# Survei Strategi Pengujian Software Menggunakan Metode *Systematic Literature Review*

Munirul Huda[1,*], Muhammad Ainul Yaqin[2], Rashad Fathin Kurniawan[3], Moch Wahyu Fitra Choiri[4]

Program Studi Teknik Informatika, Universitas Islam Negeri Maulana Malik Ibrahim, Indonesia

[1]19650069@student.uin-malang.ac.id; [2]yaqinov@ti.uin-malang.ac.id; [3]19650086@student.uin-malang.ac.id; [4]19650039@student.uin-malang.ac.id;

* corresponding author

INFO ARTIKEL

**Sejarah Artikel**
Diterima: 19 Juni 2021
Direvisi: 5 Juli 2021
Diterbitkan: 30 April 2022

**Kata Kunci**
Strategi,
Pengujian,
*Software*,
*Systematic Literature Review*

ABSTRAK

Penelitian ini bertujuan untuk mengetahui berbagai macam strategi pengujian *software* yang ada, menganalisis dan membandingkannya untuk mengetahui keefektifan dari masing-masing strategi. Data-data yang digunakan dalam penelitian ini antara lain adalah paper dan artikel yang diseleksi berdasarkan kriteria yang telah ditentukan. Metode yang digunakan pada penelitian ini adalah *Systematic Literatur Review* (SLR), dimana literatur-literatur yang membahas tentang strategi pengujian *software* disurvei dan dibandingkan. Literatur-literatur yang membahas strategi pengujian *software* dikumpulkan, disurvei dan diulas sesuai dengan tahapan-tahapan metode SLR. Hasil dari penelitian ini adalah penemuan 4 klasifikasi strategi pengujian *software*, dimana masing-masing dari 4 klasifikasi tersebut dibagi lagi oleh beberapa tipe. Kesimpulan yang didapatkan adalah strategi pengujian *software* yang biasanya paling sering digunakan oleh *developer* adalah *unit testing*, dimana pengujian tersebut dilakukan untuk menemukan *defect* pada tingkat modul atau komponen.

## PENDAHULUAN

Pengujian perangkat lunak dibutuhkan dalam melakukan konfirmasi dan validasi dari hasil pengembangan suatu perangkat lunak. Pengujian dilakukan secara sistematis dengan menggunakan sejumlah strategi pengujian perangkat lunak. Pengujian perangkat lunak digunakan untuk mencegah peluang kesalahan yang dibuat oleh manusia dalam suatu system atau dengan kata lain untuk menghindari perangkat lunak yang ada dari cacat. Pengujian perangkat lunak adalah metode untuk mengkonfirmasi apakah perangkat lunak sesuai dengan persyaratan yang ada, sekaligus untuk memastikan bahwa perangkat lunak bebas dari *error* [1]. Pelaksanaan pengujian perangkat lunak dilakukan menggunakan komponen manual maupun otomatis untuk mengevaluasi satu atau lebih properti yang dipilih. Pengujian perangkat lunak diperlukan untuk menunjukkan cacat dan kesalahan yang dibuat selama fase pengembangan.

Strategi pengujian perangkat lunak merupakan garis besar yang mendeskripsikan pendekatan pengujian *software development life cycle*, yang dimana strategi pengujian perangkat lunak mencakup hal-hal yang perlu diketahui dari sebuah proses pengujian perangkat lunak. Hal-hal tersebut seperti tujuan, total waktu, sumber daya dan lingkungan dari pengujian perangkat lunak tersebut [2]. Strategi pengujian perangkat lunak memastikan kelancaran pengembangan perangkat lunak, memastikan agar perangkat lunak yang ada sesuai dengan persyaratan dan juga memastikan agar perangkat lunak tersebut bebas dari cacat. Strategi Pengujian Software merupakan salah satu hal penting dalam pengembangan aplikasi / *software development life cycle*. Pada masa ini strategi pengujian sering dilakukan secara kurang formal dan tidak dapat

dilakukan secara efektif dikarenakan pengembang tidak mengetahui strategi mana yang optimal dalam pengujian *software* [2]. Sehingga diperlukan pengetahuan mengenai keefektifan strategi pengujian *software* agar pengembang dapat melakukan pengujian software secara optimal dan efektif.

Tujuan dari penelitian ini adalah mengklasifikasikan Strategi pengujian *software* berdasarkan kelebihan dan kekurangan. Sehingga, diharapkan dapat melakukan proses pengujian *software* secara optimal dan dapat menghasilkan *software* sesuai yang diharapkan. Oleh karena itu penelitian ini dilakukan dengan cara mengidentifikasi dan meninjau macam macam strategi pengujian *software* dengan menggunakan metode *Systematic Literature Review* (SLR). Dari survei yang dilakukan, secara garis besar strategi pengujian *software* terbagi menjadi 4 tipe, yang terdiri dari *Unit testing, Acceptance / Validation testing, Integration testing,* dan *System testing* [3]. Untuk *unit testing,* dikategorikan menjadi 3 tipe, yaitu *functional testing, heuristic / intuitive testing,* dan *structural testing* [4]. Untuk *integration testing,* dibedakan menjadi 2 macam, yaitu *bottom-up integration* dan *top-down integration* [5]. Untuk *acceptance testing,* dibagi menjadi 4 tipe, diantaranya adalah *user acceptance testing, operational acceptance testing, alpha testing* & *beta testing,* dan *contact and regulation acceptance testing* [6]. Untuk *system testing,* dibagi menjadi 4 tipe, diantaranya adalah *recovery testing, graphical user interface testing, security testing* dan *compability testing* [7]. Dari strategi pengujian perangkat lunak tesebut, masing-masing akan memiliki nilai kelebihan dan kekurangannya sendiri. Oleh karena itu, kelebihan dan kekurangan dari masing-masing strategi pengujian perangkat lunak tersebut akan dibahas dalam paper ini.

Dari survei ini, dihasilkan data mengenai berbagai macam strategi pengujian perangkat lunak. Maka dari itu, perlu adanya sebuah pengkajian dalam bentuk klasifikasi untuk memaparkan kelebihan dan kekurangan dari setiap strategi pengujian perangkat lunak. Meskipun hal tersebut diperlukan, pengklasifikasian tersebut belum ada dicantumkan pada paper dari literatur-literatur yang sudah dikumpulkan. Oleh karena itu, pengklasifikasian tersebut disediakan di dalam paper ini sehingga pembaca dapat memahami subtansi dari setiap tahap strategi pengujian perangkat lunak dengan lebih mudah.

