IMPLEMENTASI PERHITUNGAN *DEFECT* PADA *SOFTWARE* SISTEM INFORMASI KULIAH KERJA NYATA (KKN) UNIVERSITAS LAMPUNG

(Skripsi)

Disusun Oleh:

NINA DWI JAYANTI

JURUSAN ILMU KOMPUTER

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS LAMPUNG

2019

ABSTRACT

IMPLEMENTATION OF DEFECT CALCULATION IN THE REAL JOB LECTURE INFORMATION SYSTEM SOFTWARE (KKN) OF UNIVERSITAS LAMPUNG

By

NINA DWI JAYANTI

Software development requires high costs. Cyclomatic complexity metrics, Halstead metrics, LLOC metrics could be used to measure the amount of defects and evaluate the software for further testing. This research was conducted in the module of the University of Lampung's Real Work Lecture Information System (KKN) in 2018. In this study, the largest defect is obtained in the randomization file Controller.php, so developers must be aware of the Controller.php randomization file because it is a file prone to defects. In this study, the researcher obtain two alternative solutions that can be used, namely the development of Real Job Lecture Information System (KKN) and rebuilding the University of Lampung Real Work Lecture Information System (KKN). The estimated time resources required to develop the Real Work Lecture Information System (KKN) is 2¼ days to 30 days, while the estimated time resources to rebuild the Real

Work Lecture Information System (KKN) is 24 days to 180 days. The estimated cost needed to develop is Rp.3,851,096 / person and the estimated cost resource needed to rebuild the Real Work Lecture Information System (KKN) is Rp.1.956.112/person to Rp.14.670.846/ person. So software developers need to consider if they are going to develop or rebuild a Real Job Lecture Information System (KKN). This research can also be used as a measurement reference by using cyclometic complexity metrics, halstead metrics, LLOC metrics and defect calculations in other software.

Keyword: cyclomatic complexity metrics , defect, halstead metrics, LLOC metrics, resource estimation

ABSTRAK

IMPLEMENTASI PERHITUNGAN *DEFECT* PADA *SOFTWARE* SISTEM INFORMASI KULIAH KERJA NYATA (KKN) UNIVERSITAS LAMPUNG

Oleh

NINA DWI JAYANTI

Pengembangan *software* memerlukan biaya yang tinggi. *Cyclomatic complexity metrics, Halstead metrics, LLOC metrics* dapat digunakan untuk mengukur besarnya *defect*, dan evaluasi *software* untuk pengujian selanjutnya. Penelitian ini dilakukan pada modul *Administrator* Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung tahun 2018. Dalam penelitian ini diperoleh angka *defect* terbesar yakni pada file pengacakanController.php, sehingga para pengembang harus waspada pada file pengacakanController.php karna merupakan file rawan cacat.

Dalam penelitian ini diperoleh dua alternatif solusi yang dapat digunakan yaitu pengembangan Sistem Informasi Kuliah Kerja Nyata (KKN) dan membangun ulang Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung. Adapun estimasi sumber daya (waktu) yang dibutuhkan untuk melakukan pengembangan Sistem Informasi Kuliah Kerja Nyata (KKN) adalah sebesar 2¼ hari s/d 30 hari, sedangkan estimasi sumber daya (waktu) untuk membangun ulang SI KKN adalah 24 hari s/d 180 hari. Estimasi (biaya) yang dibutuhkan untuk melakukan pengembangan adalah sebesar Rp.3.851.096/orang dan estimasi sumber daya (biaya) yang dibutuhkan untuk membangun ulang Sistem Informasi Kuliah Kerja Nyata (KKN) adalah Rp.1.956.112/orang s/d Rp14.670.846/orang. Sehingga para pengembang *software* perlu mempertimbangkan jika akan melakukan pengembangan atau membangun ulang Sistem Informasi Kuliah Kerja Nyata (KKN). Penelitian ini juga bisa digunakan untuk acuan pengukuran dengan menggunakan *cyclometic complexity metrics, halstead metrics, LLOC metrics* dan perhitungan *defect* pada *software* lainya.

Kata Kunci : *cyclomatic complexity metrics , defect, estimasi sumber daya, halstead metrics, LLOC metrics.*

IMPLEMENTASI PERHITUNGAN *DEFECT* PADA *SOFTWARE* SISTEM INFORMASI KULIAH KERJA NYATA (KKN) UNIVERSITAS LAMPUNG

Oleh

NINA DWI JAYANTI

Skripsi

Sebagai Salah Satu Syarat Untuk Mencapai Gelar

SARJANA KOMPUTER

Pada

Jurusan Ilmu Komputer

Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Lampung

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS LAMPUNG

2019

Judul Skripsi	:	**IMPLEMENTASI PERHITUNGAN *DEFECT* PADA *SOFTWARE* SISTEM INFORMASI KULIAH KERJA NYATA (KKN) UNIVERSITAS LAMPUNG**
Nama Mahasiswa	:	Nina Dwi Jayanti
No. Pokok Mahasiswa	:	1517051010
Jurusan	:	Ilmu Komputer
Fakultas	:	Matematika dan Ilmu Pengetahuan Alam

MENYETUJUI

1. Komisi Pembimbing

Anie Rose Irawati, S.T., M.Cs.	**Favorisen R. Lumbanraja, Ph.D.**
NIP. 19791031 200604 2 002	NIP. 19830110 200812 1 002

2. Mengetahui
Ketua Jurusan Ilmu Komputer
FMIPA Universitas Lampung

Dr. Ir. Kurnia Muludi, M.S.Sc.
NIP. 19640616 198902 1 001

MENGESAHKAN

1. Tim Penguji

   Ketua : **Anie Rose Irawati, S.T., M.Cs.** .....

   Sekretaris : **Favorisen R. Lumbanraja, Ph.D.** .....

   Penguji
   Bukan Pembimbing : **Dr. rer.nat Akmal Junaidi, M.Sc.** .....

2. Dekan Fakultas Matematika dan Ilmu Pengetahuan Alam

Drs. Suratman, M.Sc.
NIP. 19640604 199003 1 002

Tanggal Lulus Ujian Skripsi : **27 September 2019**

PERNYATAAN

Saya yang bertanda tangan di bawah ini, menyatakan bahwa skripsi saya yang berjudul "**Implementasi Perhitungan *Defect* pada Software Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung**" merupakan karya saya sendiri dan bukan karya orang lain. Semua tulisan yang tertuang di skripsi ini telah mengikuti kaidah penulisan karya ilmiah Universitas Lampung. Apabila dikemudian hari terbukti skripsi saya merupakan hasill jiplakan atau dibuat orang lain, maka saya bersedia menerima sanksi berupa pencabutan gelar yang saya terima.

Bandar Lampung, 10 Oktober 2019

Nina Dwi Jayanti

1517051010

RIWAYAT HIDUP

Penulis dilahirkan pada tanggal 30 Maret 1998 di Fajar Baru, Mesuji, sebagai anak kedua dari empat bersaudara dengan Ayah bernama Isnen Efendi dan Ibu Ngaisah. Penulis menyelesaikan pendidikan dasar di SD Negeri 4 Fajar Baru dan selesai pada tahun 2009. Pendidikan menegah pertama di SMP Negeri 1 Panca Jaya dan selesai pada tahun 2012, kemudian melanjutkan ke pendidikan menengah atas di SMK Negeri 1 Simpang Pematang yang diselaikan pada tahun 2015. Pada tahun 2015 penulis terdaftar sebagai mahasiswa Jurusan Ilmu Komputer Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Lampung dengan jalur SNMPTN. Selama menjadi mahasiswa beberapa kegiatan yang dilakukan penulis antara lain:

1. Pada bulan Januari 2016 penulis mengikuti Karya Wisata Ilmiah di Pekon Batu Tegi, Kecamtan Air Naningan, Kabupaten Tanggamus.
2. Pada bulan Januari 2018 sampai dengan Maret 2018 penulis melaksanakan kerja praktik di PT PLN (Persero) Sektor Dalkit Bandar Lampung pada Bagian Sumber Daya Manusia (SDM).
3. Pada bulan Juli 2018 sampai dengan September 2018 melaksanakan Kuliah Kerja Nyata di Desa Mulyo Asri, Kecamatan Bumi Agung, Kabupaten Lampung Timur.

MOTO

“Allah tidak membebani seseorang melainkan sesuai dengan kesanggupannya.”
(QS. Al-Baqarah: 286)

“Sesungguhnya sesudah kesulitan itu ada kemudahan.”

(QS. Al Inshirah: 6)

“Man Jadda Wa Jadda (siapa yang bersungguh-sungguh pasti berhasil)”

“Man Shobaro Zafiro (Siapa yang bersabar akan beruntung)”

“Man Saaro’Alaa Darbi Washola (Siapa yang berjalan di jalur-Nya akan sampai)”

PERSEMBAHAN

Puji syukur saya panjatkan kepada Allah SWT atas segala berkah-Nya sehingga Skripsi ini dapat terselesaikan.

Kupersembahkan Karyaku ini kepada :

Teristimewa kedua Orangtuaku

Bapak dan Mamak yang luar biasa saya sayangi dan saya cintai. Orang yang tiada henti-hentinya melimpahkan Kasih Sayang yang bagi saya tidak ternilai berharganya dari apapun. Selalu Mengajari, Menasihati dan Membimbing dengan kesabaran dan ketulusan yang luar biasa. Setiap doanya selalu terselip nama saya agar setiap langkah saya menjadi berkah dan berguna untuk saya dan orang banyak. Mengorbankan seluruh jiwa dan raga hanya demi kebahagiaan Putra dan Putrinya. Serta mba Evi, mas Yahya, mas Doni, Icha, Akbar, Citra yang sangat saya sayangi, merekalah yang selalu membimbing, menghibur, usil, menghargai, menyayangi, menyemangati dan selalu mendoakan apapun yang terbaik untuk diriku. Dan tak terlupa untuk keluarga besar tercinta.

Keluarga Ilmu Komputer 2015, Serta Almamater Tercinta, Universitas Lampung.

SANWACANA

Alhamdulillah, puji syukur kehadirat Allah SWT, atas berkat karunia-Nya sehingga

penulis dapat menyelesaikan skripsi di Jurusan Ilmu Komputer Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Lampung.

Skripsi ini diselesaikan dengan judul penelitian "Implementasi Perhitungan *Defect* pada *Software* Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung.

Dalam penyusunan skripsi ini, penulis mendapat bantuan, dukungan dan dorongan dari berbagai pihak. Terima kasih penulis sampaikan kepada semua pihak yang telah membantu dan berperan besar dalam menyusun skripsi ini, antara lain.

1. Kedua orangtua tercinta, Bapak Isnen Efendi dan Ibu Ngaisah yang selalu mendoai dan tidak pernah henti-hentinya selalu memberikan rasa hangatnya kasih sayang, selalu mendukung, membimbing, menghargai setiap proses penulis selama ini.

2. Kakak-kakakku tercinta Evi Ningtias, Yahya Marhabah, yang selalu memberi doa, motivasi, membimbing, menghibur, menghargai, menyayangi dan usil yang tiada henti dari kecil hingga tulisan ini dibuat hal tersebut masih hangat terasa. Serta kurcaci-kurcaci Tyas Nabila Patra, Akbar Muzaki, Tasya Citra Safira, yang selalu membuat penulis terhibur ketika melihat aktivitas mereka sehari-hari di media sosial.
3. Ibu Anie Rose Irawati, S.T, M.Cs, selaku pembimbing utama saya dalam penelitian ini, yang telah memberikan ide, motivasi, pemberi semangat, nasihat, serta keikhlasan beliau yang luar biasa dalam membantu saya menyelesaikan skripsi ini.
4. Bapak Favorisen R. Lumbanraja, S.Kom, M.Si, Ph.D, selaku pembimbing kedua yang telah memberikan ide, kritik, dan nasihat sehingga penilis dapat menyelesaikan skripsi ini.
5. Bapak Dr.rer.nat. Akmal Junaidi, M.Sc., selaku pembahas yang telah memberikan banyak masukan, ide, kritik, serta saran yang bermanfaat dalam perbaikan dalam proses menyelesaikan skripsi ini.
6. Bapak Drs. Suratman, M.Sc. sebagai Dekan FMIPA Universitas Lampung.
7. Bapak Dr. Ir. Kurnia Muludi, M.S.Sc., selaku Ketua Jurusan Ilmu Komputer FMIPA Universitas Lampung.
8. Bapak Didik Kurniawan, S.Si., M.T. sebagai Sekretaris Jurusan Ilmu Komputer FMIPA Universitas Lampung.
9. Bapak Febi Eka Febriansyah, ST. M.T selaku dosen Pembimbing Akademik yang telah membimbing, memotivasi, dan mendukung penulis sehingga penulis memiliki target dalam setiap menyelesaikan sesuatu.

