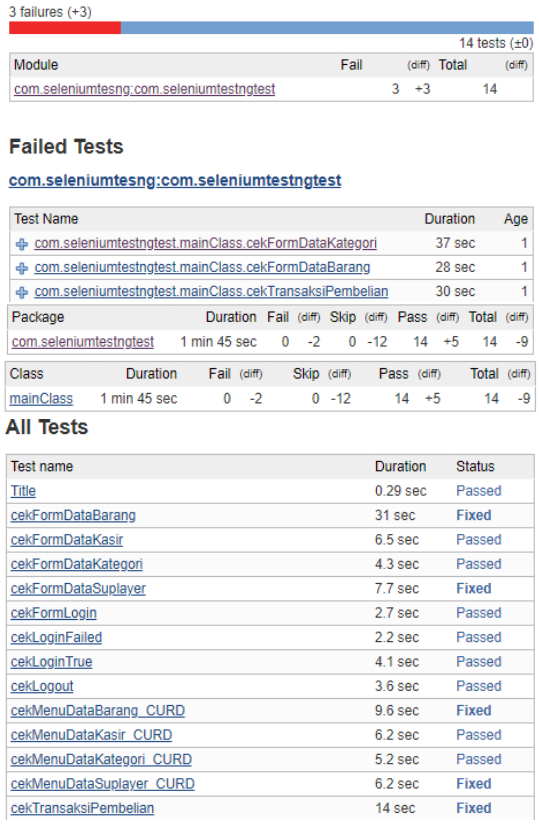
b. Status Pengujian



Gambar 13. Status Pengujian Website B.

Hasil diatas menunjukkan hasil dari status pengujian *website* secara otomatis, gambar diatas menunjukkan bahwa pengujian *website* yang telah dilakukan telah ditemukan *error* pada *form validation* pada menu kategori dan menu barang pada bagian *create* barang dan *update* barang. Menunjukkan oren yang memiliki arti status pengujian *unstable*.

c. Test Result



3 failures (+3)

14 tests (±0)

Module	Fail	(diff)	Total	(diff)
com.seleniumtesng:com.seleniumtestngtest	3	+3	14	

Failed Tests

com.seleniumtesng:com.seleniumtestngtest

Test Name	Duration	Age
com.seleniumtestngtest.mainClass.cekFormDataKategori	37 sec	1
com.seleniumtestngtest.mainClass.cekFormDataBarang	28 sec	1
com.seleniumtestngtest.mainClass.cekTransaksiPembelian	30 sec	1

Package	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
com.seleniumtestngtest	1 min 45 sec	0	-2	0	-12	14	+5	14	-9

Class	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
mainClass	1 min 45 sec	0	-2	0	-12	14	+5	14	-9

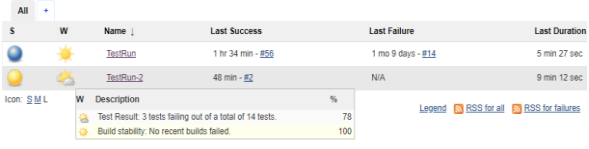
All Tests

Test name	Duration	Status
Title	0.29 sec	Passed
cekFormDataBarang	31 sec	Fixed
cekFormDataKasir	6.5 sec	Passed
cekFormDataKategori	4.3 sec	Passed
cekFormDataSuplayer	7.7 sec	Fixed
cekFormLogin	2.7 sec	Passed
cekLoginFailed	2.2 sec	Passed
cekLoginTrue	4.1 sec	Passed
cekLogout	3.6 sec	Passed
cekMenuDataBarang_CURD	9.6 sec	Fixed
cekMenuDataKasir_CURD	6.2 sec	Passed
cekMenuDataKategori_CURD	5.2 sec	Passed
cekMenuDataSuplayer_CURD	6.2 sec	Fixed
cekTransaksiPembelian	14 sec	Fixed

Gambar 8. *Test Result Website B.*

Hasil diatas menunjukkan hasil dari *test result* pengujian *website* secara otomatis, pada gambar diatas menunjukkan bahwa ditemukan 3 *failures test* dari *website* yang telah dilakukan pengujian.

d. Summary Report



S	W	Name	Last Success	Last Failure	Last Duration
		TestRun	1 hr 34 min - #56	1 mo 9 days - #14	5 min 27 sec
		TestRun-2	48 min - #2	N/A	9 min 12 sec