## METODE

Survei ini menggunakan metode penelitian *Systematic Literature Review* (SLR). *Systematic literature review* / SLR atau yang sering disebut sebagai tinjauan pustaka sistematis dalam bahasa indonesia adalah metode literature review yang mengenali, menilai, dan menginterpretasi seluruh temuan-temuan yang dikumpulkan pada satu topik penelitian, untuk menjawab pertanyaan penelitian / research question yang telah ditetapkan pada awal penelitian [8]. Secara umum, terdapat 3 tahapan besar dalam melakukan SLR, diantaranya adalah *Planning, Conducting dan Reporting* seperti pada Gambar 1


Gambar 1. Tahapan SLR

### *Planning*

*Research Question* (RQ) adalah bagian awal dan dasar berjalannya SLR. RQ memandu proses pencarian dan ekstraksi dari literatur yang akan dikumpulkan. Analisis dan hasil survei, sebagai hasil dari SLR, adalah jawaban dari RQ yang sudah ditentukan. Formulasi RQ harus didasarkan pada 5 materi yang dikenal dengan PICOC seperti pada Tabel 1:

Tabel 1. PICOC

| | **Pengertian** | **Penelitian** |
|---|---|---|
| **P** | *Population* (target investigasi) | Strategi pengujian *software* |
| **I** | *Intervention* (aspek detail dari investigasi) | Pembagian dari tipe-tipe strategi pengujian *software* |
| **C** | *Comparison* (perbandingan *intervention*) | Membandingkan tiap pembagian dari strategi pengujian *software* |
| **O** | *Outcomes* (hasil investigasi) | Keefektifan dari tiap strategi pengujian *software* |
| **C** | *Context* (setting dan lingkungan dari investigasi) | Studi di bidang industry dan akademik |

Pada Tabel 1 yang berisikan PICOC, ditemukan dan diputuskan bahwa RQ dari penelitian ini adalah sebagai berikut :

- Jurnal dan literatur apa yang paling relevan dengan strategi pengujian perangkat lunak?
- Bagaimana keefektifan dari masing-masing jenis strategi pengujian perangkat lunak?

***Conducting***

Tahapan *conduting* adalah tahapan pelaksanaan SLR, dimana seharusnya sesuai dengan protokol SLR yang telah ditentukan, dimulai penentuan *keyword* dari pencarian literatur (search string) yang basisnya adalah PICOC yang telah didesain sebelumnya. Pemahaman terhadap sinonim dan alternatif kata yang digunakan juga akan menentukan akurasi pencarian literatur yang akan dicari. *Keyword* yang digunakan dalam pencarian literatur penelitian ini adalah sebagai berikut :

*(software)*
*AND*
*(testing* OR measuring OR examining)*
*AND*
*(strategies* OR approach OR procedure)*

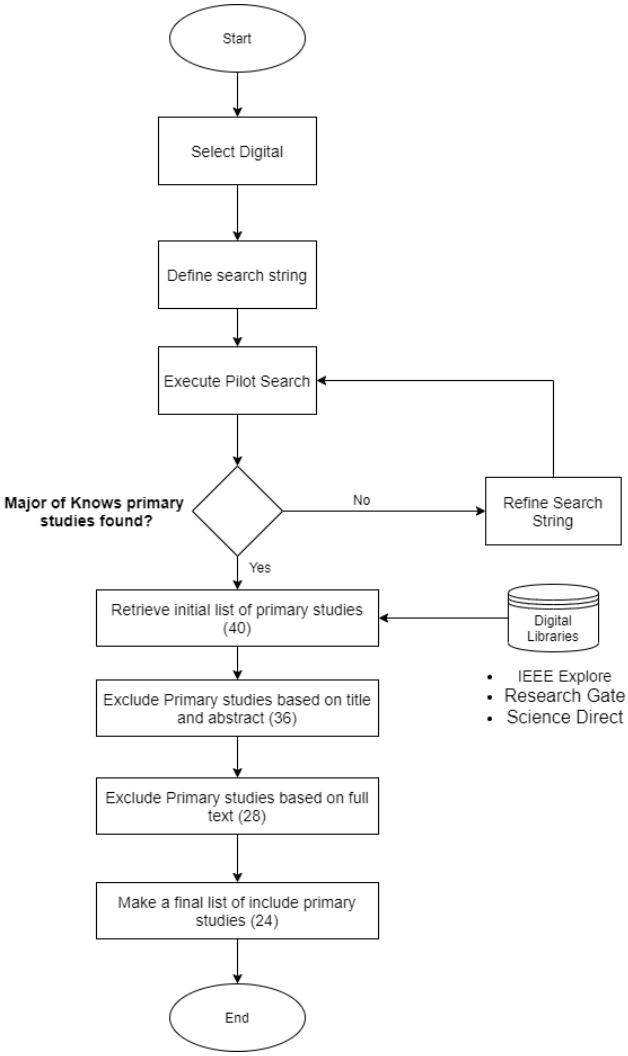
Selanjutnya tentukan sumber dari pencarian literatur, dimana sumber pencarian literatur kami didapatkan menggunakan *digital library* berupa google scholar. Sumber pencarian literature kami meliputi IEEE Xplore, ResearchGate dan ScienceDirect. Setelah literatur didapat, tentukan literatur yang sesuai dengan cara menyeleksi literatur yang ada menggunakan kriteria inklusi dan eksklusi. Berikut ini adalah table kriteria inklusi dan eksklusi dari penelitian ini:

Tabel 2. Kriteria Inklusi dan Eksklusi Literatur

| **Kriteria Inklusi** | **Kriteria Eksklusi** |
|---|---|
| Membahas tentang strategi pengujian *software*, dimana cakupan literatur tidak terlalu luas | Cakupan literatur terlalu luas |
| Literatur ditulis menggunakan Bahasa Inggris atau Bahasa Indonesia | Literatur tidak ditulis menggunakan Bahasa Inggris atau Bahasa Indonesia |
| Membahas implementasi dari strategi pengujian *software* | Literatur yang ada dipublikasikan sebelum tahun 1990 |

Pada Tabel 2 dijelaskan bahwa paper memiliki kirteria kriteria dalam pengambilan referensi diantaranya adalah paper membahas mengenai strategi pengujian *software* yang pada paper tersebut tidak membahas cakupan yang terlalu luas, literatur menggunakan bahsa inggris atau bahasa indonesia dan paper tersebut menjelesakan mengenai impelemtasi dari strategi pengujian *software.*

Berikut adalah *Flowchart* dari tahap *conducting* yang berisikan pencarian jurnal untuk direview:




Gambar 2. Flowchart Conducting

Dari Gambar 2 dijelaskan bahwa pencarian literatur dimulai dengan penentuan *keyword*, lalu diseleksi berdasarkan kriteria inklusi dan eksklusi yang ditentukan. Literatur yang didapat pada tahap pertama sejumlah 40, kemudian dengan memperhatikan kriteria yang ditentukan menjadi 36 dan 28, lalu dipersempit lagi sehingga jumlah final literatur yang digunakan adalah 24. Literatur-literatur tersebut didapatkan melalui IEEE Explore, Research Gate dan Science Direct.