10. Seluruh Bapak dan Ibu Dosen Jurusan Ilmu Komputer yang telah memberikan Ilmu dan pelajaran hidup selama penulis menjadi mahasiswa.
11. Ibu Nora, Bunda, Ibu Wiwik, Bang Zai yang telah membantu dalam segala urusan administrasi di Jurusan Ilmu Komputer dan memberikan saya semangat dalam mengejar gelar sarjana Ilmu Komputer.
12. Mas Wardoni selaku Kakak, Sahabat, Rekan Kerja dan Teman terdekat, Calon Suami penulis. Kakak yang selalu sabar menjadi pendengar cerita suka ataupun duka, pandai membuat penulis selalu tersenyum, sangat sabar membantu penulis, pemberi semangat dan masukan dalam penulis menghadapi masalah.
13. Rekan seperjuangan Novenda, Resvy Haryati, yang selalu memberikan semangat, selalu membantu dalam persiapan dari seminar proposal, seminar hasil hingga sidang komprehensif, sahabat yang selau ada, sahabat selama kuliah yang tahu kekurangan dan kelebihan masing-masing, sahabat yang paling peduli, keluarga selama di Bandar Lampung. Tak lupa juga dengan mba Egidiah Amalia selaku teman seperbimbingan, yang selalu mengajarkan penulis dalam penyusunan skripsi ini, dan saling menyemangati satu sama lainya.
14. Keluarga Wisma Dewi yang selalu memberikan semangat untuk segera wisuda.
15. Keluarga *classic* A yang telah menjadi keluarga selama kuliah di Jurusan Ilmu Komputer.
16. Keluarga Ilmu Komputer angkatan 2015 yang selalu menjadi penyemangat untuk berjuang sampai wisuda.
17. Seluruh Kakak dan Adik tingkat yang secara tidak langsung memberikan pembelajaran, ilmu dan masukan dan saran dalam menghadapi perkuliahan.

Penulis menyadari bahwa skripsi ini masih jauh dari kata sempurna, semooga skripsi ini memberikan manfaat dan keberkahan bagi semua civitas Ilmu Komputer Universitas Lampung.

Bandar Lampung, 27 September 2019

Nina Dwi Jayanti

NPM 1517051010

DAFTAR ISI

Halaman

DAFTAR TABEL

Tabel **Halaman**

DAFTAR GAMBAR

Gambar **Halaman**

I. PENDAHULUAN

A. Latar Belakang

Pengembangan *software* yang berkualitas dan kompleks membutuhkan biaya yang tinggi. Biaya untuk memperbaiki cacat akibat *requirement* setelah fase pengembangan dapat mencapai 100 kali dari biaya perbaikan pada fase tersebut, biaya untuk memperbaiki cacat pada tahap desain setelah pengiriman produk mencapai 60 kali, sedangkan biaya untuk memperbaiki cacat pada tahap desain yang ditemukan oleh pelanggan adalah 20 kali (Strangio, 2009).

Perusahaan pengembangan *software* mencari cara untuk meningkatkan kualitas *software* tanpa mengalokasikan terlalu banyak sumber daya dalam kegiatan jaminan kualitas, seperti pengujian. Menerapkan upaya pengujian yang sama untuk semua modul sistem *software* bukanlah pendekatan yang optimal, karena distribusi cacat di antara bagian-bagian individual dari suatu sistem tidak seragam. Menurut hukum *Pareto-Zipf* aturan empiris operasi 80:20 yakni 20% kode yang menyebabkan 80% terjadinya *bug*. Oleh karena itu, dilakukan sebuah penelitian untuk mencari bagian yang rawan kecacatan (Madeyski dan Jureczko, 2014).

Metrik proses adalah metrik yang menggambarkan efektifitas, peningkatan pengembangan *software* dan pemeliharaan *software,* termasuk upaya yang diperlukan untuk memproses cacat yang ditemukan selama pengujian, dan waktu respon untuk menghasilkan produk (Kaur dan Maini, 2016).

Madeyski dan Jureczko (2014) mengungkapkan bahwa metrik proses dapat digunakan untuk memprediksi tingkat kecacatan pada sebuah *software*. Pendekatan ini menggunakan hubungan/korelasi antara metrik proses dan jumlah kerusakan yang diselidiki. Penggunaan metrik proses juga bisa digunakan untuk melihat bagian mana saja yang rawan cacat. Pendekatan ini dibangun untuk mengurangi bagian rawan cacat kerusakan yang diselidiki. Prediksi kecacatan pada *software* digunakan untuk efisiensi dalam hal kepuasan konsumen. Cacat *software* paling banyak ditemukan pada modul, yakni *function* dan *method* pada kode sumber.

Kaur dan Maini (2016) mengungkapkan bahwa untuk mendeteksi *bugs* dapat menggunakan aturan dari beberapa *software metrics*. Aturan *Long method* dapat diprediksi dengan perhitungan tiga *metrics* yakni *Lines of Source Code (LOSC), Halstead Metrics,* dan *Cyclomatic Complexity Metrics.* Karakteristik internal dapat dilihat pada *Compodefectnen-Level Design Metrics* yang terdapat pada kode sumber yang dapat diketahui melalui atribut yang terdapat pada metrics *Mc Cabe Cyclomatic Complexity* dan *Halstead*. *Mc Cabe Cyclomatic Complexity Metrics* dan *Halstead Metrics* merupakan suatu ukuran untuk mengukur kompleksitas suatu program.

Penelitian ini merupakan implementasi dari *Helstad Metrics*, *Mc Cabe Cyclomatic Complexity Metrics dan Lines of Code (LOC)* untuk mengetahui jumlah *defect*, tempat terjadinya *defect*, dan evaluasi resiko yang kemungkinan terjadi guna untuk menentukan sumber daya yang diperlukan dalam proses pengujian pada *software* dengan menggunakan studi kasus Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung yang dikembangkan oleh Ichwan Almaza (2018). Hasil dari penelitian ini yakni sebagai rekomendasi untuk para pengembang *software* agar mengetahui metrik proses mana yang sesuai untuk memprediksi kecacatan sebuah *software* lainya. Manfaat dari penelitian ini adalah agar ketika melakukan pengembangan pada Sistem Informasi Kuliah Kerja Nyata (KKN) tidak perlu melakukan pengujian pada keseluruhan modul, melainkan pada modul yang sudah diprediksi rawan cacat. Penelitian ini juga untuk mengetahui seberapa besar sumber daya yang dibutuhkan dalam melakukan pengujian pada Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung.

B. Rumusan Masalah

Adapun rumusan masalah dalam penelitian ini, antara lain:

a) Indikator dalam melakukan pengukuran kecacatan *software* yang berupa *defect functional* dan banyaknya *bugs.*
b) Cara untuk mengukur kecacatan pada sebuah *software* dengan menggunakan *Mc Cabe Cyclomatic Complexity Metrics* melalui *tool* dan manual.

c) Cara untuk mengukur tingkat kecacatan pada *software* dengan menggunakan *Halstead Metrics* melalui *tool* dan manual.
d) Cara untuk mengukur kecacatan pada sebuah *software* dengan menggunakan *LOC (Lines Of Code) Metrics* melalui *tool* dan manual.
e) Mengetahui dan mengkonfirmasi jumlah *defect*, tempat munculnya *defect*, dan evaluasi resiko sehingga mampu memperkirakan sumber daya yang dikeluarkan untuk pengujian.

C. Batasan Masalah

Adapun batasan masalah dari penelitian ini, antara lain:

a) Penelitian ini membahas tentang implementasi pengukuran dengan menggunakan *Mc Cabe Cyclomatic Complexity Metrics*, *Halstead Metrics* dan *LOC (Line of Code) Metrics.*
b) Dalam pengimplementasianya, peneliltian ini melakukan perhitungan kecacatan secara manual dan mengkonfirmasi hasilnya dengan perhitungan menggunakan *tool.*
c) Implementasi pengukuran kecacatan dilakukan pada *software* yakni pada fungsi administator Sistem Inforrmasi Kuliah Kerja Nyata (KKN) Universitas Lampung yang dibangun oleh Ichwan Almaza (2018).

D. Tujuan Penelitian

Adapun tujuan dari penelitian ini, antara lain:

a) Menghitung *Mc Cabe Cyclomatic Complexity Metrics*, *Halstead Metrics* dan *LOC (Line of Code) Metrics*, dan angka *defect* dalam modul *controller* dan *view Administrator* Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung.
b) Mengetahui besarnya sumberdaya yang diperoleh untuk evaluasi/pengujian di Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung.

E. Manfaat Penelitian

Adapun manfaat dari penelitian ini, antara lain:

a) Manfaat dari penelitian ini adalah agar ketika melakukan pengembangan pada Sistem Informai Kuliah Kerja Nyata (KKN) tidak perlu melakukan pengujian pada keseluruhan modul melainkan pada modul-modul yang sudah diprediksi rawan cacat.
b) Hasil dari penelitian ini dapat menjadi rekomendasi untuk pengembang *software* dalam mempediksi kecacatan dengan metrik proses pada *software* lainya.
c) Menjadi rekomendasi persiapan sumber daya untuk pengujian selanjutnya.

II. TINJAUAN PUSTAKA

A. Pengertian *Software Metrics*

Software metrics dapat didefinisikan sebagai pengukuran secara terus menerus untuk proses pengembangan *software,* penyediaan produk, dan informasi manajemen waktu untuk meningkatkan kualitas proses dan produk *software. Software metrics* ditentukan dengan mengukur properti dan spesifikasi dari bagian software (Debbarma dkk., 2013).

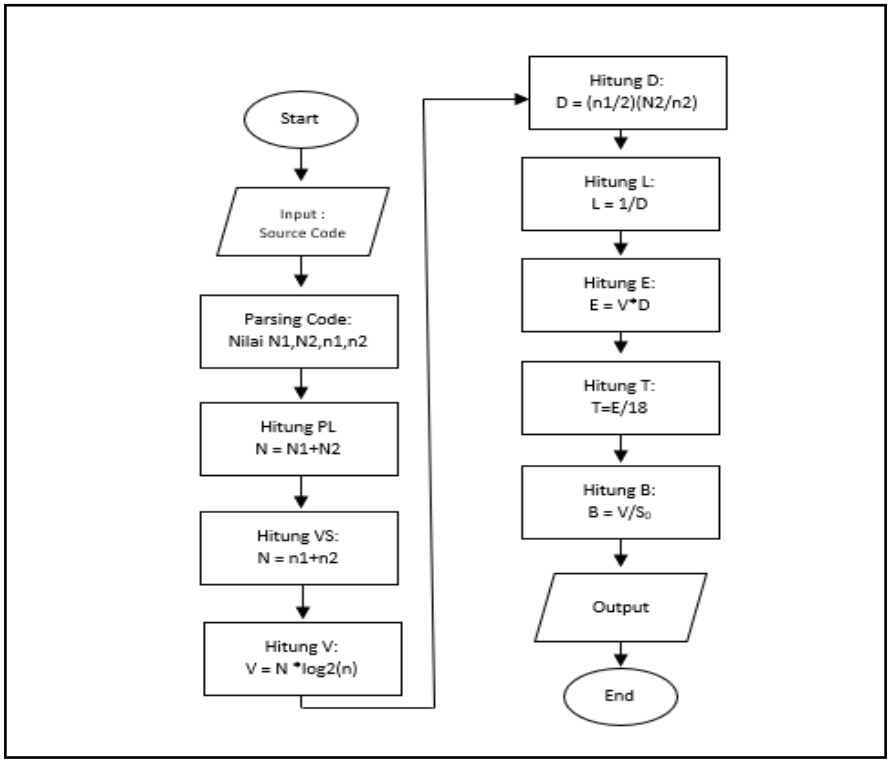
Sofware metrics juga dapat diartikan sebagai entitas penting di dalam seluruh siklus hidup *software*. *Software metrics* menyediakan pengukuran untuk pengembangan *software*, termasuk dokumen persyaratan *software*, desain, program, dan pengujian (Rawat dkk., 2012).