Icon: S M L

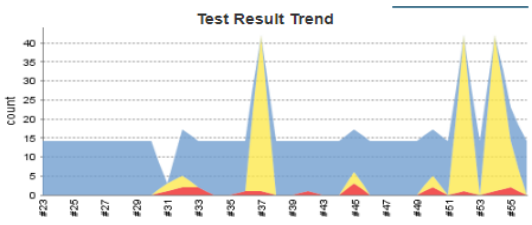
W	Description	%
	Test Result: 3 tests failing out of a total of 14 tests.	78
	Build stability: No recent builds failed.	100

Legend RSS for all RSS for failures

Gambar 9. *Summary Report Website B.*

Hasil diatas menunjukkan hasil dari *summary report* dan jumlah pengujian yang pernah dilakukan dari pengujian *website* secara otomatis, pada gambar diatas menunjukkan *health icons partially sunny* yang memiliki arti bahwa pengujian 60%-80% telah ditemukan beberapa *errors* pada *website* yang telah diuji.

3. Grafik hasil dari setiap tindakan pengujian aplikasi.



Berdasarkan gambar diatas setiap tindakan pengujian dan hasil pengujian *error*, *unstable* dan *success* akan dicatat dalam bentuk grafik sehingga dapat memudahkan *project manager* dan pemimpin perusahaan dalam melakukan *tracking* dari perkembangan aplikasi.

Hasil Evaluasi

Pengujian keakuratan atau *accuracy test* adalah serangkaian kemampuan untuk membedakan hasil pengujian dari *website* yang 100% sudah memenuhi standar dengan *website* yang masih 80% berdasarkan skenario. Nilai keakuratan tes dihitung dengan melihat proporsi benar positif (*true positive*) dan negatif sejati (*true negative*) dalam semua kasus yang dievaluasi. Secara matematis pengujian akurasi tersebut dapat dinyatakan sebagai:

$$Akurasi = \frac{3+0}{3+0+11+0} = \frac{3}{14} = 0.21 \quad (1)$$

$$\boldsymbol{Sensitivitas} = \frac{3}{3+11} = \frac{3}{14} = 0.21 \quad (2)$$

$$\boldsymbol{Spesifisitas} = \frac{0}{0+0} = \frac{0}{0} = 0 \quad (3)$$

Kesimpulan

Kesimpulan dari metode pengujian otomatis adalah mengurangi kesalahan dari manusia dalam melakukan pengujian secara manual menggunakan metode pengujian secara otomatis, meningkatkan efektivitas dan efisiensi pengujian aplikasi, melakukan pengujian berdasarkan *unit test* untuk menambah akurasi dan efektivitas pengujian aplikasi.

REFERENSI

Alfa Satyaputra, M.S.E. 2014. *Java for Beginners with Eclipse 4.2 Juno*. Elex Media Komputindo. https://books.google.co.id/books?id=8NhMDwAAQBAJ.

Azasoo, Julius Quarshie, and Kwame Osei Boateng. 2015. “A Retrofit Design

Science Methodology for Smart Metering Design in Developing Countries." In *2015 15th International Conference on Computational Science and Its Applications*, Banff, AB, Canada: IEEE, 1–7. http://ieeexplore.ieee.org/document/7166156/ (July 26, 2019).

Azhar, Nur Fajri. 2016. "Memprediksi Waktu Memperbaiki Bug dari Laporan Bug Menggunakan Klasifikasi Random Forest." 11(1): 9.

Barus, Arlinta Christy, Dian Ira Putri Hutasoit, Joel Hunter Siringoringo, and Yusfi Apriyanti Siahaan. 2015. "White Box Testing Tool Prototype Development." In *2015 International Conference on Electrical Engineering and Informatics (ICEEI)*, Denpasar, Bali, Indonesia: IEEE, 417–22. http://ieeexplore.ieee.org/document/7352537/ (July 24, 2019).

Bentes, Larissa, Herbert Rocha, Eduardo Valentin, and Raimundo Barreto. 2016. "JFORTES: Java Formal Unit TESt Generation." In *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*, João Pessoa, Paraíba, Brazil: IEEE, 16–23. http://ieeexplore.ieee.org/document/7828280/ (June 25, 2019).