### *Reporting*

*Reporting* adalah tahapan penulisan hasil SLR yang dilakukan. Setelah literatur didapatkan, literatur-literatur yang ada diseleksi berdasarkan kriteria eksklusi dan inklusi, berikut adalah hasil *reporting* yang tertera pada Tabel 3:

Tabel 3. Literatur yang Diseleksi

| | IEEE | Research Gate | ScienceDirect |
|---|---|---|---|
| **Literatur yang didapat** | 11 | 16 | 13 |
| **Literatur setelah dianalisis** | 7 | 10 | 7 |

## Kebaruan Ilmiah

Sebagai kontribusi ilmiah dan pembeda dengan hasil survei pada penelitian yang ada sebelumnya, kami memberikan kebaruan ilmiah yaitu hasil klasifikasi strategi pengujian *software* berdasarkan kelebihan dan kekurangan sehingga pengembang mengetahui strategi pengujian *software* yang paling efisien (lihat Tabel 6,7,8 dan 9).

# HASIL DAN PEMBAHASAN

## Klasifikasi Jurnal

Tabel 4. Klasifikasi Jurnal

| Tahun | Jumlah |
|---|---|
| 1990 | 1 |
| 1995 | 1 |
| 2000 | 1 |
| 2003 | 1 |
| 2004 | 2 |
| 2006 | 3 |
| 2008 | 2 |
| 2010 | 2 |
| 2011 | 1 |
| 2012 | 3 |
| 2013 | 1 |
| 2014 | 1 |
| 2016 | 1 |
| 2015 | 3 |
| 2018 | 1 |

Pada Tabel 4 menjelaskan klasifikasi jurnal berdasarkan tahunnya dimana data yang diperoleh jurnal yang diambil mulai tahun 1990 hingga tahun 2018, dimana pada tahun 2015 dan tahun 2012 terdapat 3 jurnal yang relevan dengan pembahasan paper ini.

## Klasifikasi Jurnal yang Paling Relevan dengan Strategi Pengujian *Software*

Jurnal yang paling relevan dengan strategi pengujian perangkat lunak yang tercantum dibawah digunakan sebagai data primer, dimana sisa literatur yang dikumpulkan digunakan menjadi data sekunder.

Tabel 5. Klasifikasi Jurnal yang Paling Relevan Dengan Strategi Pengujian *Software*

| Paper | Penulis | Klasifikasi | Keterangan |
|---|---|---|---|
| [9] | Michael Olan | *Unit Testing* | Berisikan tentang Unit Testing beserta impelementasinya |

| | | | |
|---|---|---|---|
| [10] | Shruti N. Pardeshi | *Testing Strategies* | Adanya penjelasan mengenai strategi pengujian serta pendekatannya. |
| [11] | Lionel Briand and Yvan Labiche | *System Testing* | Berisi Tentang salah satu pendeketan strategi pengujian *software* yaitu system testing |
| [12] | Roy W. Miller and Christopher T. Collins | *Acceptance Testing* | Paper ini membahas tentang acceptance testing dimana acceptence testing merupakan salah satu pendekatan strategi pengujian *software* dan di paper ini membahas tentang pengimplementasian pada program. |
| [6] | Che Ku Nuraini Che Ku Mohda dan Faaizah Shahbodina | *Alpha Testing & Beta Testing* | Pada paper ini menjelaskan implementasi dari Alpha testing & Beta Testing. |
| [2] | Abhijit A. Sawant, Pranit H. Bari and P. M. Chawan | Strategi Pengujian *Software* | Pada paper ini berisikan detail mengenai strategi pengujian *software* beserta pendektannya. |
| [4] | Ermira Daka and Gordon Fraser | *Unit Testing* | Pada paper ini berisikan survei mengenai unit testing pada kalangan *developer*, serta terdapat beberapa kelebihan mengenai unit testing. |
| [13] | Sahil Batra dan Dr. Rahul Rishi | *Testing Strategies* | Paper ini berisikan tahapan dalam mengimprovisasi kualitas dari software menggunakan pengujian strategi. |
| [14] | Hareton K. N. Leung dan Lee White | *Integration Testing* | Terdapat bahasan mengenai integration testing, kelebihan, kekurangan integration testing dan implementasinya. |
| [15] | Victor R Basili | *Comparation* | Berisikan perbandingan keefektifitasan pada strategi pengujian software |

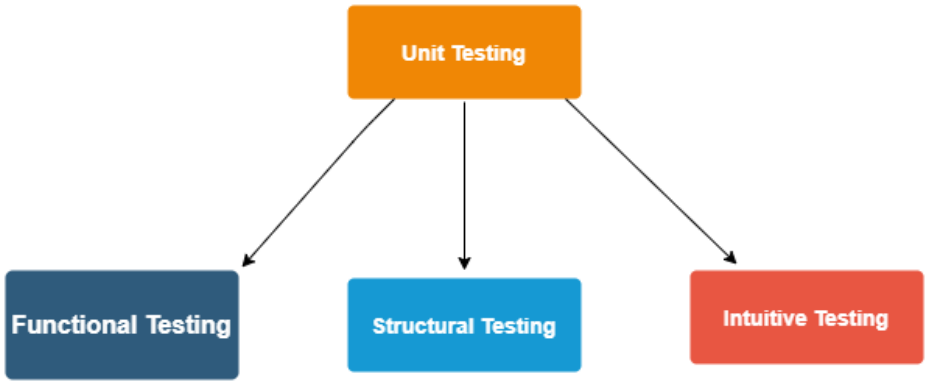
Berdasarkan Tabel 5 paper yang paling banyak ditemukan adalah paper yang membahas mengenai *Unit testing*. Paper tersebut dikutip melalui IEEE.

## Klasifikasi Strategi Pengujian *Software*

Secara garis besar *software* strategi terbagi menjadi 4 tipe yaitu: i*ntegration testing*, *unit testing*, *acceptance testing*, *system testing.* Strategi pengujian perangkat lunak memiliki tahapan-tahapan mengenai beberapa hal yang harus dilakukan. Berikut adalah klasifikasi strategi testing *software* seperti pada Gambar 3.