Pengukuran dapat digunakan pada seluruh proyek *software* untuk membantu estimasi, kontrol kualitas, penilaian produktivitas, pengendalian proyek, membantu menilai kualitas produk dan membantu dalam pengambilan keputusan hasil proyek (Susanto dkk., 2015).

B. Halstead Metrics

Halstead Metrics pertama kali diperkenalkan oleh *Maurice Howard Halstead* pada tahun 1977 sebagai bagian dari ilmu empiris pengembangan *software*. Model pengamatan dari *halstead* bahwa *sotware metrics* harus mencerminkan implementasi atau ekspresi algoritma dalam bahasa yang berbeda, tetapi tidak tergantung pada pelaksanaan *platform* tertentu. Adapun teori *maurice Halstead* (1971-1979) bahwa sebuah program *P* adalah kumpulan token, tersusun dari 2 elemen dasar yakni *operator* dan *operand.* Adapun kesulitan dalam menghitung *Halstead Metrics* yakni membedakan jenis *operator* dan operan pada setiap bahasa pemrograman dan juga membandingkan seberapa kompleksitasnya sebuah kode sumber (Chhabra dan Bansal, 2014).

Halstead Metrics merupakan metrik yang berfungsi untuk mengevaluasi dan melakukan pengukuran kode sumber bedasarkan *operator* dan *operand*. *Operator* merupakan simbol-simbol yang digunakan untuk melakukan operasi tertentu. Pada dasarnya ada beberapa jenis *operator* yang digunakan dalam kode sumber bahasa pemrograman *PHP* antara lain *operator arimatika*, *operator* pembandingan, *operator* logika, *operator string*, dan *operator assigm*ent. *Operand* merupakan nilai asal yang digunakan di dalam proses operasi. Pranata dkk., (2014) mengungkapkan bahwa dengan mengidentifikasi *operator* dan *operand* dari program, maka dapat diidentifikasikan atribut untuk *software* tertentu, seperti *Program Length, Difficult Level* dan sebagainya Berikut ini *flowchart* dari proses perhitungan kompleksitas kode sumber berdasarkan metode *Halstead's Volume.*



Gambar 1. *Flowchart metode Halstead's Volume* (Pranata dkk., 2016).

Dari Gambar 1 dapat dilihat bahwa pengukuran kompleksitas kode sumber menggunakan *Halstead's Volume Metrics* diawali dengan input kode sumber oleh user, selanjutnya proses *parsing code* dilakukan oleh sistem untuk mendapatkan nilai *N1, N2, n1, n2* yang selanjutnya akan digunakan untuk proses perhitungan *PL (Program Length), VS (Vocabuary Size), V* (*Program Volume*), *D (Difficulty Level), L (Program Level), E (Effort to Implement), T (Time to Implement),* dan *B (Number of Delivered Bugs),* dan hasilnya akan ditampilkan oleh *user.*

Berikut merupakan metode *Halstead Metrics:*

- Jumlah *operator* yang berbeda dalam program *(n1)*
- Jumlah *operand* yang berbeda dalam program *(n2)*
- Total jumlah kemunculan *operator* dalam program *(N1)*
- Total jumlah kemunculan *operand* dalam program *(N2)*

- *Vocabulary Size (n)*

  Vocabulary size adalah penjumlahan *operator* dan *operand* yang berbeda dalam program. Dapat dihitung dengan menggunakan persamaan 1 berikut:

  n = n1 + n2.....(1)

- *Program Length (N)*

  Panjang program adalah penjumlahan antara total jumlah kemunculan *operator* dan *operand* dalam sebuah program. Dapat dihitung dengan menggunakan persamaan 2 berikut:

  N = N1 + N2.....(2)

- *Program Volume (V)*

  Program Volume (V) adalah informasi isi dari program, diukur dalam bit matematika. Dihitung sebagai panjang program dikali 2 basis algoritma dari *vocabulary size (n).* Dapat dihitung dengan menggunakan persamaan 3 berikut:

  V = N x log2 (n).....(3)

- *Difficulty Level (D)*

  Tingkat kesulitan atau rawan kesalahan *(D)* program adalah sebanding dengan jumlah angka *operator* unik dalam program *D* juga sebanding dengan rasio antara jumlah total *operand* dan *operand* unik. (jika *operand* yang sama digunakan berkali-kali dalam program, akan lebih rentan terhadap kesalahan). Dapat dihitung dengan menggunakan persamaan 4 berikut:

  D = (n1/2) x (N2/n2) .....(4)

- *Program Level (L)*

  Program Level (L) merupakan kebalikan dari *difficulty level* program. (Program tingkat rendah lebih rentan daripada program tingkat tinggi). Dapat dihitung dengan menggunakan persamaan 5 berikut:

  $L = 1/D$ .....(5)

- *Effort To Implement (E)*

  Effort to implement (E) akan sebanding dengan *Volume program* dan *difficulty Level Program.* Dapat dihitung dengan menggunakan persamaan 6 berikut:

  $E = V \times D$.....(6)

- *Time to Implement (T)*

  Time to implement (T) akan sebanding dengan *effort to implement (E).*

  Halstead telah menemukan pembagian *effort* yakni 18 untuk mewakili waktu setiap detiknya. Dapat dihitung dengan menggunakan persamaan 7 berikut:

  $T = E /18$.....(7)

- *Number of Delivered Bugs (B)*

  Number of Delivered Bugs (B) akan berkorelasi dengan kompleksitas keseluruhan perangat lunak. Dapat dihitung dengan menggunakan persamaan 8 berikut:

  $B = V/ S_0$ ($S_0 = 3000$).....(8)

Contoh perhitungan *Halstead metrics* pada *Psedocode* 1 adalah sebagai berikut:

```
#include <studio.h>
void main()
{int a,b,c,n;
      scanf("%d %d",&a,&b);
      if(a<b){c=a;
   }
   if(c==b){c==a+1;
   }
      else{c=b;
               if(c==b){c=b+1;
        }
   }
n=c;
while(n<=8)
   {if(b>c)
      {c=2;
   }
   else{n=n+c+7;
   }
   n=n+1;
   }
printf("%d%d%d",a,b,n);
}
```

Pseudocode 1. Kode sederhana dalam bahasa C (Debbarma, 2013).

Tabel 1. Kemunculan *operator* pada *Pseudocode* 1

Operator	Kemunculan	*Operator*	Kemunculan
#	1	<	1
Include	1	=	7
stdio.h	1	>	1
Main	1	+	5
(....)	8	<=	1
{...}	8	==	3
Int	1	Printf	1
Scanf	1	If	4
%d	5	While	1
;	10	,	7
"...."	2	&	2
<...>	1	Void	1
Else	2		
n1 = 25		*N1* = 75	

Tabel 2. Kemunculan *operand* pada *Pseudocode* 1

Operand	Kemunculan	*Operand*	Kemunculan
A	5	1	3
B	8	2	1
C	10	7	1
N	8	8	1
n2=8		*N2*=37	

Berdasarkan Tabel 1 dan Tabel 2 didapatkan hasil sebagai berikut:

Vocabulary Size (n) = 33

Program length (N) = 112

Program volume (V) = 1112, 60

Difficulty Level (D) = 57,5

Program level (L) = 0,017

Effort to implement (E) = 63974,5

Time to implement (T) = 3554,13

Number of delivered bug (B) = 0,37

Analisis: berdasarkan perhitungan yang dilakukan, maka diketahui angka *bug* pada *Pseudocode* 1 yakni *B* = 0,37.

Contoh perhitungan *Halstead Metrics* pada *Psedocode 2* adalah sebagai berikut:

```
<?php
//
//--------------------------------------------------
Class CyclomaticComplexityNumber{
      public function example($x,$y)
      {if($x>23||$y<42)
              {for($i=$x; $i>=$x && $i<=$y; ++$i)
        {
        }
   else
   {switch($x+$y)
        {case 1:
```

```
            break;
        case 2:
                break;
        case 3:
                break;
          }
    }
    }
        file_exists('/tmp/log') or touch('/tmp/log');
}
//-------------------------------------------------
//
```

Pseudocode 2. Kode sederhana dalam bahasa PHP (Pdepend.org).

Tabel 3. Kemunculan *operator* pada *Pseudocode* 2

Operator	Kemunculan	*Operator*	Kemunculan
>	1	&&	1
<	1	<=	1
=	1	+	1
>=	1	||	1
{}	5	++	1
n1 = 10		*N1*= 14	

Tabel 4. Kemunculan *operand* pada *Pseudocode* 2

Operand	Kemunculan
X	5
Y	4
I	4
23	1
42	1
n2 = 5	*N2* = 15

Berdasarkan Tabel 3 dan Tabel 4 didapatkan hasil sebagai berikut:

n1 = 10

N1 = 14

n2 = 5

N2 = 15

Vocabulary size (n) = 15

Program length (N) = 29

Program Volume (V) = 42,8

Difficulty level (D) = 15

Program Level (L) = 0,06

Effort To Implement (E) = 642

Time to implement (T) = 35,6

Number of dilevery bug (B) = 0,0014

Analisis : pada perhitungan yang dilakukan pada *Pseudocode* 2, maka diketahui angka *bug* yakni 0,0014.

C. Mc Cabe Cyclomatic Complexity Metrics

Mc Cabe cyclomatic Complexity Metrics adalah *software metrics* yang berfungsi untuk menghitung kompleksitas sebuah program *software*. Kompleksitas ditentukan dengan mengukur jumlah jalur independen melalui program. Semakin tinggi angka yang diperoleh dalam perhitungan maka akan semakin kompleks pula kode programnya. Namun, faktanya bahwa program dengan jumlah *Mc Cabe* yang tinggi (misalkan >10) cenderung sulit dipahami dan memiliki kemungkinan lebih tinggi prediksi kecacatanya (Pranata dkk., 2014).

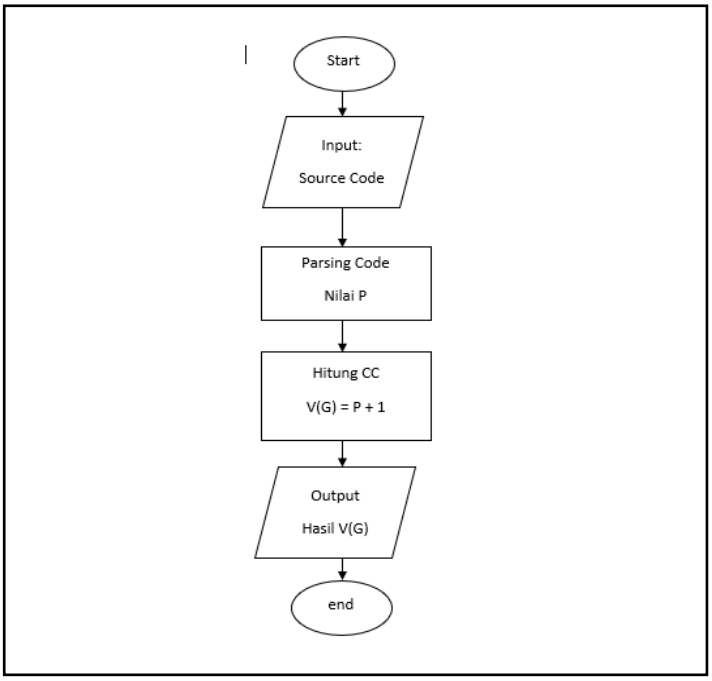
Perhitungan *Mc Cabe Cyclomatic Complexity* yakni menggunakan *control flowgraph* dari program. Berikut merupakan rumus untuk perhitungan *Mc Cabe Cyclomatic Complexity* pada persamaan 9:

Cyclomatic Complexity V(G) = P + 1.....(9)

Dimana:

P = Jumlah *Predikate Node*

Gambar 2 merupakan *flowchart dari Mc Cabe Cyclomatic Complexity Metrics*:



Gambar 2. *Flowchart Mc Cabe Cyclomatic Complexity Metrics* (Pranata dkk., 2016).