Bhateja, Neha. 2015. "A Study on Various Software Automation Testing Tools." *International Journal of Advanced Research in Computer Science and Software Engineering*: 3.

Chaini, Hari Sankar, and Sateesh Kumar Pradhan. 2015. "Test Script Execution and Effective Result Analysis in Hybrid Test Automation Framework." In *2015 International Conference on Advances in Computer Engineering and Applications*, Ghaziabad, India: IEEE, 214–17. http://ieeexplore.ieee.org/document/7164698/ (June 10, 2019).

Garousi, Vahid, and Erdem Yildirim. 2018. "Introducing Automated GUI Testing and Observing Its Benefits: An Industrial Case Study in the Context of Law-Practice Management Software." In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Vasteras: IEEE, 138–45. https://ieeexplore.ieee.org/document/8411745/ (June 10, 2019).

Húska, Juraj. 2014. "Automated Testing of the Component-Based Web Application User Interfaces." *2014*: 90.

Klammer, Claus, and Rudolf Ramler. 2017. "A Journey from Manual Testing to Automated Test Generation in an Industry Project." In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Prague, Czech Republic: IEEE, 591–92. http://ieeexplore.ieee.org/document/8004387/ (June 9, 2019).

Laukkanen, Eero, and Mika Mantyla. 2015. "Build Waiting Time in Continuous Integration -- An Initial Interdisciplinary Literature Review." In *2015 IEEE/ACM 2nd International Workshop on Rapid Continuous Software Engineering*, Florence, Italy: IEEE, 1–4. http://ieeexplore.ieee.org/document/7167165/ (July 25, 2019).

O'Broin, Caoimh. 2015. "The Impact of Test Automation on Software Testers." *2015*: 74.

Polkhovskiy, Denis. 2016. "Comparison between Continuous Integration Tools." *2016*: 59.

Putra, Bayu Aji Mahendra, Siti Rochimah, and Jl Arief Rahman Hakim. 2014. "Gitcode: Manajemen Tempat Penyimpanan Kode Sumber Berbasis Git untuk Lingkungan Perkuliahan." 2(1): 3.

Rajasekhar, P, and Dr R Mahammad Shafi. 2014. "Agile Software Development and Testing: Approach and Challenges in Advanced Distributed Systems." : 5.

Ramya, Paruchuri, Vemuri Sindhura, and P. Vidya Sagar. 2017. "Testing Using Selenium Web Driver." In *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, Coimbatore: IEEE, 1–7. http://ieeexplore.ieee.org/document/8117878/ (June 16, 2019).

Seth, Nikita, and Rishi Khare. 2015. "ACI (Automated Continuous Integration) Using Jenkins: Key for Successful Embedded Software Development." In *2015 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS)*, Chandigarh, India: IEEE, 1–6. http://ieeexplore.ieee.org/document/7453279/ (June 21, 2019).

Sharma, R M. 2014. "Quantitative Analysis of Automation and Manual Testing." 4(1): 6.

Sharma, Sachin. 2012. "Study and Analysis of Automation Testing Techniques." 3(12): 8.

Sneha, Karuturi, and Gowda M Malle. 2017. "Research on Software Testing Techniques and Software Automation Testing Tools." In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, Chennai: IEEE, 77–81. https://ieeexplore.ieee.org/document/8389562/ (June 11, 2019).

Soni, Mitesh. 2015. "End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery." In *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, Bangalore, India: IEEE, 85–89. http://ieeexplore.ieee.org/document/7436936/ (June 10, 2019).

Vasˇkova, Lenka. 2014. "Advanced Methods for Integration Testing." *2014*: 52.

Vilkomir, Sergiy, John Baptista, and Gourav Das. 2017. "Using MC/DC as a Black-Box Testing Technique." In *2017 IEEE 28th Annual Software Technology Conference (STC)*, Gaithersburg, MD: IEEE, 1–7. http://ieeexplore.ieee.org/document/8234460/ (July 24, 2019).

Xiong Zhen-hai, and Yang Yong-zhi. 2014. "Automatic Updating Method Based on Maven." In *2014 9th International Conference on Computer Science & Education*, Vancouver, BC, Canada: IEEE, 1074–77. http://ieeexplore.ieee.org/document/6926628/ (June 25, 2019).

Zhang, Lin et al. 2014. "Software Testing: A Case Study of a Small Norwegian Software Team." *2014*: 74.