Gambar 3. Hirarki Strategi Pengujian Perangkat Lunak

***Unit testing***




Gambar 4. Hirarki *Unit Testing*

Unit adalah modul / kumpulan baris kode terkecil yang dapat diuji. *Unit testing* merupakan salah satu level pengujian yang dimana kumpulan pengujian-pengujian tersebut membuat gambaran besar dari pengujian sistem. *Unit testing* melengkapi integrasi dan pengujian tingkat sistem. *Unit testing* juga harus melengkapi tinjauan kode dan penelusuran. *Unit testing* umumnya dilihat sebagai kelas pengujian *white box*, agar unit testing bisa untuk melihat dan mengevaluasi kode yang diterapkan, daripada harus mengevaluasi kesesuaian dengan beberapa set requirement. Teknik-teknik unit testing adalah Sejumlah teknik pengujian yang efektif dapat digunakan dalam *unit testing*. Teknik pengujian itu secara luas menjadi 3 jenis Seperti pada Gambar 4.

***a. Functional Testing***

*Functional testing* merupakan jenis pengujian *software* yang memvalidasi sistem *software* terhadap persyaratan / spesifikasi fungsional. Pengujian fungsional melibatkan pengujian *black box* dan tidak memperhatikan *source code* aplikasi. Pengujian ini memeriksa *user interface*,

API, *database*, Keamanan, komunikasi klien / *server*, dan fungsionalitas lain dari aplikasi yang sedang diuji. Pengujian ini dapat dilakukan melalui manual atau dilakukan secara otomatisasi.

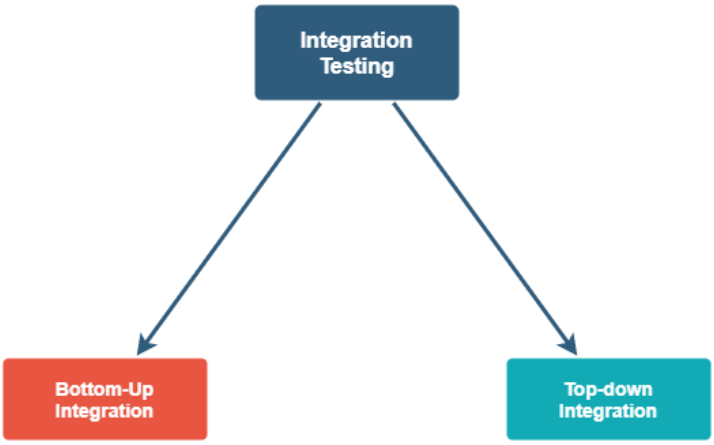
***b. Structural Testing***

*Structural testing* merupakan jenis pengujian *software* yang menggunakan desain internal *software* untuk pengujian atau dengan kata lain pengujian *software* yang dilakukan oleh tim yang mengetahui tahap pengembangan *software*. *Structural Testing* didesain untuk memverivikasi bahwa sistem pengembangan dan program program bekerja. Obejektifnya adalah untuk meyakinkan bahwa desain produk berstruktur kuat dan berfungsi dengan baik.

***c. Heuristic / Intuitive Testing***

Teknik jenis ini adalah teknik pengujian berbasis pengalaman dimana analis tes menggunakan pengalamannya untuk menebak area aplikasi yang bermasalah. Teknik ini membutuhkan penguji yang terampil dan berpengalaman. Ini adalah jenis teknik Pengujian *Black-Box* dan dapat dilihat sebagai pendekatan tidak terstruktur untuk pengujian *software*.

***Integration Testing***




Gambar 5. Hirarki *Integration Testing*

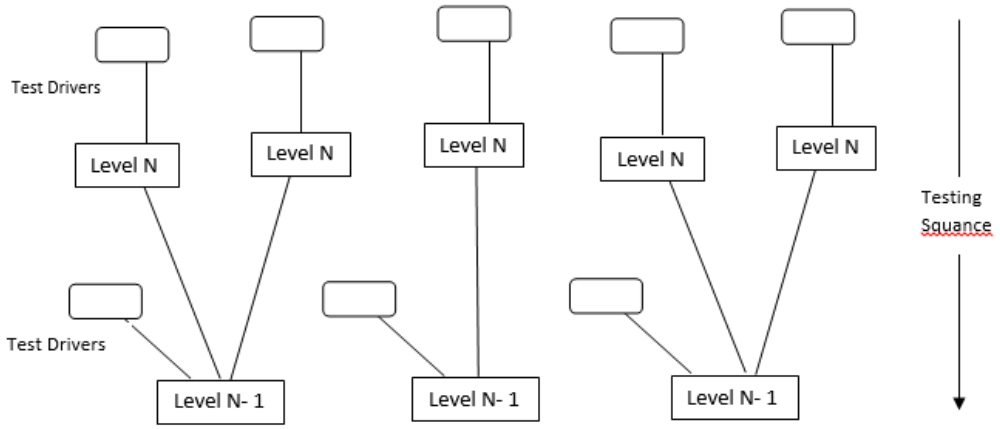
*Integration* adalah pengujian yang diterapkan ketika semua bagian modul digabungkan untuk membentuk *system*. Pengujian dilakukan di tingkat modul, bukan di tingkat statement seperti pada unit testing. *Integration* testing menekankan interaksi antara modul dan juga interfacenya. Integration testing dibagi menjadi 2 yaitu *Bottom-Up Integration* dan *Top-down Integration* seperti yang dijelaskan pada Gambar 5.

***a. Bottom-up Integration***

*Bottom-up integration* memulai pengembangan dan pengujiannya dengan modul atom. Karena modulnya terintegrasi dari bawah ke atas, pemrosesan yang diperlukan elemen subordinate hingga tingkat yang diperlukan selalu tersedia. Berikut langkah-langkah yang diimplementasikan dalam strategi *bottom-up integration* :

i. Komponen tingkat rendah digabungkan menjadi *cluster* sehingga dapat melakukan sub-fungsi *software* yang spesifik.
ii. Sebuah *driver* lalu dilaporkan untuk mengoordinasikan masukan dan keluaran (I / O) test case.

iii. Grupnya diuji.
iv. *Driver* terlepas, dan *cluster* yang ada digabungkan dan bergerak ke atas dalam struktur program seperti tertera pada Gambar 6.