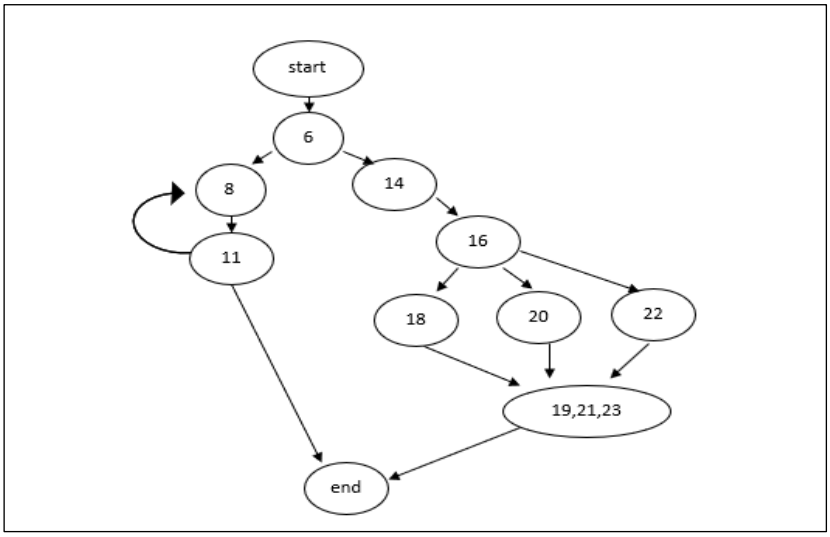
Dari Gambar 2 dapat dilihat bahwa pengukuran dengan metode *Mc Cabe Cyclomatic Complexity Metrics* diawali dengan *input source code* oleh *user*, selanjutnya proses *parsing code* dilakukan oleh sistem untuk mendapatkan nilai P *(predicate Node)* yang kemudian digunakan untuk proses perhitungan *Cyclomatic Complexity (CC)*, pada tahap akhir hasil *Cyclomatic Complexity V(G)* dutampilkan kepada *user.*

Psedeocode 3 merupakan potongan kode program yang digunakan untuk menunjukan contoh perhitungan *Cyclomatic Complexity Metrics.*

```
<?php
//
//--------------------------------------------------
Class CyclomaticComplexityNumber{
      public function example($x,$y)
      {if($x>23||$y<42)
              {for($i=$x; $i>=$x && $i<=$y; ++$i)
        {
        }
   else
   {switch($x+$y)
        {case 1:
           break;
      case 2:
              break;
      case 3:
              break;
        }
   }
   }
      file_exists('/tmp/log') or touch('/tmp/log');
}
//--------------------------------------------------
//
```

Pseudocode 3. Contoh kode sederhana dalam bahasa PHP (Pdepend.org).

Dari *Pseudocode* 3, diperoleh *flowgraph* pada Gambar 3.



Gambar 3. *Flowgraph Pseudocode* 3.

Berdasarkan Gambar 3, diperoleh nilai *Cyclomatiic Complexity Metrics* sebagai berikut:

V (G) = P + 1

V (G) = 4 + 1

V(G) = 5

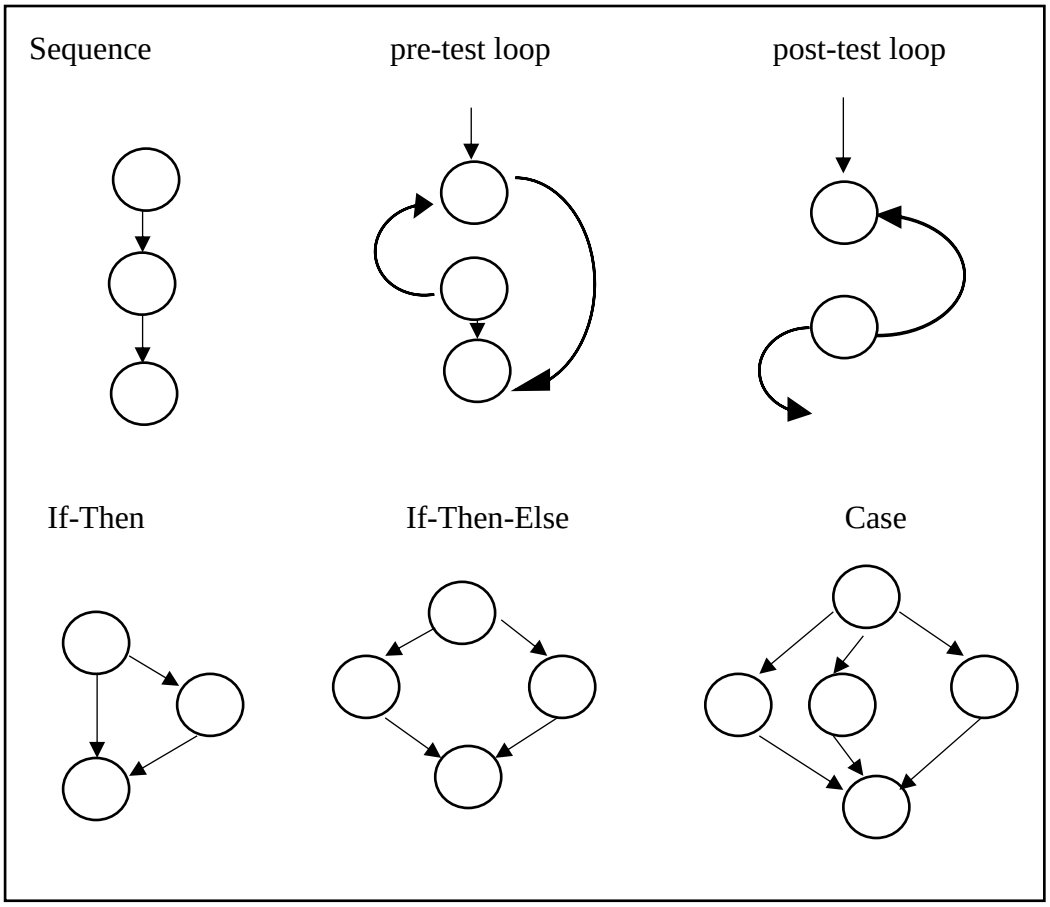
Software Engineering Institute, mengkategorikan evaluasi resiko sesuai dengan *cyclomatic complexity*.

Tabel 5. Besarnya resiko evaluasi *software* berdasarkan *Cyclomatic Complexity Metrics* (Ankita, 2014)

Cyclomatic Complexity	***Risk Evaluation***
1-10	*Software* bebas dari resiko
11-20	*Software* memiliki resiko sedang
21-50	*Software* memiliki resiko tinggi
>50	*Software* sangat beresiko

Berdasarkan Tabel 5, maka contoh kasus pada Gambar 3, memiliki resiko yang rendah untuk pengujian, sehingga modul tersebut layak diuji *defect* secara penuh.

Gambar 4 merupakan notasi diagram alir yang biasa muncul dalam baris kode pemrograman.



Gambar 4. Notasi diagram alir (Nuris, 2015).

D. LOC (Lines Of Code) Metrics

LOC (Lines of Code) metrics merupakan metrik kompleksitas berbasis ukuran. *LOC metrics* menghitung setiap baris kode sumber suatu program tanpa menghitung baris kosong dan komentar (Zhang dkk., 2007).

Tashtoush dkk., (2014) menjelaskan bahwa *LOC (Lines Of Code) Metrics* digunakan untuk mengevaluasi ukuran *software*, namun tidak cukup untuk

mengindikasi kompleksitas *software*. *LOC* dapat dihitung dengan menggunakan cara, sebagai berikut:

- *LOC (Lines Of Code).*
- *SLOC (Source Lines Of Code).*
- *CLOC (Comment Lines Of Code).*
- *S&CLOC (Source Lines Of Code With Comment Lines).*
- *BLOC (Blank Lines Of Code).*
- *PLOC (Physical Lines Of Code).*
- *LLOC (Logical Lines Of Code).*

Jones dalam (Tashtoush dkk., 2014) menjelaskan bahwa ada beberapa metode untuk menghitung *LOC*, yakni:

- Menghitung garis yang dapat dieksekusi saja.
- Menghitung garis deklarasi dieksekusi dan data.
- Menghitung garis deklarasi dieksekusi, komentar dan data.
- Menghitung garis yang dapat dieksekusi, komentar, deklarasi data, dan *Job Control Language (JCL).*
- Menghitung garis pada layar *input* sebagai garis fisik.
- Menghitung garis dengan determinan logis sebagaimana diakhiri.

LOC dihitung dalam metode yang berbeda, sehingga nilainya mungkin berbeda dari satu orang ke orang lain (Tashtoush dkk., 2014)

E. Pemrograman PHP

a. Web Application Framework

Web application framework adalah sebuah kerangka kerja perangkat lunak yang dirancang untuk membantu dalam pembangunan *web* dinamis, aplikasi *web*, w*eb services*, dan *web resources*. Dengan menggunakan sebuah *framework*, proses pembangunan *web* akan menjadi semakin mudah, cepat, dan hemat biaya, karena sebagian besar *framework* telah mengimplementasikan fitur-fitur seperti *Data Persistence, Session Management, User Authentication, Security, Caching,* dan *Adminis trative Interface* (Susanto dkk., 2015).

b. Laravel

Laravel merupakan salah satu *web application framework* yang bersifat *open source*. *Framework* dapat digunakan pada PHP 5 atau diatasnya dan berbasis MVC *(Model View Controller)*. *Laravel* pertama kali dirilis pada 22 Februari 2012, dan versi terbarunya adalah vers i 4.2.11 yang dirilis pada 4 Oktober 2014 (Susanto dkk., 2015).

c. Metrics Tools

Php metric adalah *tool* yang digunakan untuk mengukur *software* metrik pada sebuah program atau projek yang berbasis PHP. *Php metric* dikembangkan oleh Jean-Francois Lepine dan bersifat *open source. Php metric* menyajikan informasi metrik secara lengkap dalam bentuk *chart* dan tabel, dan dilengkapi dengan relasi antar kelas dalam bentuk *relation map* (Susanto dkk., 2015).

Anggreiningsih.,dkk (2017) menggunakan metodologi untuk mengerate *report php metrics tool* dengan langkah sebagai berikut:

- Menginstal *composer.*
- Mendownload phpmetric.phar dan menyimpannya pada lokasi tertentu.
- Menggetikkan perintah pada *command prompt* sebagai berikut

```
phpphpmetric.phar--report-
html=file_report_name.html
location/of/your/sourcecode
```

- *Report* akan digenerate dan disimpan secara otomatis dilokasi tempat phpmetrics.phar disimpan.

d. Operator dan *Operand* pada Bahasa Pemrograman PHP

Operator adalah sesuatu yang menghasilkan nilai dari satu atau lebih data. Sedangkan *operand* yaitu nilai asal yang digunakan oleh *operator*. Berikut ini jenis-jenis *operator* dalam bahasa pemrograman PHP, antara lain:

1. *Operator* Aritmatika *(Aritmatich Operator)*

Tabel 6. *Operator* aritmatika PHP (Tan dan Caroline, 2018)

Operator	**Nama**	**Deskripsi**
$a + $b	Penjumlahan	Menjumlahkan nilai $a dan $b
$a - $b	Pengurangan	Mengurangi nilai $a dengan $b
$a * $b	Perkalian	Mengali nilai $a dengan $b
$a / $b	Pembagian	Membagi nilai $a dengan $b
$a % $b	Modulo	Menampilkan sisa bagi nilai $a dengan $b
$a ** $b	Pangkat	Menampilkan hasil $a pangkat $b (baru diperkenalkan di PHP 5.6)
$a++	*Post-increment*	Penambahan sesudah proses lain dijalankan.
++$a	*Pre- increment*	Penambahan sebelum proses lain dijalankan.
$a- -	*Post-decrement*	Pengurangan sesudah proses lain dijalankan.
--$a	*Pre-increment*	Pengurangan sebelum proses lain dijalankan.

Berdasarkan Tabel 6, *operator* aritmatika yang digunakan dalam bahasa pemrograman PHP terdapat 10 jenis *operator*.

2. *Operator* Perbandingan

Operator perbandingan ditujukan untuk membandingkan dua buah nilai.

Tabel 7. *Operator* perbandingan (Tan dan Caroline, 2018)

Operator	**Nama**	**Deskripsi**
$a == $b	*Equal*	Bernilai *TRUE* jika nilai $a sama dengan $b
$a=== $b	*Identical*	Bernilai *TRUE* jika nilai dan type dari $a dan $b sama
$a != $b	*Not equal*	Bernilai *TRUE* jika nilai $a tidak sama dengan $b
$a <> $b	*Not equal*	Bernilai *TRUE* jika nilai $a tidak sama dengan $b
$a !== $b	*Not identifical*	Bernilai *TRUE* jika nilai dan *type* dari $a dan $b tidak sama
$a < $b	Lebih kecil *(smaller than)*	Bernilai *TRUE* jika nilai $a lebih kecil dari $b
$a > $b	Lebih besar *(greater than)*	Bernilai *TRUE* jika nilai $a lebih besar dari $b
$a <= $b	Lebih kecil sama dengan *(smaller than or equal to)*	Bernilai *TRUE* jika nilai $a lebih kecil atau sama dengan $b
$a >= $b	Lebih besar sama dengan *(greater than or equal to)*	Bernilai *TRUE* jika nilai $a lebih besar atau sama dengan $b

Berdasarkan Tabel 7, *operator* perbandingan yang digunakan dalam bahasa pemrograman PHP terdapat 9 jenis *operator*.

3. *Operator* Logika

Operator logika merupakan *operator* yang digunakan untuk membandingkan dua kondisi logika, yaitu logika benar *(true)* dan logika

salah *(false)*. Nilai yang dibandingkan harus bertipe *boolean* (Rerung, 2018).

Tabel 8. *Operator* logika (Rerung, 2018)

Operator	**Nama**	**Deskripsi**
&&	*And*	Bernilai *TRUE*, jika $a dan $b sama-sama bernilai *TRUE*
||	*Or*	Bernilai *TRUE*, jika salah satu dari $a atau $b bernilai *TRUE*
Xor	*Xor*	Bernilai *TRUE*, jika salah satu dari $a atau $b bernilai *TRUE*, tapi tidak keduanya.
!	*Not*	Bernilai *TRUE*, jika $a bernilai *FALSE*
AND	*Logical and*	Bernilai *TRUE*, jika salah satu dari $a atau $b bernilai *TRUE*
OR	*Or*	Bernilai *TRUE*, jika salah satu dari $a atau $b bernilai *TRUE*

Berdasarkan Tabel 8, *operator* logika yang digunakan dalam bahasa pemrograman PHP terdapat 6 jenis *operator*.

4. *Operator String*

Operator string adalah *operator* yang digunakan sebagai penyambungkan *string (string concatenation). Operator* ini menggunakan karakter titik (.) jika operasi bukan *string*, akan dikonversi secara otomatis (Rerung, 2018).

5. *Operator Assigment*

Operator assigment merupakan *operator* untuk memasukan/menginput sebuah nilai kedalam variabel. *Operator* ini menggunakan karakter sama dengan (=) (Rerung, 2018).

6. *Bitwise Operator*

Tabel 9. *Operator bitwise* (Atkinson dan Suraski, 2004)

Operator	**Nama**	**Contoh**
&	*And*	$a & $b
|	*Or*	$a | $b
^	*Exlusive or*	$a ^$b
~	*One's complement or not*	~$a
>>	*Shift all bits to the right*	$a >> $b
<<	*Shift all bits to the left*	$a << $b

Berdasarkan Tabel 9, *bitwise operator* yang digunakan dalam bahasa pemrograman PHP terdapat 6 jenis *operator*.

7. *Casting Operator*

Tabel 10. *Casting operator* (Atkinson dan Suraski, 2004)

Operator	**Nama**	**Contoh**
(int) (integer)	*Integer cast*	(integer) $i
(float) (double) (real)	*Floting-point cast*	(float) $f
(string)	*String cast*	(string)$s
(bool) (boolean)	*Boolean cast*	(boolean)($a - 3)
(array)	*Array cast*	(array) $c

(object)	*Object cast*	(object) $a

Berdasarkan Tabel 10, *casting operators* yang digunakan dalam bahasa pemrograman PHP terdapat 10 jenis *operator.*

8. *Miscellaneous Operators*

Tabel 11. *Miscellaneous operators* (Atkinson dan Suraski, 2004)

Operator	**Nama**	**Contoh**
.	*Concatenate*	$a . $b
$	*Indirect reference*	$$a
@	*Silence (suppress error message)*	@($a/$b)
? :	*Ternary conditional expression*	($a == 3) ? “yes” : “no”
{}	*Variable embedded in a string*	{$a}
‘’	*Execute a string in the command shell*	‘1s - 1’
=>	*Assign array element index*	Array (1=>’january’)
->	*Reference an object*	$c ->method()
::	*Reference a class*	myClass::method()
Instanceof	*Tests if an object is an instance of a certain class*	$c instanceof myClass

Berdasarkan Tabel 11, *Miscellaneous operators* yang digunakan dalam bahasa pemrograman PHP terdapat 10 jenis *operator.*

F. Penelitian Terdahulu

a. Penelitian tentang *Halstead Metrics & Mc Cabe Cyclomatic Complexity Metrics*

Susanto dkk., (2015) mengungkapkan bahwa pada penelitian yang dilakukan dibangun aplikasi *web Jurnal Logic* menggunakan *framework Laravel,* kemudian dilakukan pengukuran *software metric* terhadap aplikasi web tersebut menggunakan tools *Php Depend* dan *Php Metric*. Dari penelitian ini didapatkan bahwa nilai *Efficiency, Complexity, Understandability, Reusability,* dan *Maintainability* produk dapat ditingkatkan dengan membuat *subclass,* memisahkan fungsi kedalam kelas secara spesifik, membuat algoritma yang lebih sederhana, dan menambahkan komentar pada kode. *Complexity* pada suatu produk *software* dipengaruhi oleh nilai *Cyclomatic Complexitty*. Nilai *Complexity* dapat ditingkatkan dengan membuat *algoritma* yang lebih sederhana.

Debbarma dkk., (2013) mengungkapkan bahwa metrik kompleksitas *software* memiliki kecenderungan untuk digunakan dalam menilai kualitas pengembangan perangakat lunak dan salah satu bagian penting dari SLDC. *Volume,* kontrol, dan kompleksitas berbasis data adalah pentingnya sistem *software* saat ini menuntut penerapan teknik pengujian yang efektif. Selain itu, diamati bahwa metrik kompleksitas *software* memungkin penguji untuk menghitung jalur eksekusi asiklik melalui program dan meningkatkan kualitas *software*. Analisis statis ini dapat mengarah pada pengurangan biaya pengembangan *software* dan meningkatkan keandalan pengujian dan kualitas

software dengan mengevaluasi metrik kompleksitas seperti *LOC, NPATH(NC), Mc Cabe Cyclomatic Complexity, Halstead Metrics.*

b. Defect dan *Bugs*

Menurut Vipindeep dan Jalote (2005) s*oftware bugs* dapat didefinisikan sebagai bagian dari kode yang akan menghasilkan *error, failur* atau tidak berfungsinya program. Menurut standar IEEE, bugs adalah langkah, instruksi atau data yang salah dalam suatu program. Kegagalan disebabkan karena *bug* dan dapat mengubah perilaku eksternal program. Secara umum *bugs* dapat diklasifikasikan berdasarkan frekuensi kemunculan *bug* dan tingkat keparahan *bug.* Tingkat keparahan *bug* dapat dilihat dalam uraian sebagai berikut:

1. *Catastrophic : Defect* yang mengakibatkan dampak yang sangat serius (fungsi hilang, masalah keamanan, dan lain-lain).
2. *Major : Defect* yang dapat menyebabkan konsekuensi serius bagi sistem, seperti kehilangan beberapa data.
3. *Minor : Defect* yang dapat menyebabkan konsekuensi kecil bagi sistem, seperti menampilkan hasil dalam format yang berbeda-beda.
4. *No effect : Defect* yang tidak menghambat kinerja sistem, namun memberikan interpretasi yang sedikit berbeda, seperti kesalahan penulisan dalam dokumentasi program.

Secara umum *bugs* dapat dikelompokan sebagai berikut:

1. *Bugs* yang disebabkan oleh *pointer* dan memori.

2. *Bugs* yang disebabkan oleh *Aliases.*
3. *Bugs* yang disebabkan oleh kesalahan sinkronisasi.
4. *Bugs* yang disebabkan oleh beberapa pola *bug* umum yang berhubungan dengan sinkronisasi.
5. *Bugs* yang disebabkan oleh kesalahan data dan aritmatika.
6. *Bugs* yang disebabkan oleh kerentanan keamanan.
7. *Bugs* yang disebabkan oleh operasi yang berlebihan.

Menurut Chillarege, dkk (1992) *defect* dapat dikelompokan dalam beberapa jenis. *Defect function* yang berarti kesalahan yang memengaruhi kemampuan signifikan, antarmuka pengguna akhir, antarmuka produk, antarmuka dengan arsitektur perangkat keras, atau struktur data global dan harus memerlukan perubahan desain formal. *Defect interface* merupakan kesalahan dalam berinteraksi dengan komponen lain,modul, *call statement, control block,* dan daftar parameter. *Defect checking* merupakan kesalahan dalam memeriksa alamat logika program yang gagal memvalidasi data dan nilai sebelum digunakan. *Defect assignment* merupakan kesalahan beberapa baris kode, seperti inisialisasi blok kontrol atau struktur data. *Defect timing/serilization* merupakan kesalahan dalam pengelolaan sumber daya. *Defect build/package/merge* merupakan kesalahan yang terjadi dalam sistem *library,* perubahan manajemen atau versi kontrol. *Defect documentation* merupakan kesalahan yang dapat mempengaruhi pubikasi dan catatan pemeliharaan. *Defect algorithm* termasuk efisiansi penerapan algoritma atau struktur data tanpa mengurangi perubahan *design.*

c. Korelasi

Korelasi adalah salah satu teknik statistik yang digunakan untuk mencari hubungan antara dua variabel atau lebih (Anitawidanti 2010). Uji korelasi yang digunakan dalam penelitian ini adalah uji hipotesis *paired t-test*, uji koefisien korelasi *Pearson* dan uji koefisien korelasi *Spearman.*

Montolalu dan Langi (2018) mengungkapkan bahwa uji *paired t-test* adalah salah satu metode pengujian hipotesis data yang berpasangan. Uji *paired t-test* dapat dilakukan dengan persamaan 10.

$$t_{hit} = \frac{\overline{D}}{\frac{SD}{\sqrt{n}}} \quad \text{.....(10)}$$

Dimana

$$SD = \sqrt{var}$$

$$var(s^2) = \frac{1}{n-1}\sum_{i=1}^{n}(x_{i-\bar{x}})^2$$

Persamaan 11 merupakan Uji korelasi *Pearson* (Suparto, 2014).

$$r = \frac{n\sum xy - (\sum x)(\sum y)}{\sqrt{\{n\sum x^2 - (\sum x)^2\}\{n\sum y^2 - (\sum y)^2\}}} \quad \text{.....(11)}$$

Dimana :

r = koefisien korelasi pearson

x = nilai variabel pertama

y = nilai variabel kedua

n = jumlah data

Dasar penggunaan korelasi *Spearman* adalah rengking (peringkat) Anitawidanti (2010). Uji korelasi *Spearman* dapat dilakukan dengan menggunakan persamaan 12 berikut:

$$\rho = \frac{1-6\sum D^2}{\mathrm{n}(n^2-1)} \quad \text{.....(12)}$$

Dimana:

p = koefisien korelasi spearman (rho)

D = perbedaan skor antar dua variabel

n = jumlah kelompok

d. Prediksi Cacat *Software*

McDonald dkk. dalam (Sabarudin, 2017) mengungkapkan bahwa istilah cacat atau *bug* mengacu pada menifestis dari kesalahan dalam kode sumber (source code), dimana kesalahan adalah suatu tindakan yang keliru yang dibuat oleh pengembang. Cacat *software* adalah kekurangan/cacat pada sebuah *software* karena melakukan proses yang tidak terduga.