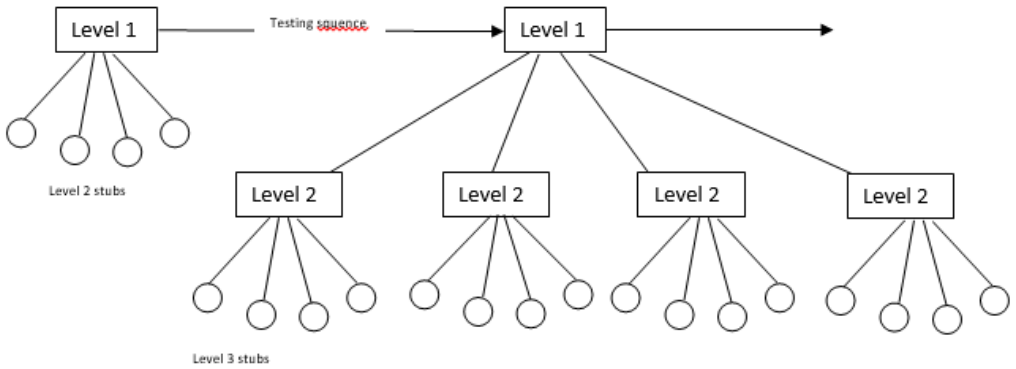



Gambar 6. *Bottom-up Integration Testing*

***b. Top-down Integration***

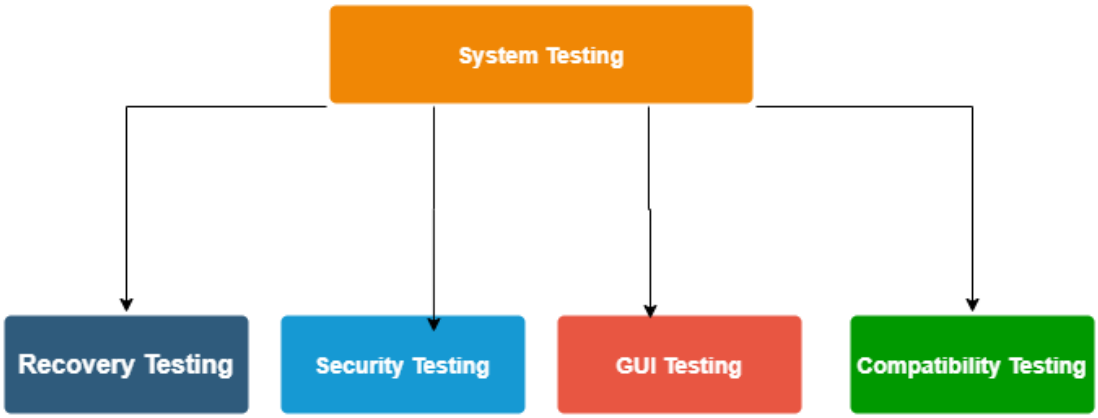
Integrasi *top-down* adalah prosedur *incremental* untuk membangun format program. Format-format terintegrasi dengan bergerak ke bawah struktur, dimulai dari modul main control. Modul *subordinate* sampai modul main control kemudian dimasukkan ke struktur di kedalaman pertama. Aktivitas integrasi dilakukan dengan 5 langkah di bawah :

i. Format main control diperlukan untuk test *driver*, dan *stub* diganti untuk seluruh komponen yang secara langsung berada di bawah modul *central control*.
ii. Terikat pada teknik *Integration*, *subordinate stub* yang digunakan diganti satu per satu dengan bagian yang asli.
iii. Testing dilakukan setiap elemen terintegrasi.
iv. Setelah mengani setiap rangkaian eksperimen, *stub* lain diganti dengan komponen yang asli.
v. Pengujian regresi dapat dilakukan agar memastikan jika kesalahan baru belum muncul. Masalah logistik dapat terjadi pada tahap ini. Masalahnya terjadi saat menguji modul tingkat rendah yang membutuhkan pemeriksaan tingkat atas. *Stub* menggantikan modul level bawah pada permulaan pengujian *top-down*. Jadi tidak ada data yang bisa bergerak ke atas. Langkah langkah tersebut di ilustrasikan pada Gambar 7.




Gambar 7. *Top-down integration testing*

### *System testing*




Gambar 8. Hirarki *System Testing*

*System testing* adalah tingkat pengujian *software* atau hardware yang dimana pengujian dilakukan secara lengkap, sistem diintegrasi untuk menilai kepatuhan sistem dengan persyaratan yang ditentukan. Syestem Testing memiliki 4 macam diantaranya adalah *Recovery testing*, *Security Testing*, *GUI Testing* dan *Compatibility Testing* seperti yang tertera pada Gambar 8. Pengujian dimasukkan dalam kategori *black-box testing*. Beberapa jenis pengujian sistem yang berbeda adalah sebagai berikut:

#### *a. Recovery testing*

Dalam proses pengujian ini, aplikasi dapat melakukan pemulihan dari kerusakan, kegagalan *hardware* dan masalah serupa lainnya. Selain itu, *recovery otomatis*, *re-initialization*, *check-pointing mechanism*, *data recovery*, dan *restart* dievaluasi kebenarannya. *Recovery testing* memaksa *software* untuk gagal dalam berbagai cara agar dapat diketahui jika rekonstruksi telah dilakukan dengan benar.

#### *b. Security testing*

Security testing mencoba untuk mengevaluasi mekanisme perlindungan yang dibangun ke dalam sistem yang dapat melindungi sistem dari gangguan yang dapat merusak sistem. Selama *security testing*, tester akan mencoba untuk menembus *system* secara disengaja, melakukan hal-hal seperti mencoba memperoleh sandi, menyerang sistem dengan *software custom* yang

dirancang untuk memecah pertahanan yang telah dibangun, membuat *system* kewalahan yang akan membuat *system* menolak pelayanan kepada user, menyebabkan *system error* secara sengaja, melakukan pembobolan disaat pemulihan, maupun melakukan penelusuran terhadap data yang tidak aman dengan harapan untuk menemukan *key* untuk masuk kedalam sistem.

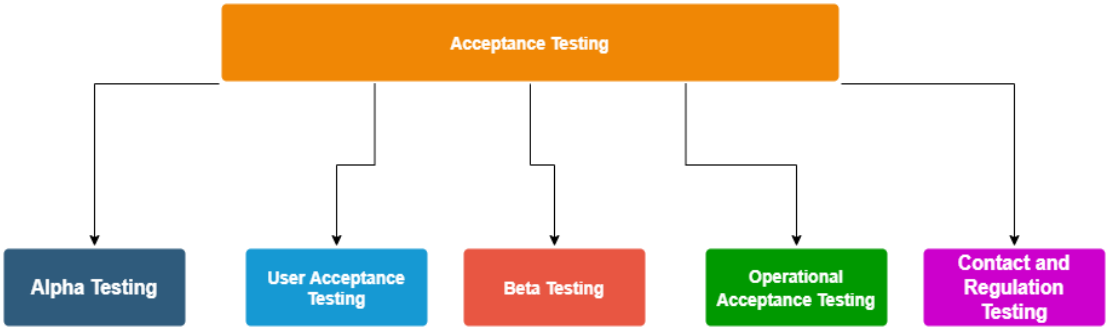
***c. Graphical user interface testing***

Dalam pengujian ini, GUI dari produk terlibat untuk memastikan produk memenuhi spesifikasi tertulis. Pengujian GUI dilakukan untuk memeriksa kontrol pada UI seperti menu, tombol, ikon, semua jenis toolbar, bar menu, kotak dialog, window, dll.

***d. Compatibility testing***

*Compatibility testing* bertanggung jawab pada kompatibilitas sistem dengan lingkungan sekitarnya. *Compability testing* memeriksa keselarasan sistem yang dikembangkan dengan berbagai komponen lain seperti *hardware*, *software* tambahan, DBMS dan sistem operasi, dll.

***Acceptance testing***




Gambar 9. Hirarki *Acceptance Testing*

Dalam pendekatan ini, pengujian dilakukan untuk mengotentikasi apabila produk dikembangkan sesuai standar dan kriteria rinci dan memenuhi semua persyaratan yang ditentukan oleh *user* atau tidak. pengguna melakukan jenis pengujian ini, dimana produk dikembangkan secara eksternal oleh pihak ketiga. *Acceptance testing* masuk dalam kategori pendekatan *black-box testing*, di mana user tidak terlalu banyak terlibat dalam kerja internal pengujiannya. Pendekatan *acceptance testing* juga dikenal sebagai pengujian QA, pengujian validasi, pengujian final, *application* testing ataupun factory acceptance testing. Dalam rekayasa perangkat lunak, acceptance testing dijalankan pada dua tingkat yang berbeda, satu di sistem tingkat provider dan satu lagi di tingkat end user. Acceptance Testing memiliki beberapa macam seperti yang dijelaskan pada Gambar 9, diantaranya yaitu:

***a. User Acceptance Testing***

*User acceptance testing* merupakan langkah penting sebelum sistem akhirnya diterapkan ke *end user*. *User acceptance testing* umumnya dilakukan oleh *software* yang sebenarnya digunakan untuk memastikan bahwa *software* tersebut dapat menangani tugas yang ditentukan dalam skenario dunia nyata.

***b. Alpha Testing & Beta Testing***

Tim QA atau pengembang umumnya telah melakukan pengujian jenis ini. *integration testing*, *system testing* dan *unit testing* menggabungkan nama sebagai *alpha testing*. *Alpha testing* dilakukan dengan adanya pengembang dan dengan tidak adanya user. Di dalam pengujian, kriteria-kriteria seperti kesalahan ejaan, garis putus-putus, mendung arah dll akan

diperiksa. Setelah berhasil menyelesaikan *alpha testing*, *beta testing* dilakukan. *Beta testing* dilakukan oleh *user* yang menggunakan *software* dalam skenario sungguhan. *User* menyediakan penilaian kepada *developer* untuk hasil percobaan. Timbal balik dari user akan digunakan untuk meningkatkan kinerja sistem / produk sebelumnya dirilis ke user / pelanggan lain.

### *c. Operational Acceptance Testing*

Operational acceptance testing (pengujian kesiapan fungsional) adalah pendekatan untuk memastikan semua proses dan prosedur tertentu dari sistem tersedia yang dapat memungkinkan user / tester untuk menggunakannya.

### *d. Contact and Regulation Testing*

Dalam pengujian ini, sistem diuji terhadap kriteria yang disyaratkan sebagaimana disebutkan dalam kontrak dan juga membuktikan apabila system tersebut memenuhi segala peraturan pemerintah maupun regulasi otoritas lokal, hingga semua standar pentingnya.

## Klasifikasi Kelebihan dan Kekurangan Strategi Pengujian *Software*

Dalam penelitian ini telah dilakukan klasifikasi jurnal yang telah dilakukan pengumpulan pada tahap sebelumnya beberapa jurnal yang kami kumpulkan, berdasarkan dengan metode yang telah kami lakukan. Berikut beberapa hasilnya.

### *Unit Testing*

Tabel 6. Kelebihan dan Kekurangan *Unit Testing*

| Tipe Pengujian | Kelebihan | Kekurangan | Sumber |
|---|---|---|---|
| ***Functional Testing*** | - Cocok digunakan untuk jenis kode yang berskala besar pada *software*<br>- Pengujian tidak dilakukan melalui kode<br>- Sebagai penjembatan antara prespektif user dan *developer* | - Terbatasnya ruang untuk pengujian karena hanya sebagian yang diuji<br>- Secara pengujian kurang efisien karena bergantung dari *skill tester* | [16] |
| ***Structural Testing*** | - Memberikan pengujian perangkat secara menyeluruh<br>- Membantu dalam menemukan *defect* pada tahap awal.<br>- Membantu dalam penghapusan kode yang tidak terpakai | - Membutuhkan pengetahuan tentang kode saat melakukan tes.<br>- Membutuhkan pelatihan pada penggunaan tool untuk pengujian. | [4] |
| ***Intuitive Testing*** | - Terbukti efektif bila digunakan dalam kombinasi teknik pengujian yang lain.<br>- Pengujiannya hemat biaya dan waktu | - Bergantung pada pengalaman orang yang menguji.<br>- Tidak dapat menjamin bahwa *software* telah mencapai tolak ukur kualitas yang diharapkan | [16]<br>[9] |

| | | | |
|---|---|---|---|
| | - Sangat membantu untuk menebak bagian kode yang bermasalah | | |

## *Integration Testing*

Tabel 7. Kelebihan dan Kekurangan *Integration Testing*

| Tipe Pengujian | Kelebihan | Kekurangan | Sumber |
|---|---|---|---|
| ***Top-down Integration Testing*** | - Mudah saat melacak *bug*<br>- Memungkingkan dalam mendapatkan contoh awal<br>- Pelacakan *defect* pada desain utama lebih mudah | - Pengujian fungsi modul akhir (dasar) dilakukan pada tahap akhir.<br>- Membutuhkan versi desain yang lebih banyak | [5] |
| ***Bottom-up integration Testing*** | - Pengujian memiliki tingkat ketelitian yang lebih tinggi<br>- *Bug* yang terdapat pada *software* system mudah ditemukan | - Modul pada tingkat tertinggi akan di test paling akhir.<br>- Struktur aplikasi masih tidak dapat dilihat. | [14] |