McDonald dkk. dalam (Sabarudin, 2017) menjelaskan bahwa cacat *software* dapat terjadi karena hal, sebagai berikut:

a. Akibat kesalahan manusia, kesalahan ini terjadi saat programmer melakukan kesalahan saat pengetikan tanpa disadari.
b. kesalahan sistematik selama pengembangan, adanya kesalahan suatu proses yang keluaranya digunakan untuk proses yang lain.

c. Akibat kesalahan penggunaan peralatan pengembang.
d. Salah pengertian terhadap kebutuhan *customer*.

Lebih lanjut, McDonald dkk. dalam (Sabarudin, 2017) kualitas *software* dan tingkat cacat dapat diatasi dengan tiga tingkat yaitu:

a. Deteksi, yaitu melakukan pengujian terhadap *software* sampai menemukan cacat yang mungkin terjadi dan memperbaikinya.
b. Analisa, dengan menganalisa cacat sebelumnya untuk mengetahui kecendrungan dan memahami kenapa bisa terjadi dan tidak terdeteksi.
c. Pencegahan, dilakukan secara proaktif mencari dan menghilangkan potensi cacat *software.*

Suffian dan Ibrahim (2012) mengungkapkan bahwa prediksi kecacatan adalah proses proaktif untuk mengkarakterisasi banyak jenis cacat yang ditemukan dalam konten, desain, dan kode *software* dalam menghasilkan produk berkulaitas tinggi. LOC *Metrics* dan *Mc Cabe Cyclomatic Complexity Metrics* digunakan untuk memprediksi kecacatan *software*. Dapat dihitung dengan menggunakan persamaan 13 dan 14 berikut:

$$Defect = 4.86 + 0.018\ Lines\ of\ Code \quad \text{.....(13)}$$

$$Defect = 4.2 + 0.0015\ Lines\ of\ Code^{4/3} \quad \text{.....(14)}$$

e. Penelitian yang Berfokus pada Memprediksi Kecacatan *Software*

Sehgal dan Mehrotra (2015) mengungkapkan bahwa untuk memprediksi kesalahan *(faults)* dapat menggunakan dua tahapan yakni sebagai berikut:

(i) memprediksi kesalahan pada fase pengembangan awal (sebelum perancangan).

Pada tahap ini, dapat dilakukan dengan memprediksi probabilitas panjang kode program dan kegunaan dari komponen. Dalam menentukan peluang tingkat keandalan sebuah *software,* Seghal dan Mehrotra mengusulkan produk tingkat keandalan *Cartesian,* misalnya *Lr = {high, medium, low},* dimana *Lr* merupakan himpunan tingkat keandalan *(realibily).* Untuk menetukan tingkat keandalan suatu komponen didefinisikan dengan *Lv = {high,low}.* Dimana *Lv* merupakan himpunan *levels volume*. Dengan menggunakan *Lv,* akan diketahui komponen mana yang rentan terhadap kesalahan. Dengan demikian keandalan *(realibility)* komponen adalah kombinasi *levels volume (Lv)* dan *level* penggunaan *(Lu).* Pemetaan *level volume* dan level penggunaan ke keandalan dengan fungsi *fr,* Dapat dihitung dengan menggunakan persamaan 15 berikut:

$$Fr = \frac{1}{Lv}\ x\ Lu \longrightarrow Lr \quad \text{.....(15)}$$

Pada Tabel 12 menunjukkan fungsi pemetaan semua komponen antara level *volume* ke level pengguna.

Tabel 12. Prediksi keandalan fase *design*

Volume	***Usage***	***Reliability***
High	* *high*	*Low*
High	* *medium*	*Low*
High	* *low*	*High*
Low	* *high*	*Low*
Low	* *medium*	*Medium*
Low	* *low*	*Low*

(ii) Memprediksi kesalahan pada fase pengembangan awal (setelah perancangan).

Langkah 1 : mengambil 10 komponen bersifat interaktif dan independen yang memiliki 20.000 *LOC.*

Langkah 2 : menghitung *volume* pada masing-masing komponen dengan *Halstead Metrics* dengan menggunakan persamaan 16 berikut:

$$V = N \log_2 (n1+n2) \quad \text{.....(16)}$$

Dimana $N = N1 + N2$. *N1* adalah total kemunculan *operator* dan *N2* total kemunculan *operand, n1* adalah jumlah *operator* unik dan *n2* adalah jumlah *operand* yang unik. *(n1 + n2)* merupakan *vocabolary size.*

Langkah 3 : menghitung kesalahan *(fault)* menggunakan *Halstead Metrics* dengan menggunakan persamaan 17 berikut:

$$Fault\ B = V/S_0 \quad \text{.....(17)}$$

Dimana *V* adalah *Volume* dan S_0 merupakan jumlah rata-rata *(desicions),* $S_0 = 3000$.

Langkah 4 : menghitung MTBF yaitu waktu rata-rata kegagalan. Dapat dihitung dengan menggunakan persamaan 18 berikut:

$$MTBF = \frac{mission\ times + sample\ population}{failur\ with\ in\ mission\ time} \quad \text{.....(18)}$$

Dimana *mission time* adalah waktu untuk pengumpulan data *reliabilitas, sample population* adalah jumlah komponen yang diambil, dan *failur with in mission time* adalah jumlah kegagalan.

Langkah 5 : *reliabilitas* adalah *probabilitas* bahwa suatu sistem akan bekerja tanpa kegagalan untuk jangka waktu tertentu. Nilai *reliabilitas* terletak antara 0 dan 1 yang dapat dihitung dengan persamaan 19 berikut :

$$Reliability = e^{-planned\ service\ life/MTBF} \quad \text{.....(19)}$$

Dimana *service life* adalah jumlah waktu suatu perangkat diperbaiki, atau lama operasi yang diharapkan sebelum suatu perangkat gagal.

Priyambadha dan Rochimah (2013) mengungkapkan bahwa kloning, *Cyclomatic Complexity Metrics, Halstead Metrics* mampu memprediksi kecacatan sebuah *software* proses kerja yang dilakukan

dalam penelitian ini adalah dengan mengumpulkan data dari *project* pengembangan *software* ArgoUML, Ant, jEdit dan jHotDraw, yang selanjutkan dilakukan pendefinisian karakteristik menggunakan perangkat pembantu bernama SonarQube. Kemudian dilakukan perhitungan *regresi* dan *korelasi* data dengan menggunakan pendekatan analisis. Proses terakhir yakni analisa hasil yang dilakukan dengan melihat hasil perbandingan *kooefision* determinasi tiap-tiap kombinasi variabel data. Penelitian ini menghasilkan kesimpulan bahwa gabungan antara kloning, kompleksitas dan *LOC* memiliki pengaruh paling tinggi terhadap terjadinya cacat. Nilai tersebut mencapai rata-rata 95% terhadap terjadinya cacat.

G. Pengujian Manual

Pengujian manual dilakukan pada tahap awal pengembangan *software*, seperti pada tahap analisis. Dalam melakukan pengujian manual yakni dengan menyusun *tase case* (Sharma dan Sangwhan, 2016).

1. White Box Testing

Pada pengujian *white box, test case* didasarkan pada implementasi entitas *software*. Merancang *test case* untuk menguji fungsi internal *software* dari perspektif pengembang. *White box testing* berfokus pada logika internal dan struktur kode sumber. *White box testing* digunakan untuk mendeteksi kesalahan logis dalam kode program. pengujian ini diterapkan pada *unit testing dan integration testing* (Nindra dan Dodeti, 2012).

a. Unit Testing

Unit testing merupakan pengujian yang dilakukan dilevel terendah. *Unit testing* yakni pengujian modul atau komponen. *Unit testing* ditujukan untuk memverifikasi fungsionalitas bagian kode tertentu, biasanya pada tingkat fungsi program (Ghuman, 2014).

b. Integration Testing

Pengujian dilakukan pada kedua antar kedua antara komponen dan struktural yang lebih besar yang sedang dibangun. Pengujian integrasi adalah semua jenis pengujian *software* yang berupaya memverifikasi antarmuka antar komponen dengan desain *software* (Ghuman, 2014).

c. System Testing

System testing dilakukan untuk pengujian keseluruhan sistem. Pengujian ini didasarkan pada spesifikasi fungsional dan kualitas non-fungsional (Ghuman, 2014).

III. METODOLOGI PENELITIAN

A. Tempat dan Waktu

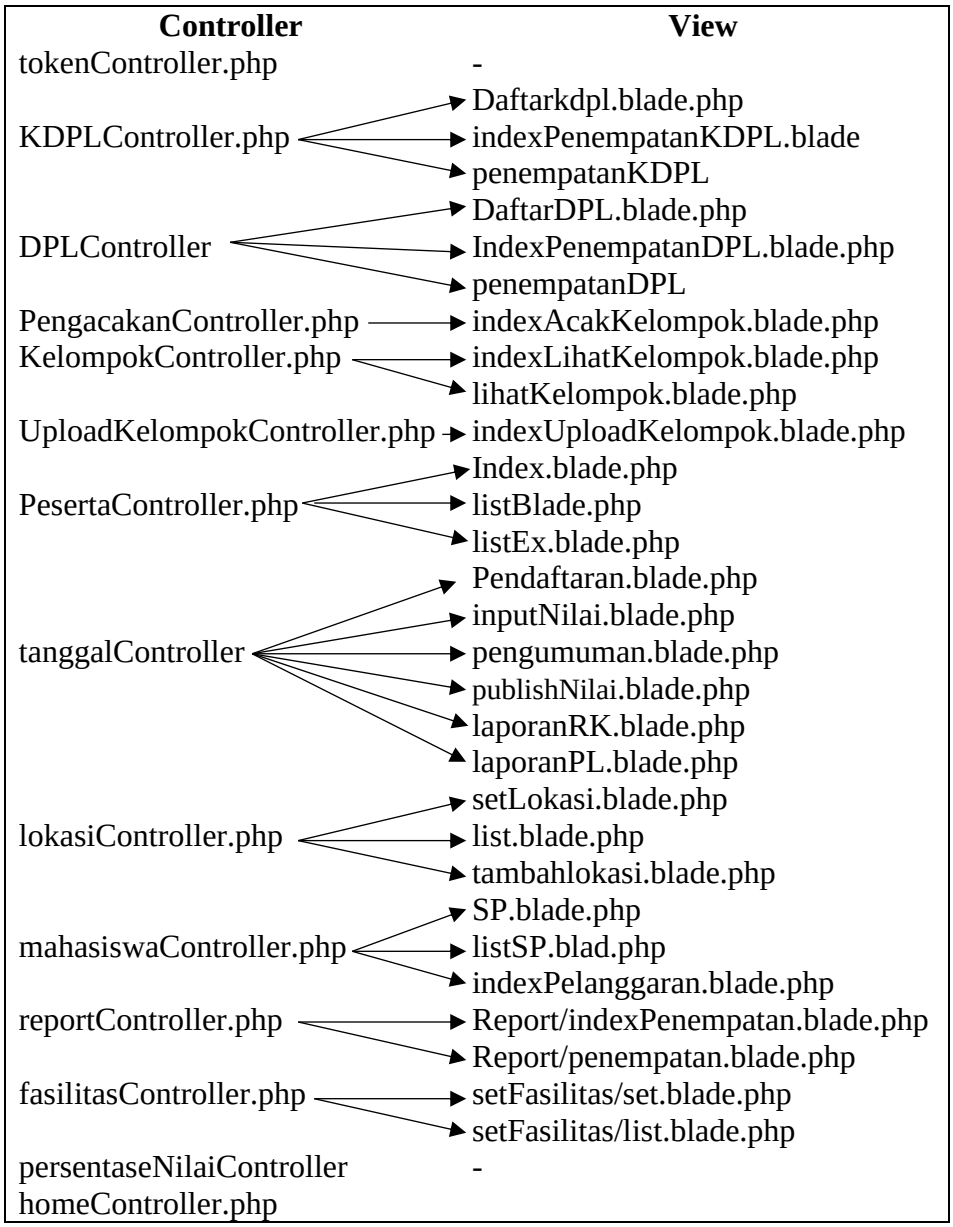
Penelitian dilakukan di Gedung Ilmu Komputer Fakultas Matematika dan Ilmu Pengetahuan Alam (FMIPA). Waktu penelitian dilaksanakan pada semester genap 2018/2019.