## *System Testing*

Tabel 8. Kelebihan dan Kekurangan *System Testing*

| Tipe Pengujian | Kelebihan | Kekurangan | Sumber |
|---|---|---|---|
| ***Recovery Testing*** | - Memulihkan kualitas system karena setiap kali *bug* terdeteksi dan diperbaiki, kualitas sistem meningkat.<br>- Membantu mengurangi resiko kegagalan *software* produk di pasaran | - Memakan waktu karena pengujian bersifat secara acak.<br>- Biaya proses mahal<br>- Tidak dimungkinkan untuk mendeteksi semua potensi bug dalam beberapa kasus karena terkadang masalah tersebut tidak dapat diprediksi | [17] |
| ***Security Testing*** | - Mengidentifikasi serta mengetahui kerentanan sebuah sistem<br>- Mendapatkan pengetahuan penting tentang sistem digital *software* | - Jika pengujian tidak dilakukan dengan benar maka dapat mengakibatkan kerusakan yang besar.<br>- *Security testing* bisa sangat mahal dan rumit | [18] |

| | | | |
|---|---|---|---|
| | - Setelah melakukan *security testing*, maka akan mendapatkan sistem keamanan yang baik sehingga dapat mendapatkan Kepercayaan terhadap kustomer/klien | tergantung dari kondisi kasus(*software*). | |
| ***Graphical user interface testing*** | - Mengurangi jumlah resiko menuju *end of development life cycle* secara efisien<br>- Menguji antarmuka dari sudut pandang customer<br>- Meningkatkan keandalan produk dan meningkatkan kualitas produk. | - Membutuhkan memori yang lebih banyak, yang mengakibatkan sistem menjadi lambat.<br>- Metode pengujian sulit karena akses terbatas atau tidak ada akses ke *source code*. | [7] |
| ***Compatibility testing*** | - *Feedback* dalam tahap pengujian akan meningkatkan proses pengembangan.<br>- Memasitkan bahwa setiap prasyarat ditetapkan dan disetujui oleh klien dan teknisi<br>- Menjamin kesuksesan dalam bisnis | - *Compatibility testing* menjadikan kenaikan biaya produksi dan waktu.<br>- Penundaan pengujian biasa terjadi pada *compatibility testing*. Yang pada akhirnya dapat mengakibatkan kemunduran jadwal. | [19]<br>[20] |

## *Acceptance testing*

Tabel 9. Kelebihan dan Kekurangan *Acceptance Testing*

| Tipe Pengujian | Kelebihan | Kekurangan | Sumber |
|---|---|---|---|
| ***User Acceptance Testing*** | - Menjadikan biaya pemeliharaan yang berkelanjutan serendah mungkin.<br>- Memberikan peluang optimal untuk mengidentifikasi dan memperbaiki fitur yang rusak.<br>- Memberikan pandangan *end user* | - Pengguna mungkin menyesuaikan dengan cara kerja sistem dan tidak melaporkan kerusakan/*defect*<br>- Kemajuan test sulit diukur | [6] |

| | | | |
|---|---|---|---|
| ***Alpha testing*** | - Pengujian ini memberikan pandangan tentang kualitas *software* pada tahap awal<br>- Membantu dalam mengungkapkan bug yang dapat menimbulkan ancaman. | - *Alpha testing* tidak melibatkan pengujian *software* secara mendalam<br>- Waktu eksekusi pengujian lama | [6] |
| ***Beta testing*** | - Membantu dalam penganalisisan *feedback use*<br>- Membantu dalam mengurangi risiko kegagalan *software* dengan cara memahami sudut pandang *end user* mengenai produk | - Jika tidak ada *feedback* yang tepat yang diterima, pengujian akan menjadi sia-sia<br>- Menemukan *beta tester* yang tepat dengan pengetahuan yang baik tentang cara menggunakan produk dan fiturnya tergolong sulit. | [6] |
| ***Operational Acceptance Testing*** | - Memungkinkan interaksi fitur<br>- Memeriksa kerentanan keamanan produk *software*<br>- Memvalidasi berbagai aspek produk sebelum dirilis, sehingga memastikan pengalaman pengguna yang maksimal | - Merupakan pengujian non-fungsional, jadi fungsi pengujian bukanlah fitur dari prosedur pengujian.<br>- Stimulasi berbasis operasional sangat kompleks sehingga *developer* sistem bahkan tidak tahu apa solusi yang benar atau terbaik untuk masalah yang diberikan. | [21]<br>[10] |
| **Contact and Regulation Testing** | - Memvalidasi apabila standar dan hukum yang disyaratkan ditaati dengan benar<br>- Memvalidasi apabila UI dan fungsi berfungsi seperti yang diharapkan | - Spesifikasi profil, level dan modul harus spesifik.<br>- Pengetahuan lengkap tentang berbagai standar, norma, dan peraturan sistem yang akan diuji harus diketahui. | [2] |

Menurut data yang telah kami kumpulkan melalui survei literatur, strategi pengujian *software* memiliki kelebihan dan kekurangan. Kelebihan dan kekurangan tersebut telah kami klasifikasikan berdasarkan pendekatannya seperti *halnya unit testing, validasi testing, integrasi testing* dan *system testing*. *Unit testing* sendiri memiliki kelebihan diantaranya adalah cocok digunakan jika skala dari kode besar, memberikan pengujian yang menyeluruh, pengujian memakan biaya yang sedikit dan waktu yang efektif. Tetapi unit testing juga memiliki kekurangan diantaranya membutuhkan pengetahuan kode dalam melakukan pengujian, Pengujian tidak efisien jika skill dari tester tidak memumpuni.

Selanjutnya untuk *Integration testing* beberapa kelebihannya diantaranya mudah untuk menemukan bug, mudah dalam menemukan *defect* atau cacat, Pengujian memiliki tingkat ketelitian yang lebih tinggi. Dari kelebihan-kelebihan yang ada, integration testing juga memiliki kekurangan diantaranya adalah pengujian integrasi membutuhkan banyak versi *design* dan kerangka aplikasi pada saat melakukan pengujian masih belum bisa dilihat. Dapat disimpulkan pengujian integrasi memiliki tingkat efektif yang tinggi karena mudah mencari *bug* dan *defect* serta tingkat pengujian yang lebih tinggi.

Selanjutnya untuk *system testing* memiliki kelebihan diantaranya adalah membantu mengurangi resiko kegagalan produk *software* di pasaran, mengetahui kerentanan sebuah system pada *software* dan membantu dalam kesuksesan *software* di segi bisnis. Meskipun system testing memiliki beberapa kelebihan, *system testing* juga memiliki kekurangan diantranya adalah biaya prosses pengujian yang mahal, jika pengujian tidak dilakukan dengan benar maka akan mengakibatkan kerusakan besar. Dapat disimpulkan bahwa *system testing* sendiri memiliki tingkat resiko yang tinggi meskipun manfaat dari pengujian ini sangat membantu dalam menciptakan *software* yang baik di pasaran.