B. Data dan Alat

1. Data

Adapun data primer yang digunakan, sebagai berikut:

a. *Source Code* Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung yang dibangun oleh Ichwan Almaza (2018) pada modul *Administrator,* seperti yang dirangkum dalam Gambar 5.



Gambar 5. Daftar nama file source code Sistem Informasi KKN Universitas Lampung.

b. *Database* kkn.sql

2. Alat

Adapun alat yang digunakan dalam penelitian ini, sebagai berikut:

a. Perangkat Keras *(Hardware)*

Perangkat keras yang digunakan dalam penelitian ini adalah 1 unit Notebook dengan spesifikasi:

- *Processor* : Intel(R) Celeron(R) CPU N3050 @1,60 GHz, Core 2, Threads 2, Cache 2 MB L2.
- *Installed memory (RAM)* : 2,00 GB DDR3-1600 MHz.
- *Hard drive* : 500 GB SATA-II 5400 RPM

b. Perangkat Lunak *(Software)*

- Sistem Operasi *Windows* 10 Pro 64 bit

  Sistem operasi berfungsi sebagai penghubung antara lapisan *software* dan lapisan *hardware.*
- *PHP Metrics Tool* versi 2.4.1

  *Tool PHP yang d*igunakan untuk menghitung *Cyclomatic Complexity Metrics, Halstead Metrics, LOC Metrics.*
- XAMPP versi 3.2.2

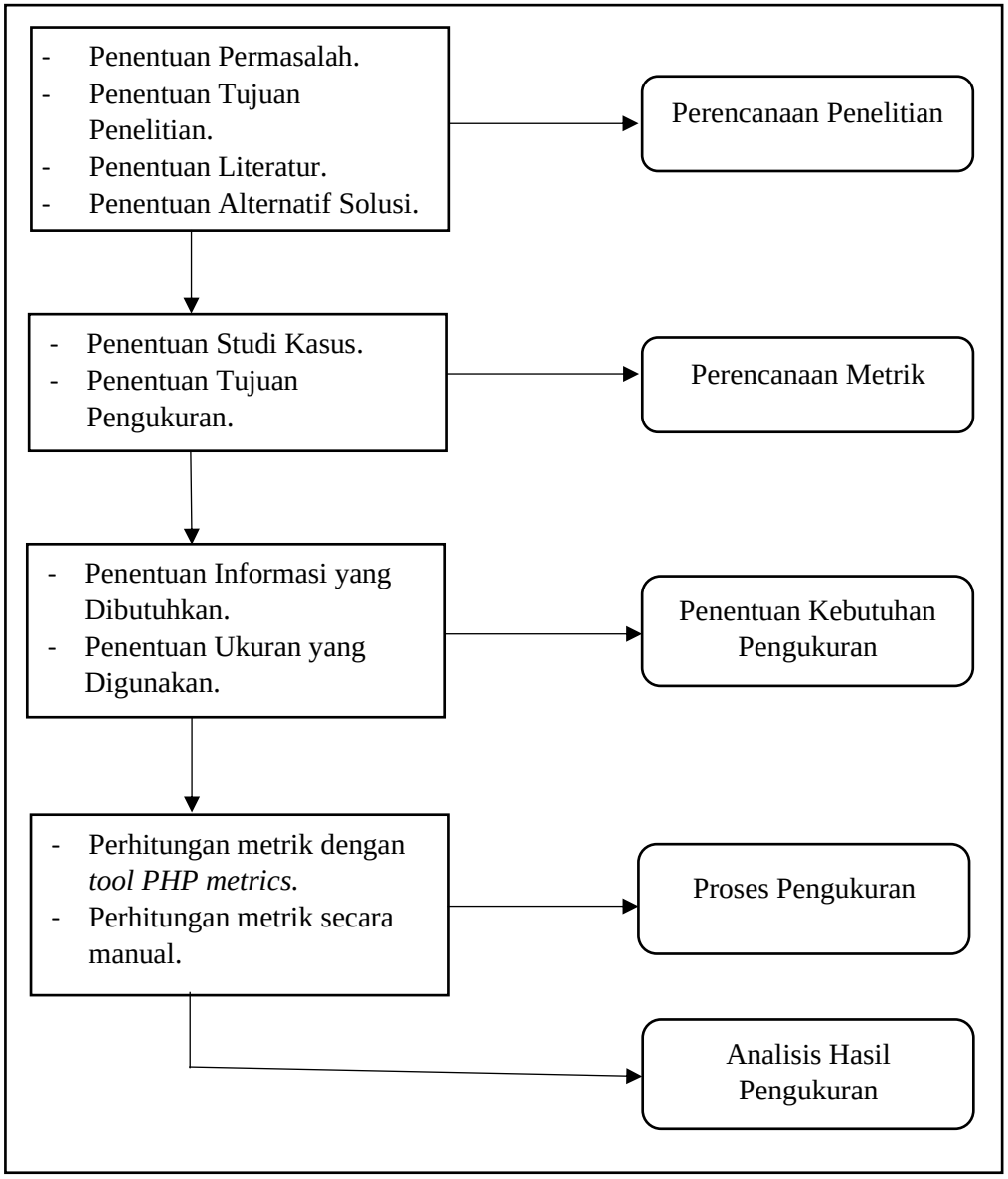
  XAMPP digunakan untuk membuat *web server* pribadi, sehingga program yang sudah dibuat dapat dijalankan di *localhost.*
- Sublime Text 3 versi 3.2.1

  Sublime Text 3 adalah *software* penyunting teks dan penyunting kode sumber yang berjalan pada sistem operasi *windows.*

- *Web Browser (Google Chrome* versi 75.0.3770.142*)*

  Web Browser (Google Chrome) adalah *compailer* bahasa HTML.

C. Metode Penelitian



Gambar 6. Aliran proses pengerjaan penelitian.

Gambar 6 merupakan proses perencanaan metrik berfokus pada tujuan yang ingin dicapai setelah melakukan penelitian. Pada tahap aktifitas metrik merupakan langkah-langkah dalam mencapai tujuan, pada tahapan ini terdiri

dari mencari detail informasi yang diperlukan, menentukan ukuran yang akan digunakan, dan memilih metrik yang akan digunakan dalam penelitian.

Tahapan selanjutnya adalah proses pengukuran, yang dilakukan dengan mengukur menggunakan *PHP metrics tool* dilanjutkan dengan menghitung secara manual. Setelah proses pengukuran selesai, maka akan dilakukan tahapan terakhir, yaitu analisis hasil pengukuran. Selanjutnya, akan dijabarkan dalam uraian berikut:

1. Perencanaan Penelitian

a. Penentuan Permasalahan

Pada penelitian ini, hal yang menjadi permasalahan yakni tentang analisis perhitungan kecacatan (*defect*) pada Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung. Seperti yang diketahui, bahwa pengujian membutuhkan sumberdaya yang tinggi. penelitian ini dilakukan untuk memperkirakan modul-modul yang rawan kecacatan (*defect*), sehingga dapat digunakan sebagai acuan untuk perbaikan/peningkatan Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung, dengan mempertimbangkan sumber daya yang dimiliki.

b. Penentuan Tujuan Penelitian

Pada penelitian ini, adapun tujuan yang akan dicapai yakni mengkonfirmasi jumlah *defect*, tempat terjadinya *defect* dan evaluasi

resiko *software* sehingga dapat diketahui sumber daya yang diperlukan untuk melakukan pengujian pada Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung.

c. Penentuan Literatur

Literatur dijabarkan pada bab II.

d. Penentuan Alternatif Solusi

Dalam mendeteksi *defect*, digunakan implementasi perhitungan *software metrics* seperti *Cyclomatic Complexity Metrics, Halstead Metrics, dan Line Of Code (LOC) Metrics.* Penelitian dilakukan dengan perhitungan secara manual dan mengkonfirmasi dengan perhitungan *PHP metrics tool.* Hasil dari penelitian ini adalah diketahui seberapa besar sumber daya yang diperlukan dalam pengujian Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung.

2. Penentuan Perencanaan Metrik

Pengukuran dilakukan dengan menggunakan tiga metrik yakni *Halstead Metrics, Cyclomatic Complexity Metrics,* dan *LOC (Line Of Code) Metrics.* Pada pengukuran *Halstead Metrics* dilakukan dengan menganalisis *operator* dan *operand* program sehingga diketahui nilai *bug* pada setiap fungsi. Pada *Cyclomatic Complexity Metrics* pengukuran dilakukan dengan analisis *flowgraph* guna untuk mengetahui besarnya resiko evaluasi pada setiap fungsi. Pengukuran *LOC Metrics* dilakukan

analisis panjang kode sumber guna untuk mengetahui perkiraan jumlah *defect* pada sistem.

3. Penentuan Kebutuhan Pengukuran

a. Detail Informasi yang Dibutuhkan

Pengukuran dilakukan pada Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung dalam modul *Administrator* Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung berbasis *web* PHP. Seperti yang telah ditulis pada Tabel 13.

b. Ukuran yang Digunakan

Ukuran yang digunakan dalam penelitian ini adalah ukuran kompleksitas, panjang baris dan tingkat kecacatan pada Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung. Kompleksitas dan tingkat kecacatan akan menghasilkan sebuah ukuran kualitas pada aplikasi tersebut.

Ukuran pada *Mc Cabe Cyclomatic Complexity Metrics*, yakni *V (G).* Ukuran dalam *Halstead Metrics* yakni panjang program (N), *vocabulary size (n), program volume (V), difficulty level (D), program level (L), efford to implement (E), time to implement (T),* dan *number of delivered bugs (B).* Ukuran dalam *LOC (Line Of Code) metrics* yakni jumlah baris kode sumber.

4. Proses Pengukuran

Perhitungan Menggunakan *PHP Metrics Tools*

Source code dari modul *Administrator* dianalisis menggunakan *phpmetrics.* Hasil analisis dengan *phpmetric* berupa *report file html*, dan untuk mengenerate *report* tersebut dilakukan dengan cara:

- Menginstal *composer.*
- *Mendownload* phpmetric.phar dan menyimpannya pada lokasi tertentu.
- Menggetikkan perintah pada *command prompt* sebagai berikut

```
Phpphpmetric.phar--
reporthtml=file_report_name.html
location/of/your/sourcecode
```

- *Report* akan digenerate dan disimpan secara otomatis dilokasi tempat phpmetrics.phar disimpan.