Tahap akhir dari strategi pengujian *software* adalah acceptance testing. *Acceptance testing* sendiri memiliki beberapa kelebihan diantaranya adalah menjadikan biaya pemeliharan lanjutan menjadi lebih murah atau efisien, membantu dalam mengurangi risiko kegagalan *software* dengan cara memahami sudut pandang *end user* mengenai produk, Memvalidasi berbagai aspek produk sebelum dirilis sehingga memastikan pengalaman user yang maksimal. Dari kelebihan kelebihan yang ada, pengujian acceptance juga memiliki kekurangan diantaranya adalah waktu pengujian yang lama, Jika tidak ada *feedback* yang tepat yang diterima maka pengujian akan menjadi sia-sia, dan keberhasilan dari pengujian ini bergantung pada bagaimana user menanggapi semua tahapan pengujian yang ada. Dapat disimpulkan bahwa pengujian ini akan berjalan sangat efektif jika end-user memberikan *feedback* yang baik.

## KESIMPULAN

Secara garis besar, strategi pengujian *software* terbagi menjadi 4 tipe, yang *Unit testing*, *Acceptance / Validation testing*, *Integration testing*, dan *System testing*. Dari tipe-tipe strategi pengujian *software* tersebut, masing-masing memiliki kelebihan dan kekurangannya tersendiri. Strategi pengujian *software* yang biasanya paling sering digunakan *developer* adalah unit testing, dimana pengujian tersebut dilakukan untuk menemukan *defect* pada tingkat modul atau komponen. Strategi pengujian *software* merupakan aktivitas penting dalam pengembangan peragkat lunak. Oleh karena itu, strategi yang ada ini diperlukan untuk menjaga kualitas dari perangkat lunak.

## REFERENSI

[1] I. Jovanovic, "Software Testing Methods and Techniques," *IPSI BgD Trans. Internet Res.*, vol. 5, no. 1, pp. 30–41, 2009, [Online]. Available: http://www.internetjournals.net/journals/tir/2009/January/Full Journal.pdf#page=31.

[2] A. A. Sawant, P. H. Bari, and P. . Chawan, "Software Testing Techniques and Strategies," *J. Eng. Res. Appl.*, vol. 2, no. 3, pp. 980–986, 2012.

[3] S. Thakare, S. Chavan, and P. Chawan, "Software Testing Strategies and Techniques," *Citeseer*, vol. 2, no. 4, pp. 682–687, 2012, [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Software+Testing+Strategies+and+Techniques#0.

[4] E. Daka and G. Fraser, "A survey on unit testing practices and problems," *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 201–211, 2014, doi: 10.1109/ISSRE.2014.11.

[5] M. J. Rehman, F. Jabeen, A. Bertolino, and A. Polini, "Software Component Integration Testing : A Survey," *Development*, no. 1, pp. 1–36.

[6] C. K. N. C. K. Mohd and F. Shahbodin, "Personalized Learning Environment: Alpha Testing, Beta Testing & User Acceptance Test," *Procedia - Soc. Behav. Sci.*, vol. 195, no. July, pp. 837–843, 2015, doi: 10.1016/j.sbspro.2015.06.319.

[7] E. Börjesson and R. Feldt, "Automated system testing using visual GUI testing tools: A comparative study in industry," *Proc. - IEEE 5th Int. Conf. Softw. Testing, Verif. Validation, ICST 2012*, pp. 350–359, 2012, doi: 10.1109/ICST.2012.115.

[8] D. Budgen, B. Kitchenham, S. Charters, M. Turner, P. Brereton, and S. Linkman, "Preliminary results of a study of the completeness and clarity of structured abstracts," 2007, doi: 10.14236/ewic/ease2007.7.

[9] M. Olan, "Unit testing: test early, test often," *J. Comput. Sci. Coll.*, vol. 19, no. 2, pp. 319–328, 2003.

[10] B. Littlewood and D. Wright, "Stopping rules for the operational testing of safety-critical software," *Proc. - Annu. Int. Conf. Fault-Tolerant Comput.*, pp. 444–451, 1995, doi: 10.1109/ftcs.1995.466955.

[11] L. Briand and Y. Labiche, *A UML-Based Approach to System Testing*, vol. 1, no. 1. 2002.

[12] N. Kitiyakara, "Acceptance testing HTML," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 2418, pp. 112–121, 2002, doi: 10.1007/3-540-45672-4_11.

[13] S. Batra and R. Rahul, "Available Online at www.jgrcs.info," *J. Glob. Res. Comput. Sci.*, vol. 2, no. 7, pp. 113–117, 2011.

[14] H. K. N. Leung and P. W. L. Wong, "A study of user acceptance tests," *Softw. Qual. J.*, vol. 6, no. 2, pp. 137–149, 1997, doi: 10.1023/A:1018503800709.

[15] R. W. Selby, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Trans. Softw. Eng.*, vol. SE-13, no. 12, pp. 1278–1296, 1987, doi: 10.1109/TSE.1987.232881.

[16] P. Runeson, "A survey of unit testing practices," *IEEE Softw.*, vol. 23, no. 4, pp. 22–29, 2006, doi: 10.1109/MS.2006.91.

[17] A. von Mayrhauser, M. Scheetz, E. Dahlman, and A. E. Howe, "Planner based error recovery testing," *Proc. Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 186–195, 2000, doi: 10.1109/issre.2000.885871.

[18] B. Potter and G. McGraw, "Software security testing," *IEEE Secur. Priv.*, vol. 2, no. 5, pp. 81–85, 2004, doi: 10.1109/MSP.2004.84.

[19] I. C. Yoon, A. Sussman, A. Memon, and A. Porter, "Effective and scalable software compatibility testing," *ISSTA'08 Proc. 2008 Int. Symp. Softw. Test. Anal. 2008*, no. May 2014, pp. 63–73, 2008, doi: 10.1145/1390630.1390640.

[20] T. Zhang, J. Gao, J. Cheng, and T. Uehara, "Compatibility testing service for mobile applications," *Proc. - 9th IEEE Int. Symp. Serv. Syst. Eng. IEEE SOSE 2015*, vol. 30, pp. 179–186, 2015, doi: 10.1109/SOSE.2015.35.

[21] W. Paeffgen, S. Sassen, and C. Soddu, "Operational acceptance of integrated Galileo operations facilities provided as customer furnished items," *SpaceOps 2006 Conf.*, pp. 1–6, 2006, doi: 10.2514/6.2006-5513.