Perhitungan Manual

a. Proses pengukuran menggunakan *Mc Cabe Cyclomatic Complexity Metrics*:

- Membuat *flowgraph* dari kode sumber program, *flowgraph* dibuat berdasarkan urutan angka pada baris kode program..
- Menganalisis *flowgraph* yang telah dibuat.
- Menghitung jumlah *predikat node.*
- Menghitung *Cyclomatc Complexity (CC)* dengan menggunakan rumus *V(G) = P+1*

- Menganalisis resiko evaluasi pada *software.*

b. Proses pengukuran menggunakan *Halstead Metrics:*

- Menganalisis *operator dan operand* pada bahasa pemrograman PHP.
- Menghitung jumlah *operator* yang berbeda *(n1)* dan jumlah *operand* yang berbeda *(n2)* dari kode sumber.
- Menghitung total jumlah kemunculan *operator (N1)* dan total jumlah kemunculan *operand (N2)* dari kode sumber.
- Menghitung panjang program *(N) = N1 + N2.*
- Menghitung *vocabolary size (n) = n1 + n2.*
- Menghitung *program volume (V) = N x log2 (n).*
- Menghitung *difficulty Level (D) = (n1/2) x (N2/n2).*
- Menghitung *Program Level (L) = 1/D.*
- Menghitung *Effort to Implement (E) = V x D.*
- Menghitung *time to implement (T) = E/18.*
- Menghitung *Number of delivery bugs (B) = V/S*$_{0.}$
- Pada pengukuran *Halstead Metrics* didapatkan *Number of Delivered Bugs (B)* yang menjadi nilai akhir dari *Halstead Metrics.*

c. Proses pengukuran *LLOC (Logical Lines of Code) Metrics*

Pengukuran dengan menggunakan *LLOC (Logical Lines of Code) Metrics,* yakni sebagai berikut:

- Baris komentar tidak dihitung.
- Baris spasi tidak dihitung.

- Baris *return* tidak dihitung.

d. Perhitungan *Defect*

Defect bisa dihitung dengan menggunakan persamaan berikut:

$$Defect = 4.86 + 0.018\ Lines\ of\ Code$$

$$Defect = 4.2 + 0.0015\ Lines\ of\ Code^{4/3}$$

5. Analisis Hasil Pengukuran

1) Membandingkan jumlah *defect* pada perhitungan dengan menggunakan *tool* dan perhitungan manual.
2) Membandingkan tempat munculnya *defect* antara perhitungan menggunakan *tool* dan perhitungan manual.
3) Mengkonfirmasi sumber daya yang diperlukan dalam pengujian setelah evaluasi resiko *software* diketahui.

V. SIMPULAN DAN SARAN

A. Simpulan

Berdasarkan penelitian analisis perhitungan *defect* pada Sistem Informasi Kuliah Kerja Nyata (KKN) menggunakan *LLOC Metrics, Cyclomatic Complexity Metrics,* dan *Halstead Metrics,* maka didapatkan kesimpulan sebagai berikut.

1. Indikator *defect* pada Sistem Informasi Kuliah Kerja Nyata (KKN) dipengaruhi oleh *defect functional* dan besarnya *bugs* kode sumber.
2. Dalam pengukuran *Cyclomatic Complexity Metrics* dengan membuat *flowgraph* program, pengukuran *Halstead Metrics* dilakukan dengan menghitung jumlah *operator* dan *operand* kode program, pengukuran dengan *LLOC Metrics* dengan menghitung jumlah baris logika kode program, perhitungan *defect* dilakukan dengan *LLOC Metrics.* Pada perhitungan *defect* dilakukan pengujian korelasi antara variabel *defect* dan variabel *LLOC* dengan uji hipotesis *t-test,* uji *pearson* dan uji *spearman.* Hasil dari pengujian korelasi menyatakan bahwa semakin tinggi nilai *LLOC* maka akan semakin besar nilai *defect.*
3. Perhitungan *Cyclomatic Complexity Metrics* menghasilkan ukuran kompleksitas kode program sehingga bisa diketahui sumber daya yang dibutuhkan dalam pengembangan *software*, perhitungan *Halstead Metrics*

menghasilkan ukuran *bugs* kode progam, pengukuran *LLOC Metrics* menghasilkan ukuran besarnya logika setiap baris kode sumber untuk digunakan dalam memprediksi *defect*.

4. Kemunculan *defect* tertinggi yakni pada file pengacakanController.php, sehingga jika akan melakukan perbaikan/pengembangan fungsi, file pengacakanController.php perlu diwaspadai karena merupakan file yang memiliki kompleksitas tinggi sehingga rawan cacat.
5. Perhitungan *Cyclomatic Complexity Metrics* diperoleh nilai rata-rata yakni 20,2, dimana angka ini berarti modul *Administrator* Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung memiliki resiko sedang untuk pengujian.
6. Estimasi sumber daya (waktu) yang dibutuhkan untuk melakukan perbaikan/pengembangan fungsi sistem yakni 1088 menit atau sama dengan 18,13 jam atau sama dengan 2 ¼ hari (waktu yang diperlukan untuk pengujian dan memahami kode sumber Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung yang lama). Sedangkan untuk melakukan penambahan fungsi pada Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung membutuhkan estimasi waktu ±30 hari. Estimasi sumber daya (biaya) yang dibutuhukan untuk memperbaiki/mengembangkan Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung adalah sebesar Rp. 3.851.096/orang. Perhitungan didasarkan oleh besarnya UMK Lampung. Estimasi sumber daya (waktu) yang dibutuhkan untuk melakukan pembangunan ulang Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung

membutuhkan waktu 24 hari s/d 180 hari dan estimasi sumber daya (biaya) yang dibutuhkan untuk membangun ulang Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung adalah sebesar Rp. 1.956.112/orang s/d Rp.14.670.846/orang. (Perhitungan berasal dari UMK Lampung berdasarkan keputusan Gubernur Lampung No. G/552/V.07/HK/2018 Tahun 2018)

B. Saran

Berdasarkan penelitian yang telah dilakukan, maka didapatkan sebagai berikut:

1. *Cyclomatic Complexity Metrics*, *Halstead Metrics*, *LLOC Metrics* dapat direkomendasikan untuk menghitung *defect* pada *software* lainya berdasarkan tingkat kompleksitas dan *bugs* kode sumber.
2. Untuk memperbaiki/menambahkan fungsi pada Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung, *programmer* harus berhati-hati pada file pengacakanController.php karena memiliki angka *defect* yang tinggi. Selain itu juga pada file pengacakanController.php memiliki nilai *Cyclomatic Complexity* yang tinggi, sehingga pengujian bisa dilakukan lebih banyak pada file pengacakanController.php atau dengan melakukan perbaikan secara berhati-hati pada pengacakanController.php.
3. Melihat tingginya tingkat kompleksitas pada *source code controller* dan *view* modul *Administrator*, maka disarankan agar *programmer* membuat algoritma yang optimal untuk mengurangi tingkat kompleksitas dari *source code.*

DAFTAR PUSTAKA

Almaza, I. 2018. *Sistem Informasi Kuliah Kerja Nyata (KKN) Universitas Lampung Berbasis Web Menggunakan Framework Laravel.* Lampung: Univesitas Lampung.

Angrainingsih, R., Suryono, A. D., & Rohmawati, U. A. 2017. Perbandingan Maintainability, Fleksibility,Testability Pada Cms Open Source E-Commerce. *InfoTekJar(Jurnal Nasional Informatika dan Teknologi Jaringan)*, 10-15.

Anitawidanti, H. 2010. *Analisis Hubungan Antara Stres Kerja dengan Kepuasan Kerja Karyawan Berdasarkan Gender Studi pada PT.TRANSINDO Surya Sarana Semarang.* Semarang: Universitas Diponegoro.

Ankita. 2014. *A Framework for Improving the Concept of Cyclomatic Complexity in Object Oriented Programming.* Patiala: Thapar University.

Atkinson, L., & Suraski, Z. 2004. *Core PHP Programming.* New Jersey: Prentice Hall Professional Technical Reference.

Chhabra, P. 2014. An Effective Implementation of Improved Halstead Metrics for Software Parameters Analysis. *International Journal of Computer Science and Mobile Computing*, 146-161.

Chillarege, R., Chaar, J., & Ray, B. K. 1992. Ortoghonal Defect Classification - A Concept for In-Process Measurements. *IEEE Transactions on Software Engineering*, 943-956.

Debbarma, M. K., Debbarma, S., Debbarma, N., Chakma, K., & Jamatia, A. 2013. A Review and Analysis of Software Complexity Metrics in Structural Testing. *International Journal of Computer and Communication Engineering*, 129-133.

Ghuman, S. S. 2014. Software Testing Techniques. *International Journal of Computer Science and Mobile Computing*, 988-993.

Kan, S. H. 2003. *Metrics and Models in Software Quality Engineering 2nd ed.* Boston: Library of Congress Cataloging.

Kaur, S., & Maini, R. 2016. Analysis of Various Software Metrics Used To Defect Bad Smells. *The International Journal of Engineering and Science (IJES)*, 14-20.

Madeyski, L., & Jureczko, M. 2014. Which Process Metrics Can Significantly Improve Defect Prediction Models? An Empirical Study. *Software Qual J*, 393-422.

McCabe, T. J. 1976. A Complexity Measure. *IEE Transactions on Software Engineering*, 308-320.

Montolalu , C. E., & Langi, A. Y. 2018. Pengaruh Pelatihan Dasar Komputer dan Teknologi Informasi bagi Guru-Guru dengan Uji-T Berpasangan (Paired Sample T-Test). *Jurnal Matematika dan Aplikasi deCartesiaN*, 44-46.

Nindhra, S., & Dondeti, J. 2012. Black Box and White Box Testing Techniques-A Literature Review . *International Journal of Embedded System and Applications (IJESA)*, 29-50.

Nuris, M. 2015. *white box testing pada penilaian pembelajaran.* Malang: UIN Maulana Malik Ibrahim.

Pranata , F. N., Pradana, F., & Kurniawan, T. A. (2016). Pengembangan Sisem Perhitungan Kompleksitas Kode Sumber Berdasarkan Metrik Helstead dan Cyclomatic Complexity. *SENTRIN*, 27-35.

Priyambadha, B., & Rochimah, S. 2013. Kuantifikasi Pengaruh Kloning dan Kompleksitas Kode Terhadap Cacat pada Evolusi Perangkat Lunak. *JUTI*, 17-23.

Rawat, M. S., Mittal, A., & Dubey, S. K. 2012. Survey on Impact of Software Metrics on Software Quality. *International of Advanced Computer Science and Applications*, 137-141.

Rerung, R. R. 2018. *Pemrograman Web Dasar.* Yogyakarta: Deepublish.

Sabarudin, R. 2017. *Prediksi Cacat Sofware Menggunakan Resempling Berbasis C5.0 Untuk Menangani Ketidakseimbangan Kelas.* Jakarta: STIMIK Nusa Mandiri.

Sehgal, R., & Mehrotra, D. 2015. Predicting Faults before Testing Phase using Halstead's Metrics . *International Journal of Software Engineering and Its Applications* , 135-142.

Sharma, R., & Sangwan , M. 2016. Software Testing Principles and Strategies. *International Journal of Computer Science and Mobile Computing*, 218-223.

Strangio, M. 2009. *Recent Advances in Technologies.* Vokuvar: In-Teh.

Suffian, M. D., & Ibrahim, S. 2012. APrediction Model for System Testing Defects using Regression Analysis . *International Journal of Soft Computing And Software Engineering (JSCSE)* , 55-68.

Suparto. 2014. Analisis Korelasi Variabel- Variabel yang Mempengaruhi Siswa Dalam Memilih Perguruan Tinggi. *IPTEK* .

Susanto, M. I., Darwiyanto, E., & Wisudawan, G. A. 2015. Pengukuran Software Metrics Terhadap Implementasi Framework Laravel pada Pembangunan Aplikasi Berbasis Web . *e-Proceeding of Engineering*, 7731-7738.

Tan, R., & Wijayanto, M. 2018. *Pemrograman Web dengan PHP*. Robby Tan.

Tashtoush, Y., Al-Maolegi, M., & Arkok, B. 2014. The Correlation among Software Complexity Metrics with Case Study . *International Journal of Advanced Computer Research*, 414-419.

V, V., & Jalote, P. 2005. List of Common Bugs and Programming Practices to avoid them. 1-25.

Zhang, H., Zhang, X., & Gu, M. (2007). Predicting Defective Software Components from Code Complexity Measures. *13th IEEE International Symposium on Pacific Rim Dependable Computing*, 93-96.